

Introduction à la programmation de tablettes Android avec Android Studio

Serge Tahé, IstiA - université d'Angers
août 2016

Table des matières

1 APPRENTISSAGE DE LA PROGRAMMATION ANDROID.....	9
 1.1 INTRODUCTION.....	9
1.1.1 CONTENU.....	9
1.1.2 PRÉ-REQUIS.....	10
1.1.3 LES OUTILS UTILISÉS.....	10
 1.2 EXEMPLE-01 : IMPORTATION D'UN EXEMPLE ANDROID.....	11
1.2.1 CRÉATION DU PROJET.....	11
1.2.2 QUELQUES POINTS SUR L'IDE.....	16
1.2.2.1 Les perspectives.....	16
1.2.2.2 Gestion de l'exécution.....	17
1.2.2.3 Gestion du cache.....	17
1.2.2.4 Gestion des logs.....	18
1.2.2.5 Gestion de l'émulateur [Genymotion].....	21
1.2.2.6 Gestion du binaire APK créé.....	21
 1.3 EXEMPLE-02 : UN PROJET ANDROID BASIQUE.....	23
1.3.1 CONFIGURATION GRADLE.....	25
1.3.2 LE MANIFESTE DE L'APPLICATION.....	26
1.3.3 L'ACTIVITÉ PRINCIPALE.....	29
1.3.4 EXÉCUTION DE L'APPLICATION.....	32
1.3.5 LE CYCLE DE VIE D'UNE ACTIVITÉ.....	34
 1.4 EXEMPLE-03 : RÉÉCRITURE DU PROJET [EXEMPLE-02] AVEC LA BIBLIOTHÈQUE [ANDROID ANNOTATIONS].....	36
 1.5 EXEMPLE-04 : VUES ET ÉVÉNEMENTS.....	42
1.5.1 CRÉATION DU PROJET.....	42
1.5.2 CONSTRUIRE UNE VUE.....	42
1.5.3 GESTION DES ÉVÉNEMENTS.....	49
 1.6 EXEMPLE-05 : NAVIGATION ENTRE VUES.....	51
1.6.1 CRÉATION DU PROJET.....	51
1.6.2 AJOUT D'UNE SECONDE ACTIVITÉ.....	51
1.6.3 NAVIGATION DE LA VUE N° 1 À LA VUE N° 2.....	54
1.6.4 CONSTRUCTION DE LA VUE N° 2.....	55
1.6.5 L'ACTIVITÉ [SECONDACTIVITY].....	57
1.6.6 NAVIGATION DE LA VUE N° 2 VERS LA VUE N° 1.....	59
1.6.7 CYCLE DE VIE DES ACTIVITÉS.....	60
 1.7 EXEMPLE-06 : NAVIGATION PAR ONGLETS.....	62
1.7.1 CRÉATION DU PROJET.....	62
1.7.2 CONFIGURATION GRADLE.....	65
1.7.3 LA VUE [ACTIVITY_MAIN].....	65
1.7.4 L'ACTIVITÉ.....	68
1.7.4.1 La gestion des fragments et des onglets.....	68
1.7.4.2 Les fragments affichés.....	70
1.7.4.3 Gestion du menu.....	71
1.7.4.4 Le bouton flottant.....	73
1.7.5 EXÉCUTION DU PROJET.....	74
1.7.6 CYCLE DE VIE DES FRAGMENTS.....	75
 1.8 EXEMPLE-07 : EXEMPLE-06 RÉÉCRIT AVEC LA BIBLIOTHÈQUE [AA].....	79
1.8.1 CRÉATION DU PROJET.....	79
1.8.2 CONFIGURATION GRADLE.....	79
1.8.3 AJOUT DES PREMIÈRES ANNOTATIONS AA.....	80
1.8.4 RÉÉCRITURE DES FRAGMENTS.....	81
1.8.5 EXAMEN DES LOGS.....	84
1.8.6 ONDESTROYVIEW.....	86
1.8.7 SETUSERVISIBLEHINT.....	88
1.8.8 SETOFFSCREENPAGELIMIT.....	89
1.8.9 ONDESTROY.....	92
 1.9 EXEMPLE-08 : MISE À JOUR D'UN FRAGMENT AVEC UNE ADJACENCE DE FRAGMENTS VARIABLE.....	94
1.9.1 CRÉATION DU PROJET.....	94
1.9.2 RÉÉCRITURE DU FRAGMENT [PLACEHOLDERFRAGMENT].....	94
1.9.3 COMMUNICATION INTER-FRAGMENTS.....	101
 1.10 EXEMPLE-09 : COMMUNICATION INTER-FRAGMENTS, SWIPE ET SCROLLING.....	104
1.10.1 CRÉATION DU PROJET.....	104
1.10.2 LA SESSION.....	104

1.10.3 L'ACTIVITÉ [MAINACTIVITY].....	105
1.10.4 LE FRAGMENT [PLACEHOLDERFRAGMENT].....	105
1.10.5 DÉSACTIVER LE SWIPE OU BALAYAGE.....	106
1.10.6 DÉSACTIVER LE SCROLLING ENTRE FRAGMENTS.....	108
1.10.7 UN NOUVEAU FRAGMENT.....	109
1.10.8 FAIRE DÉRIVER TOUS LES FRAGMENTS D'UNE MÊME CLASSE ABSTRAITE.....	111
1.11 EXEMPLE-10 : FAIRE DÉRIVER TOUS LES FRAGMENTS D'UNE CLASSE ABSTRAITE.....	112
1.11.1 CRÉATION DU PROJET.....	112
1.11.2 GESTION DU MODE DEBUG.....	112
1.11.3 LA CLASSE ABSTRAITE PARENTE DE TOUS LES FRAGMENTS.....	112
1.11.4 LA CLASSE [PLACEHOLDERFRAGMENT].....	115
1.11.5 LA CLASSE [VUE1FRAGMENT].....	116
1.11.6 ASSOCIATION ONGLETS / FRAGMENTS.....	117
1.12 EXEMPLE-11 : ONGLETS DISSOCIÉS DES FRAGMENTS.....	121
1.12.1 CRÉATION DU PROJET.....	121
1.12.2 OBJECTIFS.....	121
1.12.3 LA SESSION.....	122
1.12.4 LE MENU.....	122
1.12.5 LA CLASSE [MAINACTIVITY].....	123
1.12.6 AMÉLIORATIONS.....	125
1.13 EXEMPLE-12 : CODIFIER LES RELATIONS ENTRE ACTIVITÉ ET FRAGMENTS.....	126
1.13.1 CRÉATION DU PROJET.....	126
1.13.2 L'INTERFACE [IMAINACTIVITY].....	126
1.13.3 LA CLASSE ABSTRAITE [ABSTRACTFRAGMENT].....	127
1.13.4 MODIFICATION DU GESTIONNAIRE DE FRAGMENTS.....	127
1.13.5 MODIFICATION DE LA CLASSE [MAINACTIVITY].....	128
1.13.6 MODIFICATION DE L'AFFICHAGE DES FRAGMENTS DANS [MAINACTIVITY].....	128
1.13.7 CONCLUSION.....	128
1.14 EXEMPLE-13 : EXEMPLE-05 AVEC DES FRAGMENTS.....	129
1.14.1 CRÉATION DU PROJET.....	129
1.14.2 STRUCTURATION DU PROJET.....	129
1.14.3 NETTOYAGE DU PROJET.....	131
1.14.3.1 Nettoyage des fragments.....	131
1.14.3.2 Nettoyage de la session.....	132
1.14.3.3 Suppression des onglets, du bouton flottant et du menu.....	133
1.14.4 CRÉATION DES FRAGMENTS ET DES VUES ASSOCIÉES.....	137
1.14.5 MISE EN PLACE DES FRAGMENTS ET DE LA NAVIGATION ENTRE-EUX.....	138
1.14.6 DÉFINITION DE LA SESSION.....	140
1.14.7 ECRITURE FINALE DES FRAGMENTS.....	140
1.14.8 GESTION DU CYCLE DE VIE DES FRAGMENTS.....	142
1.14.9 CONCLUSION.....	143
1.15 EXEMPLE-14 : UNE ARCHITECTURE À DEUX COUCHES.....	144
1.15.1 CRÉATION DU PROJET.....	144
1.15.2 LA VUE [VUE1].....	144
1.15.3 LA SESSION.....	148
1.15.4 LE FRAGMENT [VUE1FRAGMENT].....	148
1.15.5 LA COUCHE [MÉTIER].....	150
1.15.6 L'ACTIVITÉ [MAINACTIVITY] REVISITÉE.....	152
1.15.7 LE FRAGMENT [VUE1FRAGMENT] REVISITÉ.....	153
1.15.8 EXÉCUTION.....	156
1.16 EXEMPLE-15 : ARCHITECTURE CLIENT / SERVEUR.....	157
1.16.1 LE SERVEUR [WEB / jSON].....	157
1.16.1.1 Création du projet.....	157
1.16.1.2 Configuration Gradle.....	159
1.16.1.3 Configuration du projet.....	165
1.16.1.4 La couche [métier].....	166
1.16.1.5 Le service web / jSON.....	167
1.16.1.6 Configuration du projet Spring.....	171
1.16.1.7 Exécution du service web / jSON.....	172
1.16.1.8 Génération du jar exécutables du projet.....	176
1.16.1.9 Gestion des logs.....	177
1.16.2 LE CLIENT ANDROID DU SERVEUR WEB / jSON.....	178
1.16.2.1 Création du projet.....	178

<u>1.16.2.2</u> Configuration Gradle.....	179
<u>1.16.2.3</u> Le manifeste de l'application Android.....	180
<u>1.16.2.4</u> La couche [DAO].....	181
<u>1.16.2.4.1</u> L'interface [IDao] de la couche [DAO].....	181
<u>1.16.2.4.2</u> L'interface [WebClient].....	182
<u>1.16.2.4.3</u> La classe [Response].....	183
<u>1.16.2.4.4</u> Implémentation de la couche [DAO].....	183
<u>1.16.2.5</u> Le package [architecture].....	186
<u>1.16.2.5.1</u> L'interface [IMainActivity].....	187
<u>1.16.2.5.2</u> La classe [Utils].....	187
<u>1.16.2.5.3</u> La classe abstraite [AbstractFragment].....	188
<u>1.16.2.6</u> La vue.....	188
<u>1.16.2.6.1</u> La vue [vue1.xml].....	188
<u>1.16.2.6.2</u> Le fragment [Vue1Fragment].....	191
<u>1.16.2.7</u> L'activité [MainActivity].....	195
<u>1.16.2.7.1</u> La vue [activity-main.xml].....	195
<u>1.16.2.7.2</u> L'activité [MainActivity].....	196
<u>1.16.2.8</u> Exécution du projet.....	196
<u>1.16.2.9</u> Gestion de l'annulation.....	198
1.17 EXEMPLE-16 : GÉRER L'ASYNCRONISME AVEC RXANDROID.....	201
<u>1.17.1</u> CRÉATION DU PROJET.....	201
<u>1.17.2</u> CONFIGURATION GRADLE.....	201
<u>1.17.3</u> LA COUCHE [DAO].....	202
<u>1.17.4</u> L'INTERFACE [IDAO].....	202
<u>1.17.5</u> LA CLASSE [ABSTRACTDAO].....	202
<u>1.17.6</u> LA CLASSE [DAO].....	204
<u>1.17.7</u> LA CLASSE [MAINACTIVITY].....	204
<u>1.17.8</u> LA CLASSE [VUE1FRAGMENT].....	205
<u>1.17.9</u> EXÉCUTION.....	207
<u>1.17.10</u> GESTION DE L'ANNULATION.....	207
<u>1.17.11</u> CONCLUSION.....	208
1.18 EXEMPLE-17 : COMPOSANTS DE SAISIE DE DONNÉES.....	210
<u>1.18.1</u> CRÉATION DU PROJET.....	210
<u>1.18.2</u> LA VUE XML DU FORMULAIRE.....	210
<u>1.18.3</u> LES CHAÎNES DE CARACTÈRES DU FORMULAIRE.....	215
<u>1.18.4</u> LE FRAGMENT DU FORMULAIRE.....	215
<u>1.18.5</u> EXÉCUTION DU PROJET.....	218
1.19 EXEMPLE-18 : UTILISATION D'UN PATRON DE VUES.....	219
<u>1.19.1</u> CRÉATION DU PROJET.....	219
<u>1.19.2</u> LE PATRON DES VUES.....	219
1.20 EXEMPLE-19 : LE COMPOSANT [LISTVIEW].....	223
<u>1.20.1</u> CRÉATION DU PROJET.....	223
<u>1.20.2</u> LA SESSION.....	224
<u>1.20.3</u> L'ACTIVITÉ [MAINACTIVITY].....	224
<u>1.20.4</u> LA VUE [VUE1] INITIALE.....	225
<u>1.20.5</u> LA VUE RÉPÉTÉE PAR LE [LISTVIEW].....	226
<u>1.20.6</u> LE FRAGMENT [VUE1FRAGMENT].....	227
<u>1.20.7</u> L'ADAPTATEUR [LISTADAPTER] DU [LISTVIEW].....	228
<u>1.20.8</u> RETIRER UN ÉLÉMENT DE LA LISTE.....	229
<u>1.20.9</u> LA VUE XML [VUE2].....	230
<u>1.20.10</u> LE FRAGMENT [VUE2FRAGMENT].....	231
<u>1.20.11</u> EXÉCUTION.....	232
<u>1.20.12</u> AMÉLIORATION.....	232
1.21 EXEMPLE-20 : UTILISER UN MENU.....	234
<u>1.21.1</u> CRÉATION DU PROJET.....	234
<u>1.21.2</u> LA DÉFINITION XML DES MENUS.....	234
<u>1.21.3</u> LA GESTION DU MENU DANS LA CLASSE ABSTRAITE [ABSTRACTFRAGMENT].....	235
<u>1.21.4</u> LA GESTION DU MENU DANS LE FRAGMENT [VUE1FRAGMENT].....	238
<u>1.21.5</u> LA GESTION DU MENU DANS LE FRAGMENT [VUE2FRAGMENT].....	238
<u>1.21.6</u> EXÉCUTION.....	239
1.22 EXEMPLE-21 : REFACTORING DE LA CLASSE ABSTRAITE [ABSTRACTFRAGMENT].....	240
<u>1.22.1</u> CRÉATION DU PROJET.....	240
<u>1.22.2</u> LE MENU DES FRAGMENTS.....	240

1.22.3	LES FRAGMENTS.....	241
1.22.4	EXÉCUTION.....	241
1.23	EXEMPLE-22 : SAUVEGARDE / RESTAURATION DE L'ÉTAT DE L'ACTIVITÉ ET DES FRAGMENTS.....	241
1.23.1	LE PROBLÈME.....	241
1.23.2	LES MÉTHODES DE SAUVEGARDE / RESTAURATION DE L'ACTIVITÉ ET DES FRAGMENTS.....	243
1.23.2.1	Solution 1 : sauvegarde manuelle.....	243
1.23.2.2	Solution 2 : sauvegarde automatique.....	244
1.23.3	LA MÉTHODE DE SAUVEGARDE / RESTAURATION DU PROJET [EXEMPLE-22].....	244
1.23.4	SAUVEGARDE DU FRAGMENT [VUE1FRAGMENT].....	247
1.23.5	SAUVEGARDE DU FRAGMENT [PLACEHOLDERFRAGMENT].....	247
1.23.6	RESTAURATION DU FRAGMENT [VUE1FRAGMENT].....	248
1.23.7	RESTAURATION DU FRAGMENT [PLACEHOLDERFRAGMENT].....	248
1.23.8	GESTION DES ONGLETS.....	249
2	SQUELETTE D'UN CLIENT ANDROID COMMUNICANT AVEC UN SERVICE WEB / JSON.....	251
2.1	ARCHITECTURE DU CLIENT ANDROID.....	251
2.2	LA CONFIGURATION GRADLE.....	252
2.3	LE MANIFESTE DE L'APPLICATION.....	253
2.4	L'ORGANISATION DU CODE JAVA.....	254
2.5	ÉLÉMENTS DE L'ACTIVITÉ.....	254
2.5.1	LA VUE ASSOCIÉE À L'ACTIVITÉ.....	255
2.5.2	LE CONTENEUR DE FRAGMENTS [MYPAGER].....	255
2.5.3	LA CLASSE [CORESTATE].....	256
2.5.4	L'INTERFACE [IMAINACTIVITY].....	257
2.5.5	L'INTERFACE [IDAO].....	258
2.5.6	LA SESSION.....	259
2.5.7	LA CLASSE ABSTRAITE [ABSTRACTACTIVITY].....	262
2.5.7.1	Squelette.....	262
2.5.7.2	Implémenter l'interface [IMainActivity].....	263
2.5.7.3	Sauvegarde de l'état de l'activité et de ses fragments.....	263
2.5.7.4	Restauration de l'état de l'activité et de ses fragments.....	264
2.5.7.5	Initialisation de la couche [DAO].....	264
2.5.7.6	Initialisation de la vue associée à l'activité.....	265
2.5.7.7	Gestion des onglets.....	266
2.5.7.8	Le gestionnaire d'onglets [CustomTabLayout].....	267
2.5.7.9	Dernières initialisations.....	268
2.5.7.10	Gestion de l'image d'attente.....	269
2.5.7.11	Implémentation de l'interface [IDao].....	269
2.5.7.12	Implémentation du gestionnaire de fragments.....	270
2.5.7.13	La méthode [onResume].....	270
2.5.7.14	Résumé.....	271
2.5.8	L'ACTIVITÉ [MAINACTIVITY].....	271
2.6	LA COUCHE [DAO].....	272
2.6.1	L'INTERFACE IDAO.....	273
2.6.2	L'INTERFACE [WEBCLIENT].....	273
2.6.3	L'INTERCEPTEUR D'AUTHENTIFICATION[MYAUTHINTERCEPTOR].....	274
2.6.4	LA CLASSE [ABSTRACTDAO].....	275
2.6.5	LA CLASSE [DAO].....	276
2.7	LES FRAGMENTS.....	278
2.7.1	LA CLASSE [MENUITEMSTATE].....	278
2.7.2	LA CLASSE [UTILS].....	278
2.7.3	LA CLASSE PARENT [ABSTRACTFRAGMENT].....	279
2.7.3.1	Le squelette.....	279
2.7.3.2	Le constructeur.....	280
2.7.3.3	Gestion du menu.....	281
2.7.3.4	Gestion de l'attente de la fin d'une tâche asynchrone.....	282
2.7.3.5	Gestion des exceptions.....	283
2.7.3.6	Gestion des opérations asynchrones.....	283
2.7.3.7	Gestion du cycle de vie du fragment.....	285
2.7.3.8	Mise à jour du fragment.....	287
2.7.4	UN EXEMPLE DE FRAGMENT.....	292
2.8	EXERCICES D'ILLUSTRATION.....	294
2.8.1	EXEMPLE-17B.....	294
2.8.1.1	Le projet [Exemple-17B].....	295

<u>2.8.1.2</u> L'état du fragment [Vue1Fragment].....	297
<u>2.8.1.3</u> Personnalisation du projet.....	297
<u>2.8.1.4</u> L'activité [MainActivity].....	299
<u>2.8.1.5</u> L'état du fragment [FragmentState].....	300
<u>2.8.1.6</u> Le fragment [AbstractFragment].....	301
<u>2.8.1.7</u> Tests.....	304
<u>2.8.2</u> EXEMPLE-23 : CLIENT MÉTÉO.....	304
<u>2.8.2.1</u> Le projet.....	304
<u>2.8.2.2</u> Personnalisation du projet.....	304
<u>2.8.2.3</u> La couche [DAO].....	306
<u>2.8.2.4</u> L'activité [MainActivity].....	309
<u>2.8.2.5</u> Le fragment[MeteoFragment].....	310
<u>2.8.2.6</u> Tests.....	314
<u>2.8.3</u> EXEMPLE-16B.....	316
<u>2.8.3.1</u> Le projet Exemple-16B.....	317
<u>2.8.3.2</u> Création d'un état pour le fragment [Vue1Fragment].....	319
<u>2.8.3.3</u> Personnalisation du projet.....	319
<u>2.8.3.4</u> La couche [DAO].....	321
<u>2.8.3.5</u> L'activité [MainActivity].....	324
<u>2.8.3.6</u> L'état du fragment[Vue1Fragment].....	325
<u>2.8.3.7</u> Le fragment[Vue1Fragment].....	326
<u>2.8.3.7.1</u> Gestion du clic sur le bouton [Exécuter].....	326
<u>2.8.3.7.2</u> Le cycle de vie du fragment.....	328
<u>2.8.3.8</u> Les tests.....	329
<u>2.8.4</u> EXEMPLE-22B.....	329
<u>2.8.4.1</u> Personnalisation du projet.....	331
<u>2.8.4.2</u> L'activité [MainActivity].....	333
<u>2.8.4.2.1</u> Implémentation des méthodes de la classe parent.....	333
<u>2.8.4.2.2</u> Gestion des onglets.....	334
<u>2.8.4.2.3</u> Gestion du menu.....	335
<u>2.8.4.3</u> Le fragment [Vue1Fragment].....	337
<u>2.8.4.4</u> L'état [PlaceHolderFragmentState].....	338
<u>2.8.4.5</u> Le fragment [PlaceHolderFragment].....	339
<u>2.8.4.6</u> Tests.....	340
<u>2.9</u> CONCLUSION.....	340
3 ÉTUDE DE CAS - GESTION DE RENDEZ-VOUS.....	341
<u>3.1</u> LE PROJET.....	341
<u>3.2</u> LES VUES DU CLIENT ANDROID.....	341
<u>3.3</u> L'ARCHITECTURE DU PROJET.....	343
<u>3.4</u> LA BASE DE DONNÉES.....	344
<u>3.4.1</u> LA TABLE [MEDECINS].....	344
<u>3.4.2</u> LA TABLE [CLIENTS].....	345
<u>3.4.3</u> LA TABLE [CRENEAUX].....	345
<u>3.4.4</u> LA TABLE [RV].....	346
<u>3.4.5</u> GÉNÉRATION DE LA BASE.....	346
<u>3.5</u> LE SERVEUR WEB / JSON.....	348
<u>3.5.1</u> MISE EN OEUVRE.....	349
<u>3.5.2</u> SÉCURISATION DU SERVICE WEB.....	350
<u>3.5.3</u> LISTE DES MÉDECINS.....	350
<u>3.5.4</u> LISTE DES CLIENTS.....	353
<u>3.5.5</u> LISTE DES CRÉNEAUX D'UN MÉDECIN.....	354
<u>3.5.6</u> LISTE DES RENDEZ-VOUS D'UN MÉDECIN.....	354
<u>3.5.7</u> L'AGENDA D'UN MÉDECIN.....	355
<u>3.5.8</u> OBTENIR UN MÉDECIN PAR SON IDENTIFIANT.....	355
<u>3.5.9</u> OBTENIR UN CLIENT PAR SON IDENTIFIANT.....	356
<u>3.5.10</u> OBTENIR UN CRÉNEAU PAR SON IDENTIFIANT.....	356
<u>3.5.11</u> OBTENIR UN RENDEZ-VOUS PAR SON IDENTIFIANT.....	356
<u>3.5.12</u> AJOUTER UN RENDEZ-VOUS.....	356
<u>3.5.13</u> SUPPRIMER UN RENDEZ-VOUS.....	358
<u>3.6</u> LE CLIENT ANDROID.....	359
<u>3.6.1</u> ARCHITECTURE DU PROJET ANDROID STUDIO.....	359
<u>3.6.2</u> PERSONNALISATION DU PROJET.....	360
<u>3.6.3</u> LA COUCHE [DAO].....	362

<u>3.6.3.1</u> Implémentation des échanges client / serveur.....	363
<u>3.6.3.2</u> L'interface [IDao].....	364
<u>3.6.3.3</u> La classe [Dao].....	365
<u>3.6.4</u> L'ACTIVITÉ [MAINACTIVITY].....	369
<u>3.6.5</u> LA SESSION.....	371
<u>3.6.6</u> GESTION DE LA VUE DE CONFIGURATION.....	372
<u>3.6.6.1</u> La vue.....	372
<u>3.6.6.2</u> Le fragment.....	372
<u>3.6.6.3</u> Gestion du cycle de vie du fragment.....	376
<u>3.6.7</u> GESTION DE LA VUE D'ACCUEIL.....	377
<u>3.6.7.1</u> La vue.....	377
<u>3.6.7.2</u> Le fragment.....	378
<u>3.6.7.3</u> Gestion du cycle de vie du fragment.....	381
<u>3.6.8</u> GESTION DE LA VUE AGENDA.....	383
<u>3.6.8.1</u> La vue.....	383
<u>3.6.8.2</u> Le fragment.....	384
<u>3.6.8.2.1</u> Méthode [updateAgenda].....	385
<u>3.6.8.2.2</u> Méthode [doSupprimer].....	389
<u>3.6.8.2.3</u> Méthode [doAnnuler].....	391
<u>3.6.8.2.4</u> Option de menu [Retour à la configuration].....	391
<u>3.6.8.2.5</u> Option de menu [Retour à l'accueil].....	391
<u>3.6.8.3</u> Gestion du cycle de vie du fragment.....	391
<u>3.6.9</u> GESTION DE LA VUE D'AJOUT D'UN RENDEZ-VOUS.....	393
<u>3.6.9.1</u> La vue.....	393
<u>3.6.9.2</u> Le fragment.....	394
<u>3.6.9.3</u> Gestion du cycle de vie du fragment.....	397
<u>3.7</u> EXÉCUTION.....	399
<u>4</u> TP 1 : GESTION BASIQUE D'UNE FICHE DE PAIE.....	401
<u>4.1</u> INTRODUCTION.....	401
<u>4.2</u> LA BASE DE DONNÉES.....	401
<u>4.2.1</u> DÉFINITION.....	401
<u>4.2.2</u> GÉNÉRATION.....	402
<u>4.2.3</u> MODÉLISATION JAVA DE LA BASE.....	403
<u>4.3</u> INSTALLATION DU SERVEUR WEB / JSON.....	404
<u>4.3.1</u> INSTALLATION.....	405
<u>4.3.2</u> LES URL DU SERVICE WEB/JSON.....	406
<u>4.3.3</u> LES RÉPONSES JSON DU SERVICE WEB/JSON.....	407
<u>4.4</u> TESTS DU CLIENT ANDROID.....	409
<u>4.5</u> TRAVAIL À FAIRE.....	412
<u>5</u> TP 2 - PILOTER DES ARDUINOS AVEC UNE TABLETTE ANDROID.....	414
<u>5.1</u> ARCHITECTURE DU PROJET.....	414
<u>5.2</u> LE MATÉRIEL.....	414
<u>5.2.1</u> L'ARDUINO.....	414
<u>5.2.2</u> LA TABLETTE.....	416
<u>5.2.3</u> L'ÉMULATEUR [GENYMOTION].....	417
<u>5.3</u> PROGRAMMATION DES ARDUINOS.....	417
<u>5.4</u> LE SERVEUR WEB / JSON.....	421
<u>5.4.1</u> INSTALLATION.....	421
<u>5.4.2</u> LES URL EXPOSÉES PAR LE SERVICE WEB / JSON.....	423
<u>5.4.3</u> LES TESTS DU SERVICE WEB / JSON.....	425
<u>5.5</u> TESTS DU CLIENT ANDROID.....	428
<u>5.6</u> LE CLIENT ANDROID DU SERVICE WEB / JSON.....	432
<u>5.6.1</u> L'ARCHITECTURE DU CLIENT.....	432
<u>5.6.2</u> LE PROJET ANDROID STUDIO DU CLIENT.....	432
<u>5.6.3</u> LES CINQ VUES XML.....	432
<u>5.6.4</u> LE MENU DES FRAGMENTS.....	433
<u>5.6.5</u> LES CINQ FRAGMENTS DE L'APPLICATION.....	434
<u>5.6.6</u> LES ÉTATS DES FRAGMENTS.....	435
<u>5.6.7</u> PERSONNALISATION DU PROJET.....	435
<u>5.6.7.1</u> L'interface [IMainActivity].....	435
<u>5.6.7.2</u> La classe [CoreState].....	436
<u>5.6.8</u> LA CLASSE [MAINACTIVITY].....	437
<u>5.6.9</u> LA VUE XML [CONFIG].....	439

5.6.10 LE FRAGMENT [CONFIGFRAGMENT].....	442
 5.6.10.1 Le bouton [Rafraîchir].....	444
 5.6.10.2 Vérification des saisies.....	445
 5.6.10.3 Affichage de la liste des Arduinos.....	446
 5.6.10.4 Un patron pour afficher un Arduino.....	449
 5.6.10.5 La session.....	453
 5.6.10.6 Gestion de l'état du fragment.....	454
 5.6.10.7 Gestion du cycle de vie du fragment.....	456
 5.6.10.8 Amélioration du code.....	458
 5.6.11 COMMUNICATION ENTRE VUES.....	459
 5.6.12 LA COUCHE [DAO].....	462
 5.6.12.1 L'interface IDao.....	462
 5.6.12.2 L'interface [WebClient].....	463
 5.6.12.3 La classe [Dao].....	464
 5.6.13 L'ACTIVITÉ [MAINACTIVITY].....	466
 5.6.14 LE FRAGMENT [CONFIGFRAGMENT] REVISITÉ.....	466
5.7 TRAVAIL À FAIRE.....	468
6 ANNEXES.....	470
 6.1 INTALLATION DE L'IDE ARDUINO.....	470
 6.2 INTALLATION DU PILOTE (DRIVER) DE L'ARDUINO.....	470
 6.3 TESTS DE L'IDE.....	470
 6.4 CONNEXION RÉSEAU DE L'ARDUINO.....	472
 6.5 TEST D'UNE APPLICATION RÉSEAU.....	473
 6.6 LA BIBLIOTHÈQUE AJSON.....	476
 6.7 LA TABLETTE ANDROID.....	477
 6.8 INSTALLATION D'UN JDK.....	479
 6.9 INSTALLATION DU GESTIONNAIRE D'ÉMULATEURS GENYMOTION.....	480
 6.10 INSTALLATION DE MAVEN.....	483
 6.11 INSTALLATION DE L'IDE ANDROID STUDIO.....	484
 6.12 UTILISATION DES EXEMPLES.....	490
 6.13 INSTALLATION DU PLUGIN CHROME [ADVANCED REST CLIENT].....	501
 6.14 GESTION DU JSON EN JAVA.....	502
 6.15 INSTALLATION DE [WAMPSERVER].....	504

1 Apprentissage de la programmation Android

1.1 Introduction

1.1.1 Contenu

Ce document est la réécriture de plusieurs documents existants :

1. [Android pour les développeurs J2EE](#) ;
2. [Introduction à la programmation de tablettes Android par l'exemple](#) ;
3. [Commander un Arduino avec une tablette Android](#) ;
4. [Introduction à la programmation de tablettes Android par l'exemple - version 2](#)

et introduit les nouveautés suivantes :

- le document 1 présentait une architecture appelée AVAT (Activité-Vues-Actions-Tâches) pour faciliter la programmation asynchrone dans une application Android. Dans ce document, la bibliothèque standard RxJava est utilisée pour gérer les actions asynchrones ;
- le document 2 utilisait l'IDE Eclipse avec un plugin Android. Ce document utilise Android Studio ;
- le document 3 est repris tel quel ;
- le document 4 utilisait la bibliothèque [Android Annotations] (AA) avec l'IDE IntelliJ IDEA Community Edition. Ce document reprend la totalité du document 4 avec les différences suivantes :
 - l'IDE est désormais Android Studio ;
 - le système de build est Gradle pour tous les projets client ou serveur (dans le document 4, on utilisait parfois Maven)
 - la programmation asynchrone est réalisée avec la bibliothèque RxJava (dans le document 4, on utilisait la bibliothèque AA) ;
- ce document explore des domaines pas ou peu abordés dans les documents précédents :
 - la notion d'adjacence de fragments ;
 - la sauvegarde / restauration de l'activité et de ses fragments ;
 - le cycle de vie des fragments ;

Enfin, il présente le squelette d'un client Android communiquant avec un service web / JSON dans lequel on factorise un grand nombre d'éléments que l'on retrouve régulièrement dans ce type de clients. Ce squelette est repris par tous les exemples à partir du chapitre 2. C'est la partie vraiment innovante du document.

Les exemples suivants sont présentés :

Exemple	Nature
1	Importation d'un projet Android existant
2	Un projet Android basique
3	Un projet [Android Annotations] basique
4	Vues et événements
5	Navigation entre vues
6	Navigation par onglets
7	Utilisation de la bibliothèque [Android Annotations] avec Gradle
8 à 12	Gestion des fragments d'une application Android
13	Navigation entre vues revisitée
14	Architecture à deux couches
15	Architecture client / serveur
16	Gérer l'asynchronisme avec RxJava
17, 17B	Composants de saisie de données
18	Utilisation d'un patron de vues

19	Le composant ListView
20	Utiliser un menu
21	Utilisation d'une classe parent pour les fragments
22, 22B	Sauvegarde et restauration de l'état de l'activité et des fragments
23	Client météo
Chap 2	Squelette d'un client Android communiquant avec un service web / JSON. On y factorise un grand nombre d'éléments que l'on retrouve régulièrement dans ce type de clients Android.
Chap 3	Gestion des rendez-vous d'un cabinet de médecins
Chap 4	Exercice d'application - Gestion d'une paie basique
Chap 5	Exercice d'application - Commande de cartes Arduino

Ce document est utilisé en dernière année de l'école d'ingénieurs IstiA de l'université d'Angers [istia.univ-angers.fr]. Cela explique le ton parfois un peu particulier du texte. Les deux exercices d'application sont des textes de TP dont on ne donne que les grandes lignes de la solution. Celle-ci est à construire par le lecteur.

Le code source des exemples est disponible à l'URL [<http://tahe.developpez.com/tutoriels-cours/programmation-android-avec-android-studio-debutant/documents/dvp-android-studio.rar>].

Ce document est un document de première formation à la programmation Android. Il ne se veut pas exhaustif. Il cible essentiellement les débutants.

Le site de référence pour la programmation Android est à l'URL [<http://developer.android.com/guide/components/index.html>]. C'est là qu'il faut aller pour avoir une vue d'ensemble de la programmation Android.

1.1.2 Pré-requis

Le pré-requis à une utilisation optimale du document est une bonne maîtrise du langage Java.

1.1.3 Les outils utilisés

Les exemples qui suivent ont été testés dans l'environnement suivant :

- machine Windows 10 pro 64 bits ;
- JDK 1.8 ;
- Android SDK API 23 ;
- Android Studio, version 2.1 ;
- Emulateur Genymotion, version 2.6.0 ;

Pour suivre ce document vous devez installer :

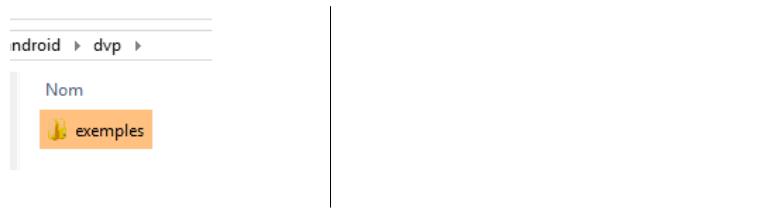
- un JDK (cf paragraphe 6.8, page 479) ;
- le gestionnaire d'émulateurs Android Genymotion (cf paragraphe 6.9, page 480) ;
- le gestionnaire de dépendances Maven (cf paragraphe 6.10, page 483) ;
- l'IDE [Android Studio] (cf paragraphe 6.11, page 484) ;

Les projets Android Studio des exemples sont disponibles à l'URL [<http://tahe.developpez.com/tutoriels-cours/programmation-android-avec-android-studio-debutant/documents/dvp-android-studio.rar>]. Pour les exécuter, vous devez suivre la procédure du paragraphe 6.12, page 490.

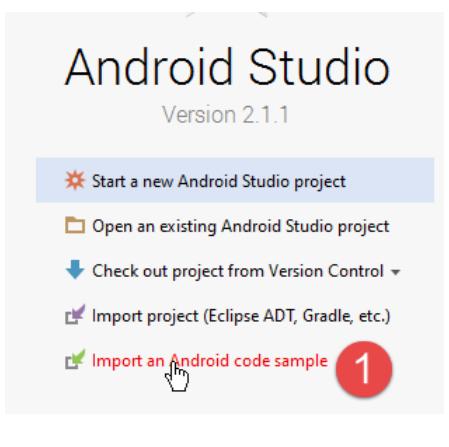
1.2 Exemple-01 : importation d'un exemple Android

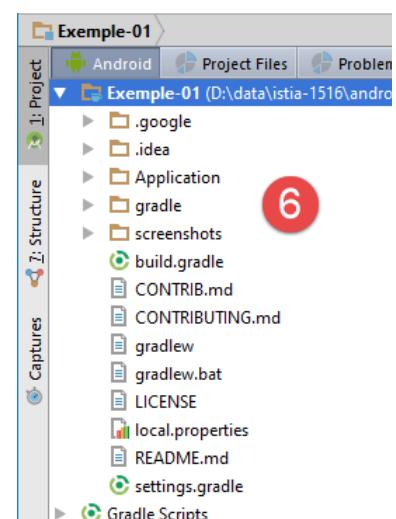
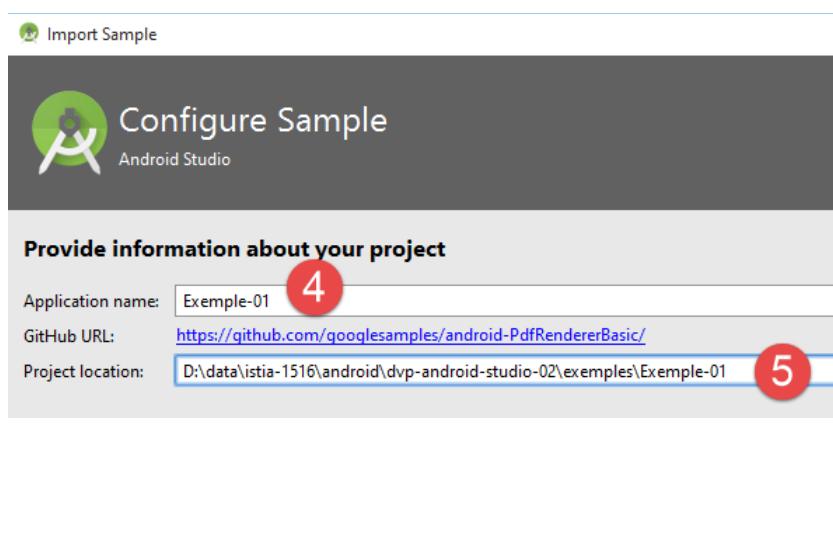
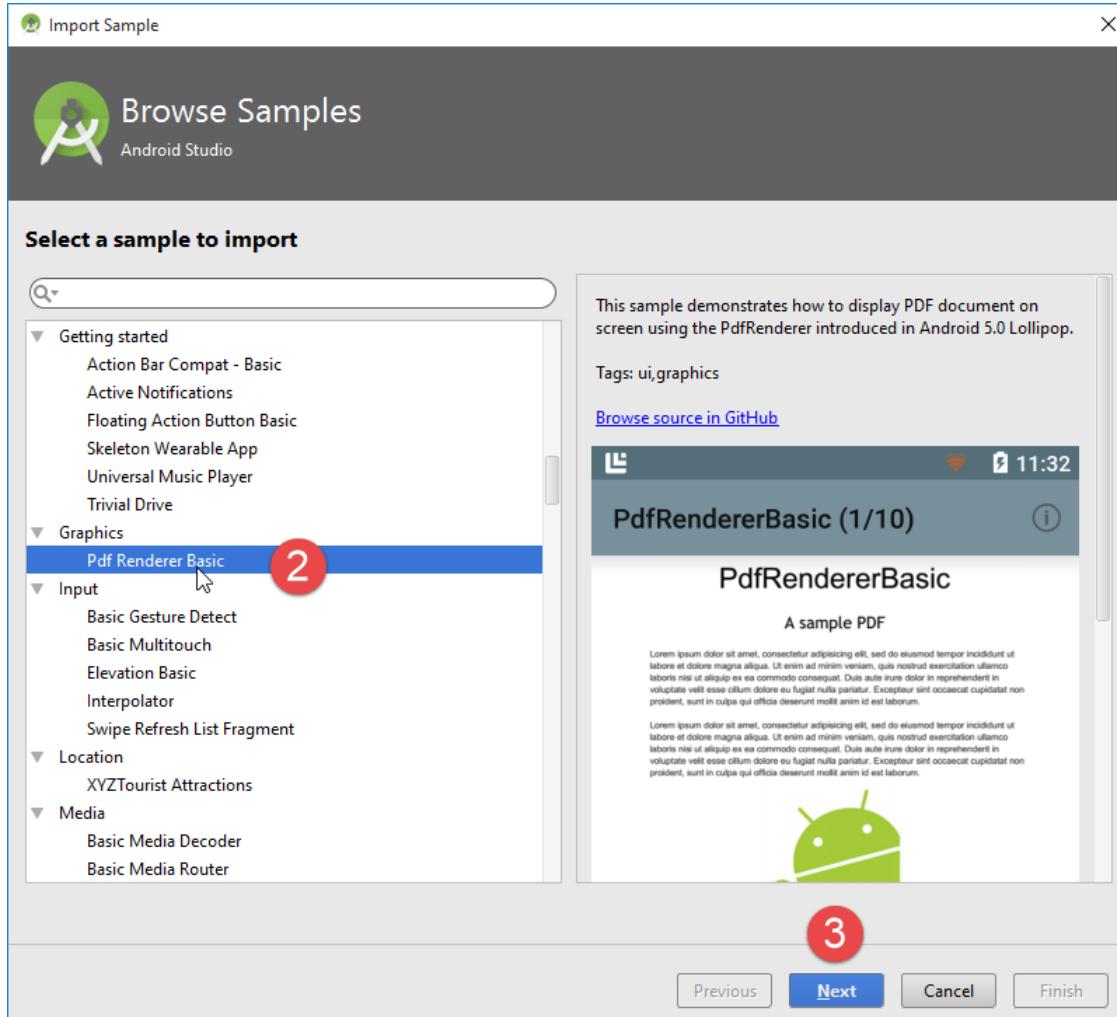
1.2.1 Crédit du projet

Créons avec Android Studio un premier projet Android. Tout d'abord créons un dossier [exemples] vide où seront placés tous nos projets :

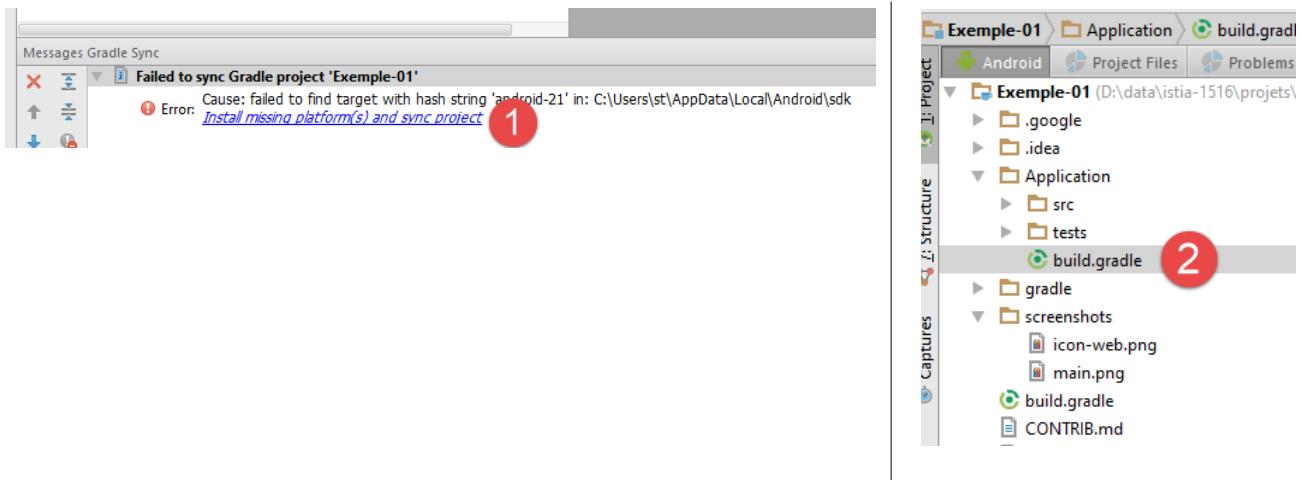


puis créons un projet avec Android Studio. Nous allons tout d'abord importer un des exemples livrés avec l'IDE [1-5] :





L'import du projet peut aboutir à des erreurs dues à l'inadéquation entre l'environnement utilisé lors de la création du projet et celui utilisé ici pour son exécution. C'est une occasion pour voir comment résoudre ce type d'erreurs. Ici, nous avons l'erreur suivante :



Le projet importé est configuré par le fichier [build.gradle] [2] suivant :

```

1. buildscript {
2.     repositories {
3.         jcenter()
4.     }
5.     dependencies {
6.         classpath 'com.android.tools.build:gradle:2.1.0'
7.     }
8. }
9.
10. apply plugin: 'com.android.application'
11.
12. repositories {
13.     jcenter()
14. }
15.
16. dependencies {
17.     compile "com.android.support:support-v4:23.3.0"
18.     compile "com.android.support:support-v13:23.3.0"
19.     compile "com.android.support:cardview-v7:23.3.0"
20. }
21.
22. // The sample build uses multiple directories to
23. // keep boilerplate and common code separate from
24. // the main sample code.
25. List<String> dirs = [
26.     'main',      // main sample code; look here for the interesting stuff.
27.     'common',    // components that are reused by multiple samples
28.     'template' ] // boilerplate code that is generated by the sample template process
29.
30. android {
31.     compileSdkVersion 21
32.     buildToolsVersion "23.0.3"
33.     defaultConfig {
34.         minSdkVersion 21
35.         targetSdkVersion 21
36.     }
37.     compileOptions {
38.         sourceCompatibility JavaVersion.VERSION_1_7
39.         targetCompatibility JavaVersion.VERSION_1_7
40.     }
41.     sourceSets {
42.         main {
43.             dirs.each { dir ->
44.                 java.srcDirs "src/${dir}/java"
45.                 res.srcDirs "src/${dir}/res"
46.             }
47.         }
48.         androidTest.setRoot('tests')
49.         androidTest.java.srcDirs = ['tests/src']
50.     }
51.
52.     aaptOptions {
53.         noCompress "pdf"
54.     }
55. }
```

- l'erreur signalée est dûe aux lignes 31, 34-35 : nous n'avons pas de SDK 21. Nous remplaçons cette version par la version 23 dont nous disposons.

Dans le fichier [build.gradle], Android Studio fait des suggestions comme ci-dessous :

```

17
18     dependencies {
19         compile "com.android.support:support-v4:23.3.0"
20         compile "com.android.support:support-v13:23.3.0"
21         compile "com.android.support:cardview-v7:23.3.0"
22     }
23
24     // The sample build uses multiple directories to
25     // keep boilerplate and common code separate from
26     // the main sample code.

```

A newer version of com.android.support:cardview-v7 than 23.3.0 is available: 23.4.0 [more...](#) (Ctrl+F1)

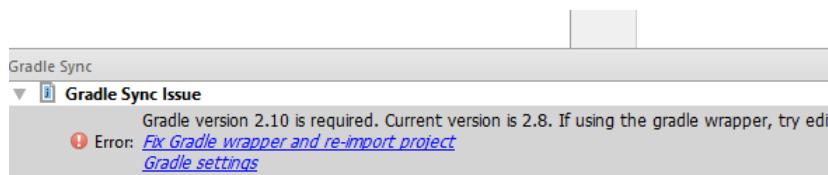
Pour accepter les suggestions, on fait [alt-entrée] sur la suggestion :

```

17
18     dependencies {
19         compile "com.android.support:support-v4:23.4.0"
20         compile "com.android.support:support-v13:23.4.0"
21         compile "com.android.support:cardview-v7:23.4.0"
22     }

```

On peut avoir également une erreur sur la version de Gradle :



Cette erreur vient d'une inadéquation entre la version de Gradle demandée par le fichier [build.gradle] du projet (la 2.10 ligne 6 ci-dessous) :

```

1. buildscript {
2.     repositories {
3.         jcenter()
4.     }
5.     dependencies {
6.         classpath 'com.android.tools.build:gradle:2.1.0'
7.     }
8. }

```

et celle inscrite dans le fichier [<projet>/gradle/wrapper/gradle-wrapper.properties] :

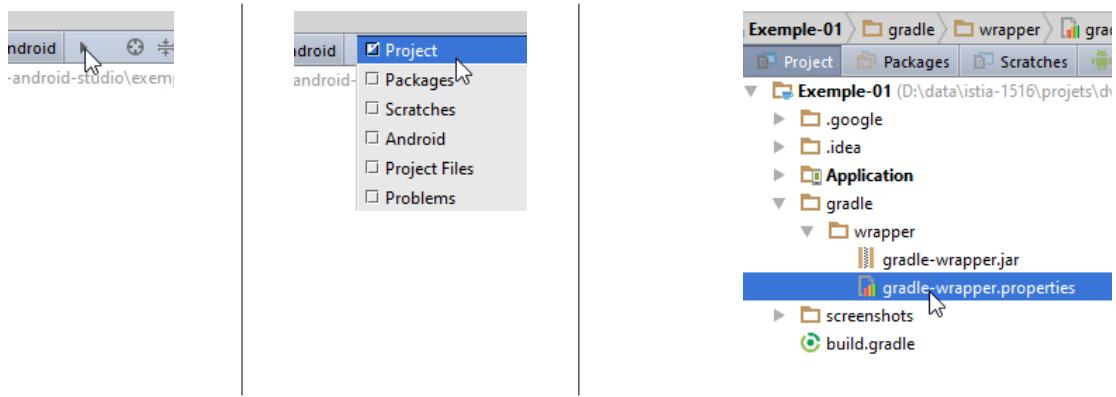
```

1. #Wed Apr 10 15:27:10 PDT 2013
2. distributionBase=GRADLE_USER_HOME
3. distributionPath=wrapper/dists
4. zipStoreBase=GRADLE_USER_HOME
5. zipStorePath=wrapper/dists
6. distributionUrl=https://services.gradle.org/distributions/gradle-2.8-all.zip

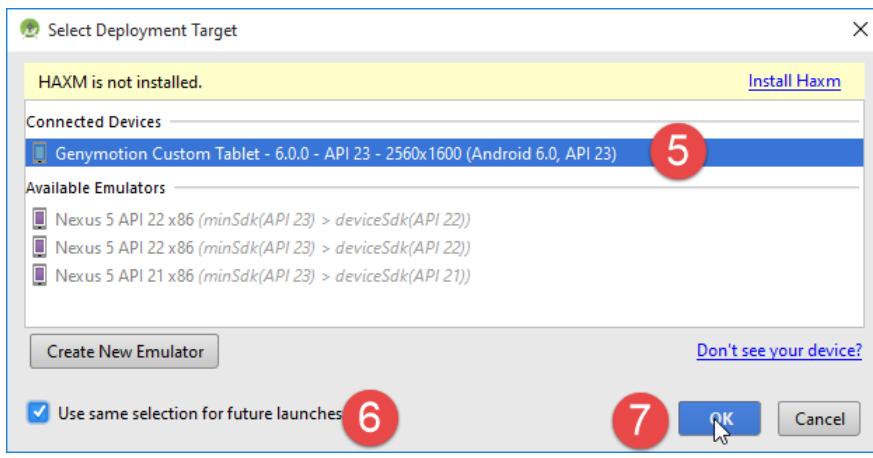
```

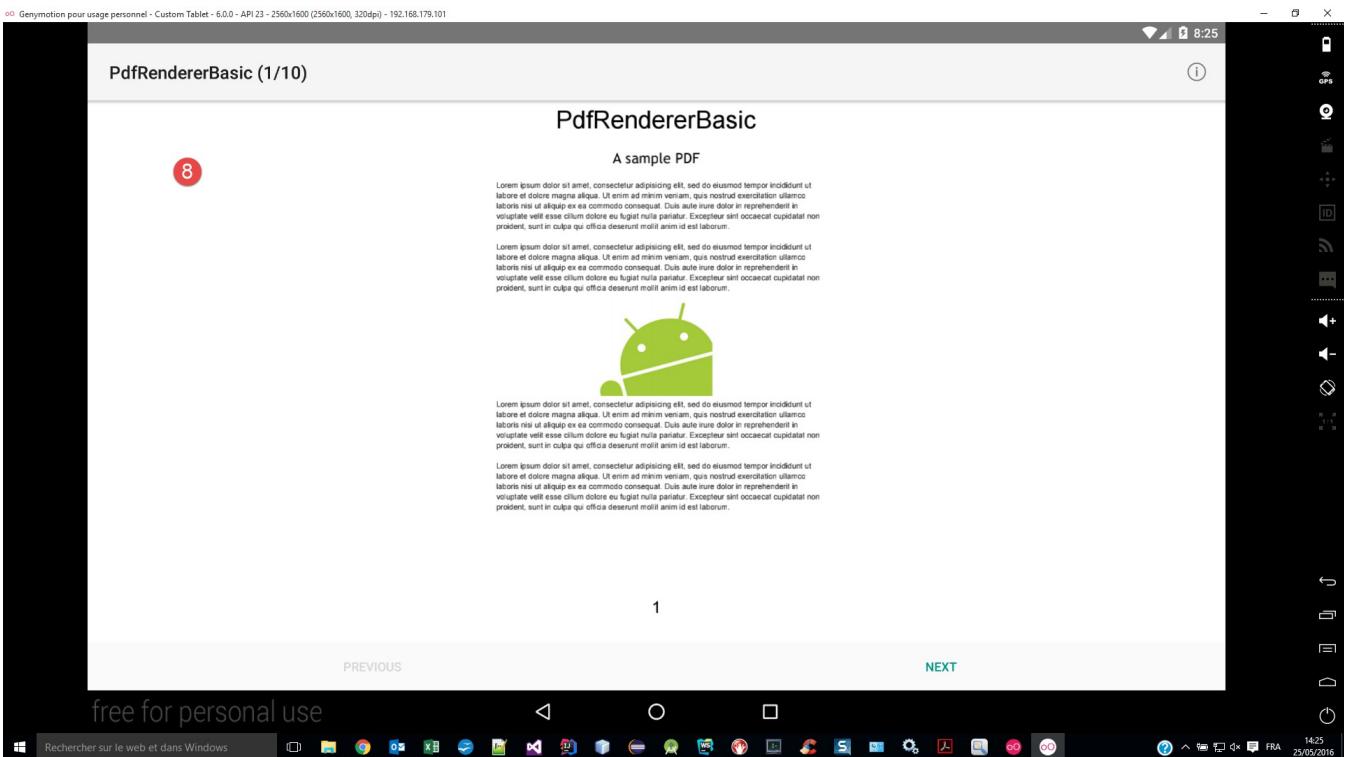
Ligne 6, ci-dessus, il faut remplacer 2.8 par 2.10.

Pour avoir accès au fichier [<projet>/gradle/wrapper/gradle-wrapper.properties], il faut utiliser la perspective projet du projet :

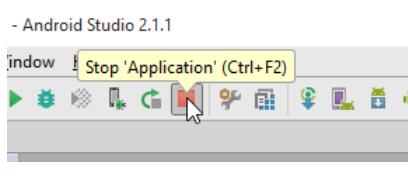


Ceci corrigé, on peut alors compiler l'application [1], lancer l'émulateur Genymotion [2] puis exécuter le projet [3] :

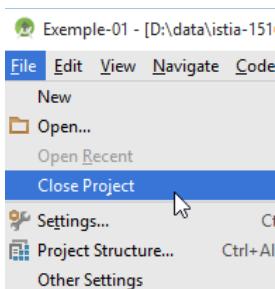




Arrêtons l'application :



On peut maintenant fermer le projet. Nous allons en créer un nouveau.

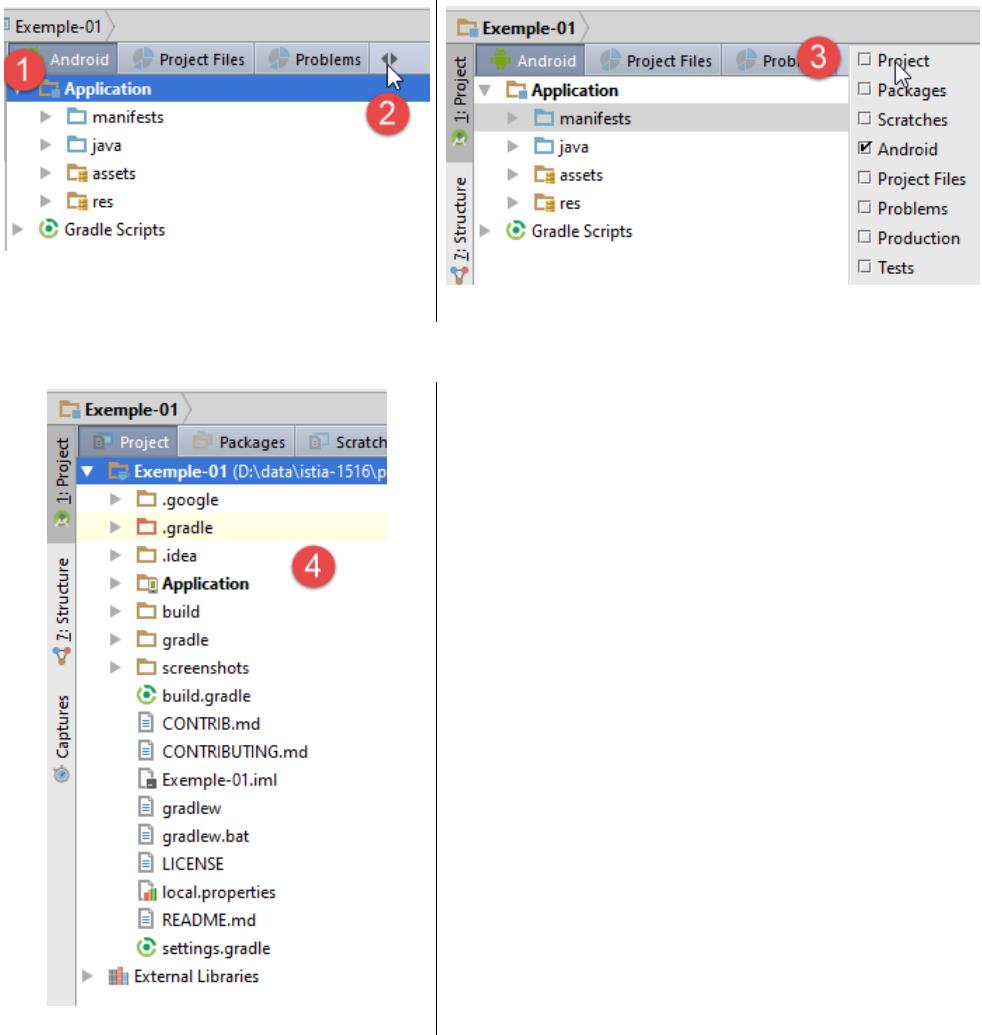


1.2.2 Quelques points sur l'IDE

1.2.2.1 Les perspectives

L'IDE Android Studio (AS) offre différentes perspectives pour travailler avec un projet. Nous en utiliserons principalement deux :

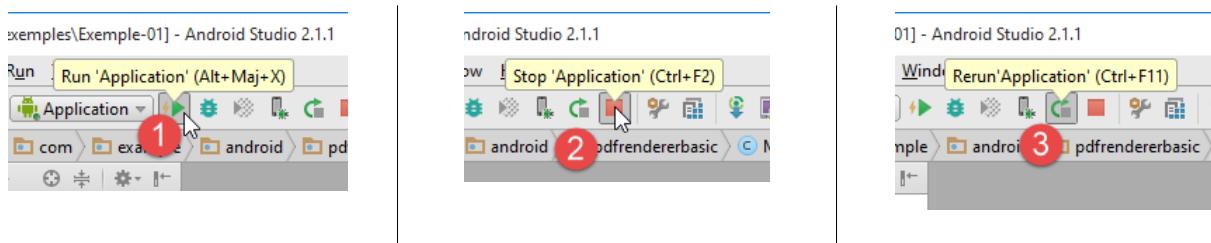
- la perspective [Android] [1] :
- la perspective [Project] [4] ;



La plupart du temps nous travaillerons avec la perspective [Android]. Lorsque nous dupliquerez un projet dans un autre, nous aurons besoin de la perspective [Project].

1.2.2.2 Gestion de l'exécution

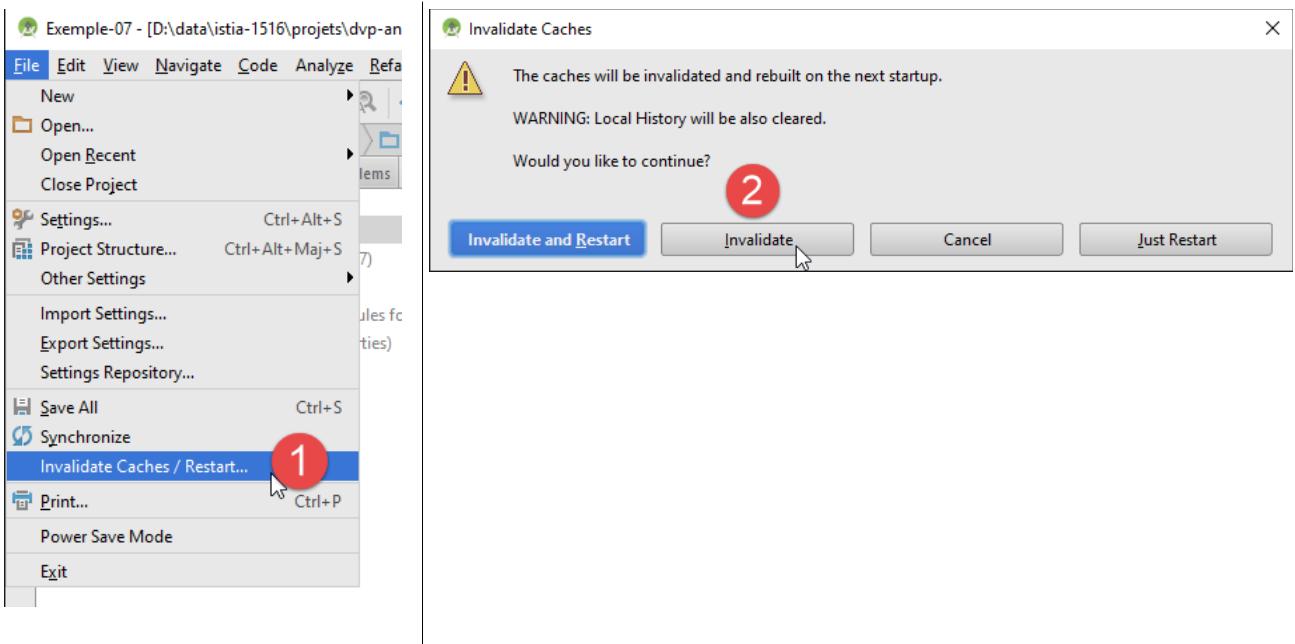
Il y a plusieurs façons d'exécuter / arrêter / réexécuter un projet AS. Il y a tout d'abord les boutons de la barre d'outils :



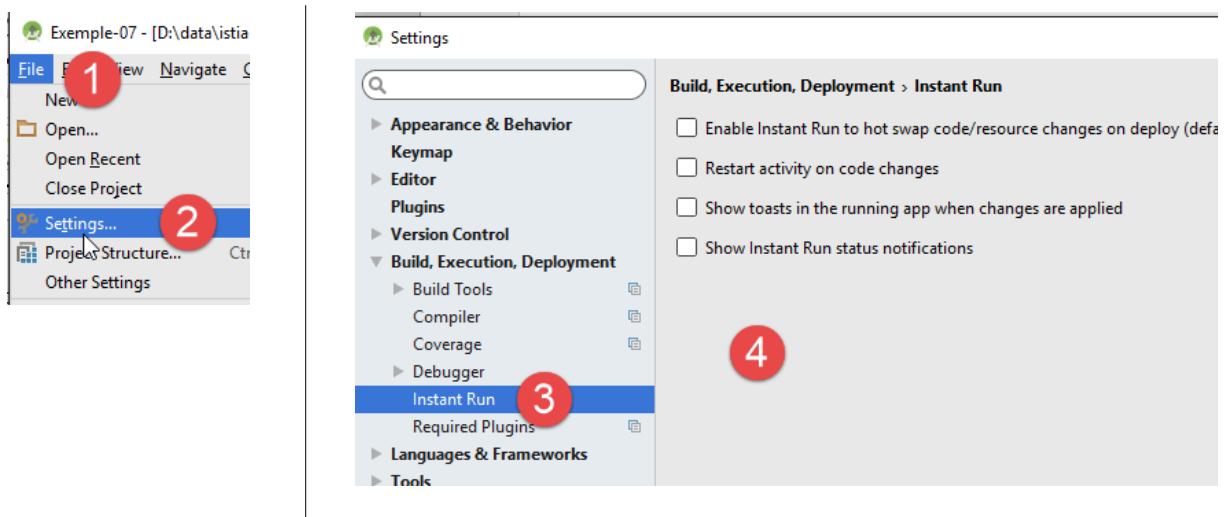
Le bouton [Rerun] [3], arrête l'exécution du projet [2] puis le relance [1].

1.2.2.3 Gestion du cache

Android Studio entretient un cache des projets qu'il gère dans le but de rendre l'IDE la plus réactive possible. Avec la version Android 2.1 (mai 2016), souvent ce cache ne prenait pas en compte les modifications de code que l'on venait de faire. Dans ce cas, il faut invalider ce cache :



Avec Android 2.1 (mai 2016), l'opération précédente devait être faite de nombreuses fois et parfois cela n'était pas suffisant à résoudre l'anomalie détectée. La solution a été d'inhiber la technologie [Instant Run] :

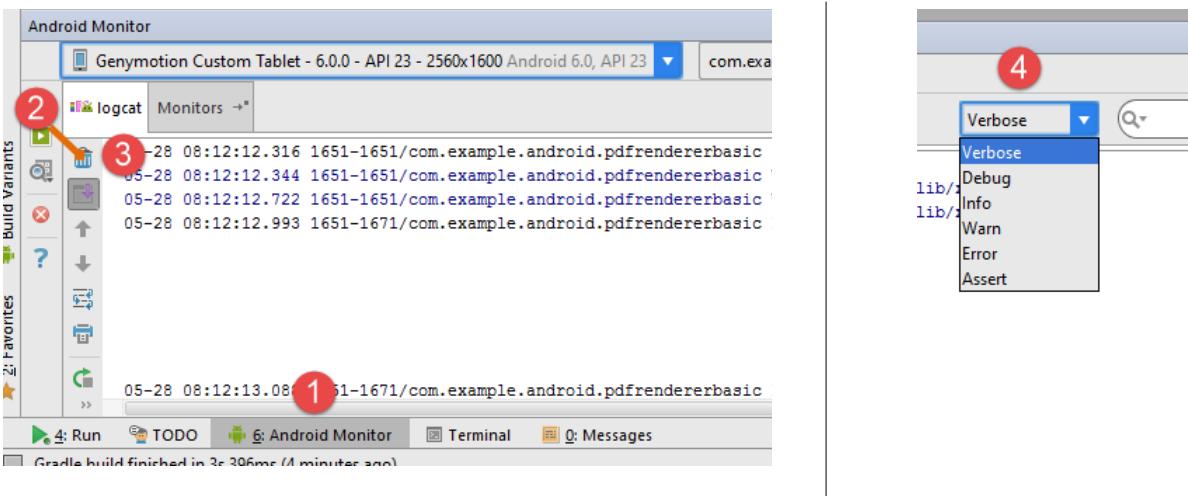


- en [3-4], tout a été désactivé ;

Dans tout ce qui suit, on a travaillé avec cette configuration du cache et on n'a pas rencontré de problèmes.

1.2.2.4 Gestion des logs

Lors de l'exécution d'un projet, des logs s'affichent dans le moniteur Android :

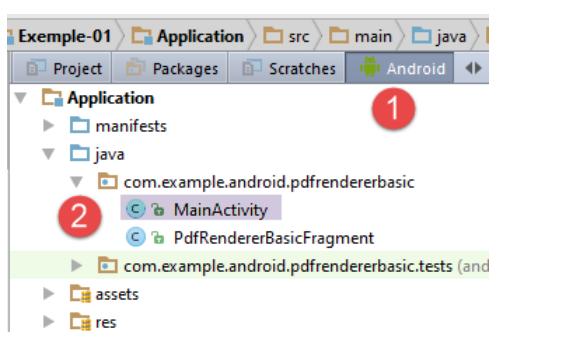


Dans l'onglet [Android Monitor] [1], les logs s'affichent dans l'onglet [logcat] [2]. Le bouton [3] permet d'effacer les logs. Ce bouton est utile lorsqu'on veut voir les logs d'une action particulière :

- on efface les logs ;
- sur le périphérique Android, on fait l'action dont on veut les logs ;
- les logs qui apparaissent alors sont ceux liés à l'action faite ;

Il existe plusieurs niveaux de logs [4]. Par défaut, c'est le mode [Verbose] qui est sélectionné. Il signifie que les logs de tous les niveaux sont affichés. On peut avec [4], sélectionner un niveau particulier.

Les logs sont très utiles pour savoir à quels moments de l'exécution d'un projet certaines méthodes sont affichées. Nous y aurons souvent recours. Prenons le code de la classe [MainActivity] du projet [Exemple-01] :



```

1. package com.example.android.pdfrendererbasic;
2.
3. import android.app.Activity;
4. import android.app.AlertDialog;
5. import android.os.Bundle;
6. import android.view.Menu;
7. import android.view.MenuItem;
8.
9. public class MainActivity extends Activity {
10.
11.     public static final String FRAGMENT_PDF_RENDERRER_BASIC = "pdf_renderer_basic";
12.
13.     @Override
14.     protected void onCreate(Bundle savedInstanceState) {
15.         super.onCreate(savedInstanceState);
16.         setContentView(R.layout.activity_main_real);
17.         if (savedInstanceState == null) {
18.             getFragmentManager().beginTransaction()
19.                 .add(R.id.container, new PdfRendererBasicFragment(),
20.                     FRAGMENT_PDF_RENDERRER_BASIC)
21.                 .commit();
22.         }
23.     }
24. }
```

```

25.     @Override
26.     public boolean onCreateOptionsMenu(Menu menu) {
27.         getMenuInflater().inflate(R.menu.main, menu);
28.         return true;
29.     }
30.
31.     @Override
32.     public boolean onOptionsItemSelected(MenuItem item) {
33.         switch (item.getItemId()) {
34.             case R.id.action_info:
35.                 new AlertDialog.Builder(this)
36.                     .setMessage(R.string.intro_message)
37.                     .setPositiveButton(android.R.string.ok, null)
38.                     .show();
39.             return true;
40.         }
41.         return super.onOptionsItemSelected(item);
42.     }
43. }

```

Ci-dessus, les méthodes [onCreate, ligne 14], [onCreateOptionsMenu, ligne 26] sont des méthodes de la classe parent [Activity] (ligne 9). Elles sont appelées à différents moments du cycle de vie de l'application. Parfois elles sont exécutées plusieurs fois. Même lorsqu'on lit les docs, il est parfois difficile de dire si une telle méthode du cycle de vie va s'exécuter avant ou après une méthode qu'on aurait écrite nous-mêmes. Or cette information est souvent importante à connaître. On peut alors mettre des logs comme ci-dessous :

```

1. public class MainActivity extends Activity {
2.
3.     public static final String FRAGMENT_PDF_RENDERER_BASIC = "pdf_renderer_basic";
4.
5.     @Override
6.     protected void onCreate(Bundle savedInstanceState) {
7.         Log.d("MainActivity", "onCreate");
8.         super.onCreate(savedInstanceState);
9.         ...
10.    }
11.
12.    @Override
13.    public boolean onCreateOptionsMenu(Menu menu) {
14.        Log.d("MainActivity", "onCreateOptionsMenu");
15.        getMenuInflater().inflate(R.menu.main, menu);
16.        ...
17.    }
18.
19.    @Override
20.    public boolean onOptionsItemSelected(MenuItem item) {
21.        Log.d("MainActivity", "onOptionsItemSelected");
22.        switch (item.getItemId()) {
23.            ...
24.        }
25.    }

```

- lignes 7, 14 et 21 on utilise la classe [Log]. Cette classe permet d'écrire des logs sur la console Android [logcat]. Les logs sont classés en divers niveaux (info, warning, debug, verbose, error). [Log.d] affiche des logs de niveau [debug]. Son premier argument est la source du message de log. En effet, diverses sources peuvent émettre des messages sur la console de logs. Afin de pouvoir les différencier, on utilise ce premier argument. Le second argument est le message à écrire sur la console de logs ;

Si nous exécutons de nouveau le projet [Exemple-01], nous obtenons les logs suivants :

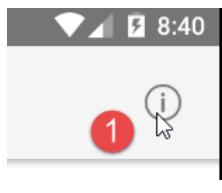
```

1. 05-28 08:37:12.709 23881-23881/com.example.android.pdfrendererbasic D/MainActivity: onCreate
2. 05-28 08:37:12.778 23881-23923/com.example.android.pdfrendererbasic D/OpenGLRender: Use EGL_SWAP_BEHAVIOR_PRESERVED: true
3.
4. [ 05-28 08:37:12.781 23881:23881 D/
5.      HostConnection::get() New Host ...
6. 05-28 08:37:12.967 23881-23881/com.example.android.pdfrendererbasic D/MainActivity: onCreateOptionsMenu

```

On découvre ainsi que la méthode [onCreate] qui crée l'activité Android est exécutée avant la méthode [onCreateOptionsMenu] qui crée le menu de l'application.

Maintenant si on clique sur l'option de menu dans l'émulateur Android [1] :



le log suivant est ajouté dans la console de logs :

```
05-28 08:41:22.881 23881-23881/com.example.android.pdfrendererbasic D>MainActivity: onOptionsItemSelected
```

Dans la suite, nous ajouterons souvent dans le code Android des instructions de logs. La plupart du temps, nous ne les commenterons pas. Elles sont là juste pour inviter le lecteur à regarder la console de logs afin de comprendre progressivement le cycle de vie d'une application Android.

1.2.2.5 Gestion de l'émulateur [Genymotion]

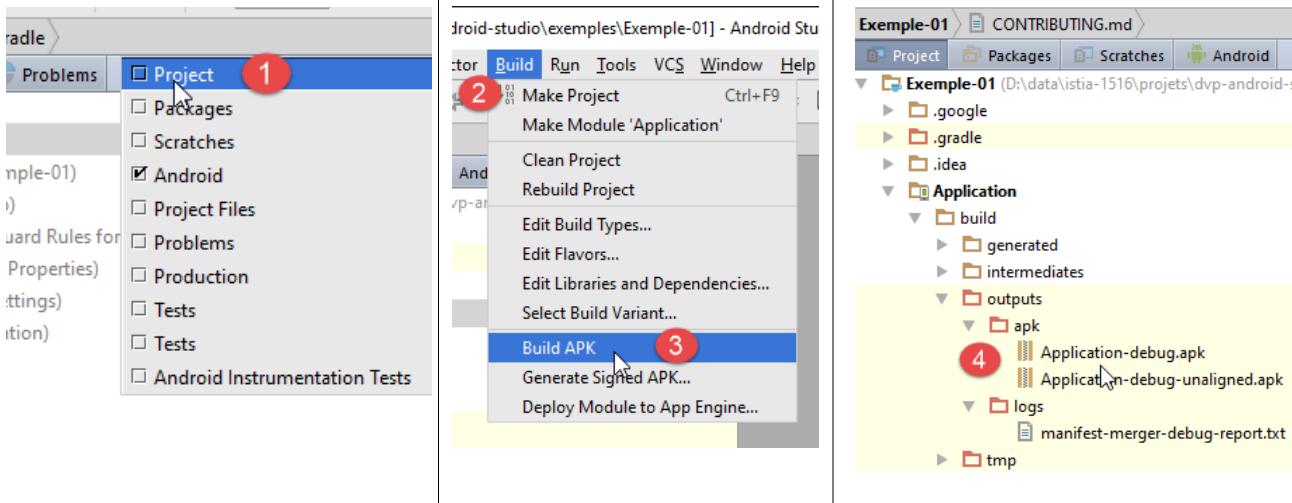
Parfois, l'émulateur Genymotion plante et on ne peut plus le relancer. Cela vient du fait que des processus Virtualbox sont restés vivants dans le gestionnaire des tâches. Ouvrez celui-ci [Ctrl+Alt+Supp] et supprimez toutes les tâches Virtualbox présentes :

Nom	Processeur	Mémoire
Usermode Font Driver Host	0%	0,3
VBoxHeadless.exe	0,1%	51,9 Mo
VBoxHeadless.exe	0%	1,0 Mo
VBoxHeadless.exe	0%	0,8 Mo
VBoxNetDHCP.exe	0%	5,4 Mo
VBoxNetDHCP.exe	0%	1,0 Mo
VBoxNetDHCP.exe	0%	0,8 Mo
VirtualBox Interface	0%	5,3 Mo
Windows Driver Foundation - P...	0%	0,3 Mo

Ceci fait, relancez l'émulateur Genymotion à partir d'Android Studio.

1.2.2.6 Gestion du binaire APK créé

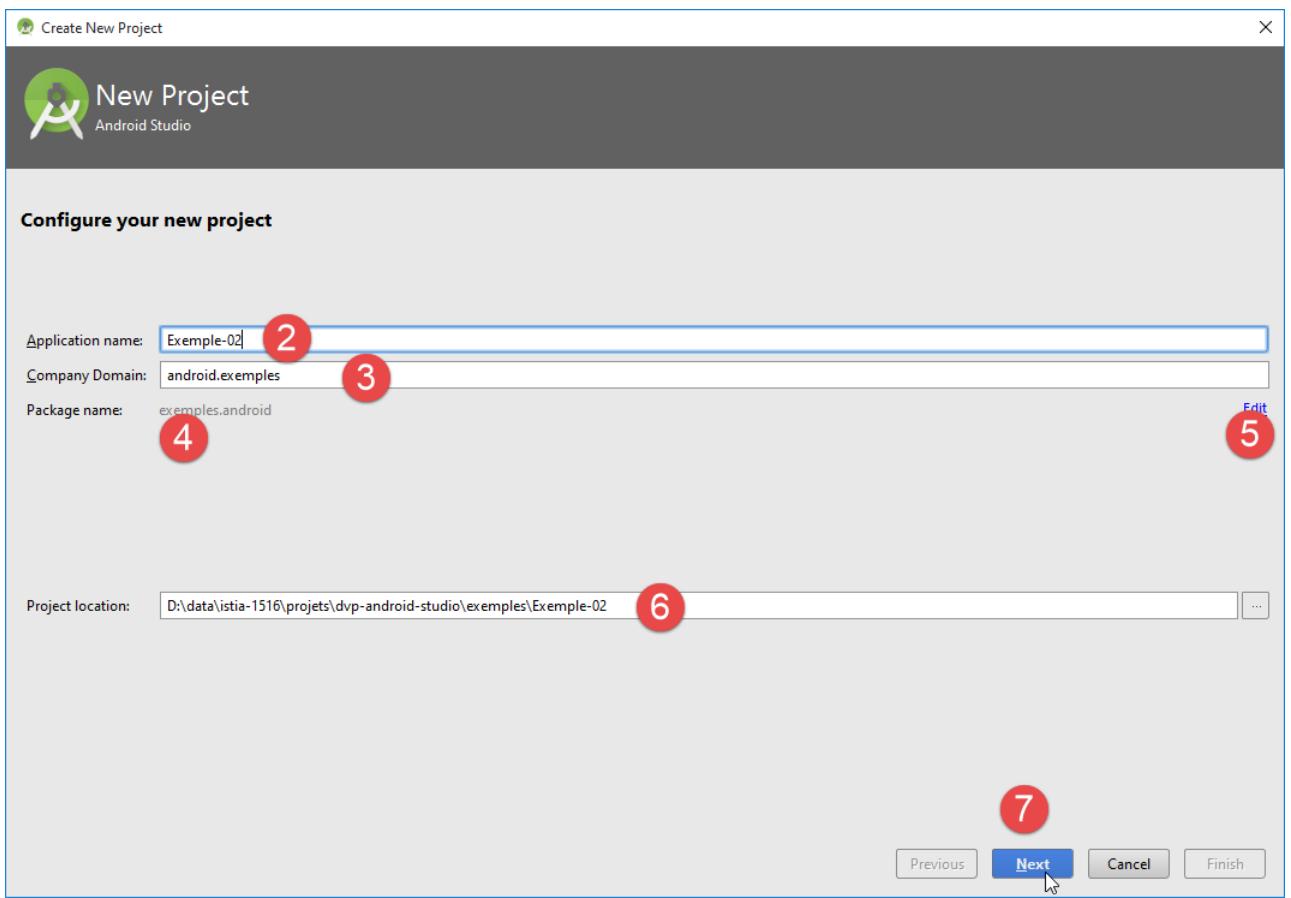
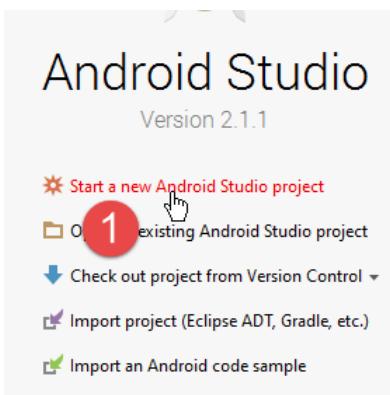
La compilation du projet produit un binaire de suffixe .apk :

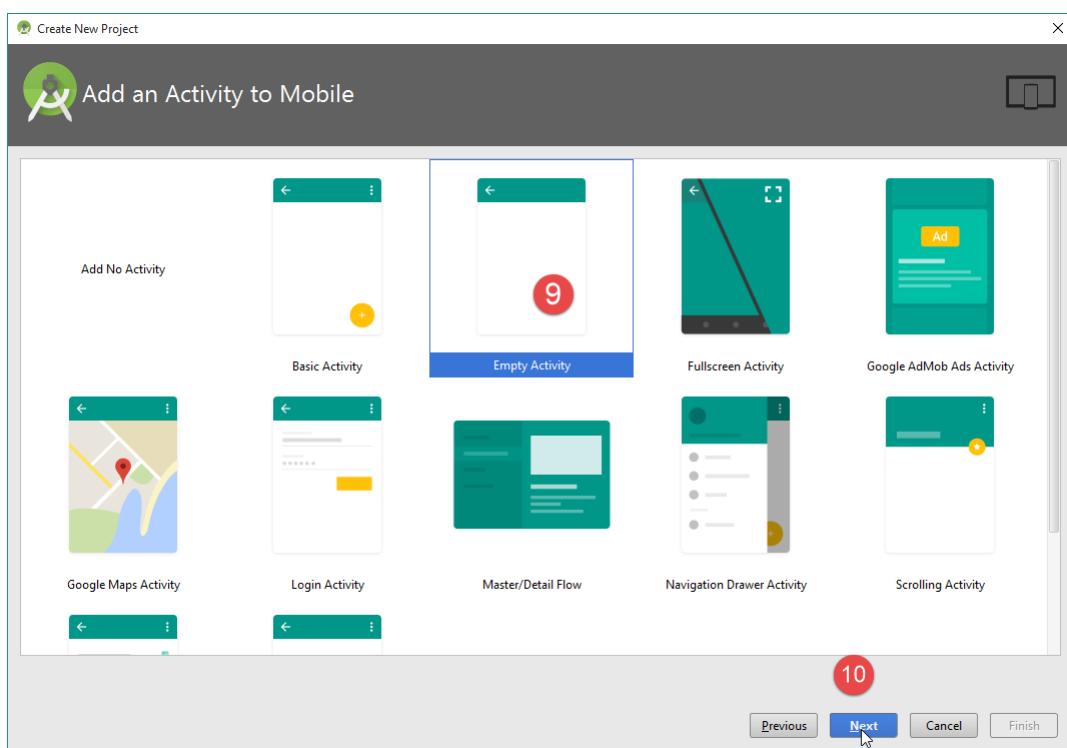
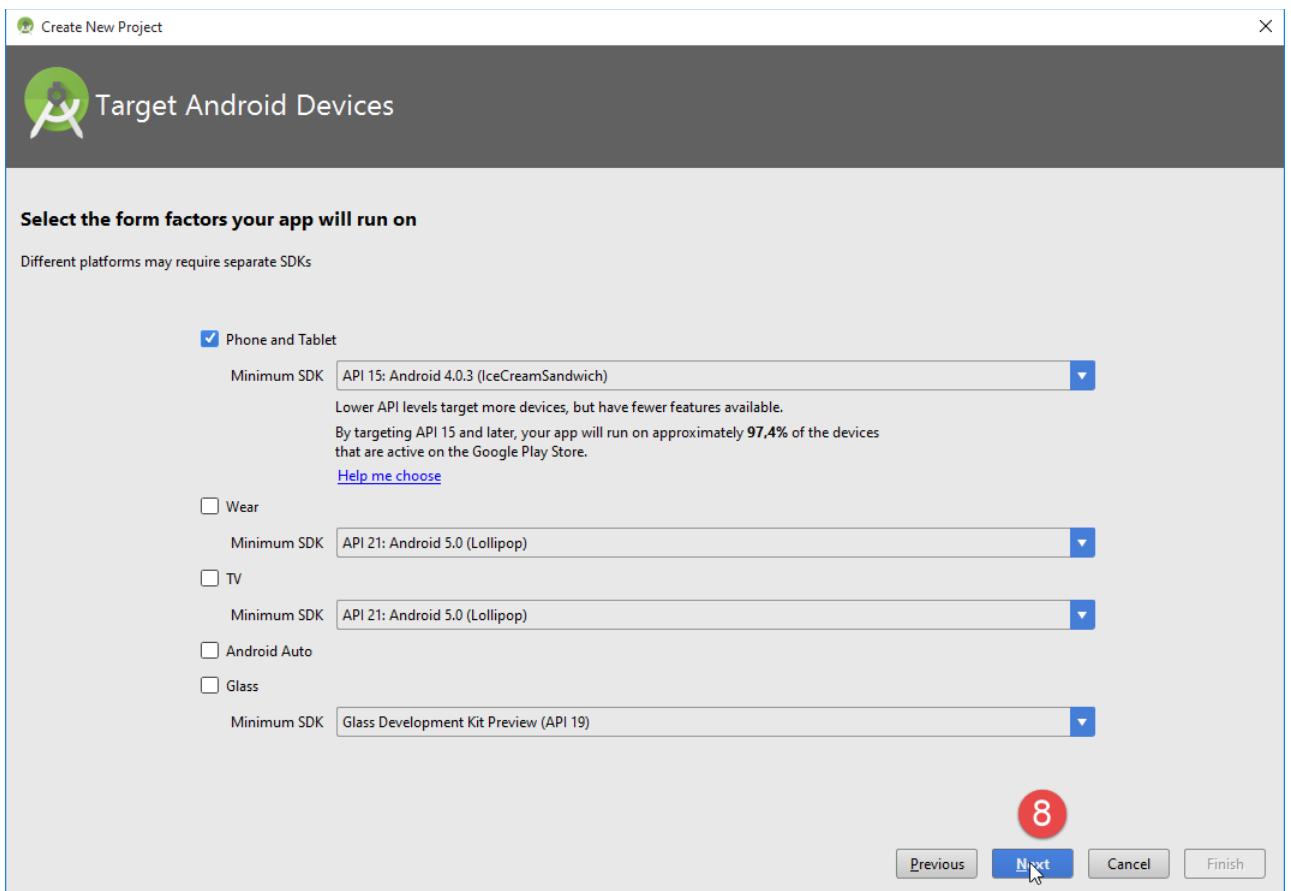


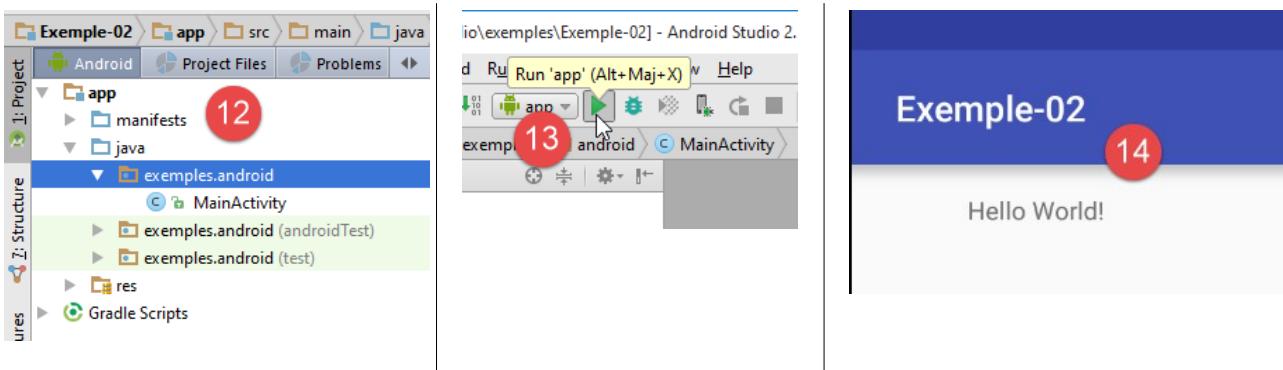
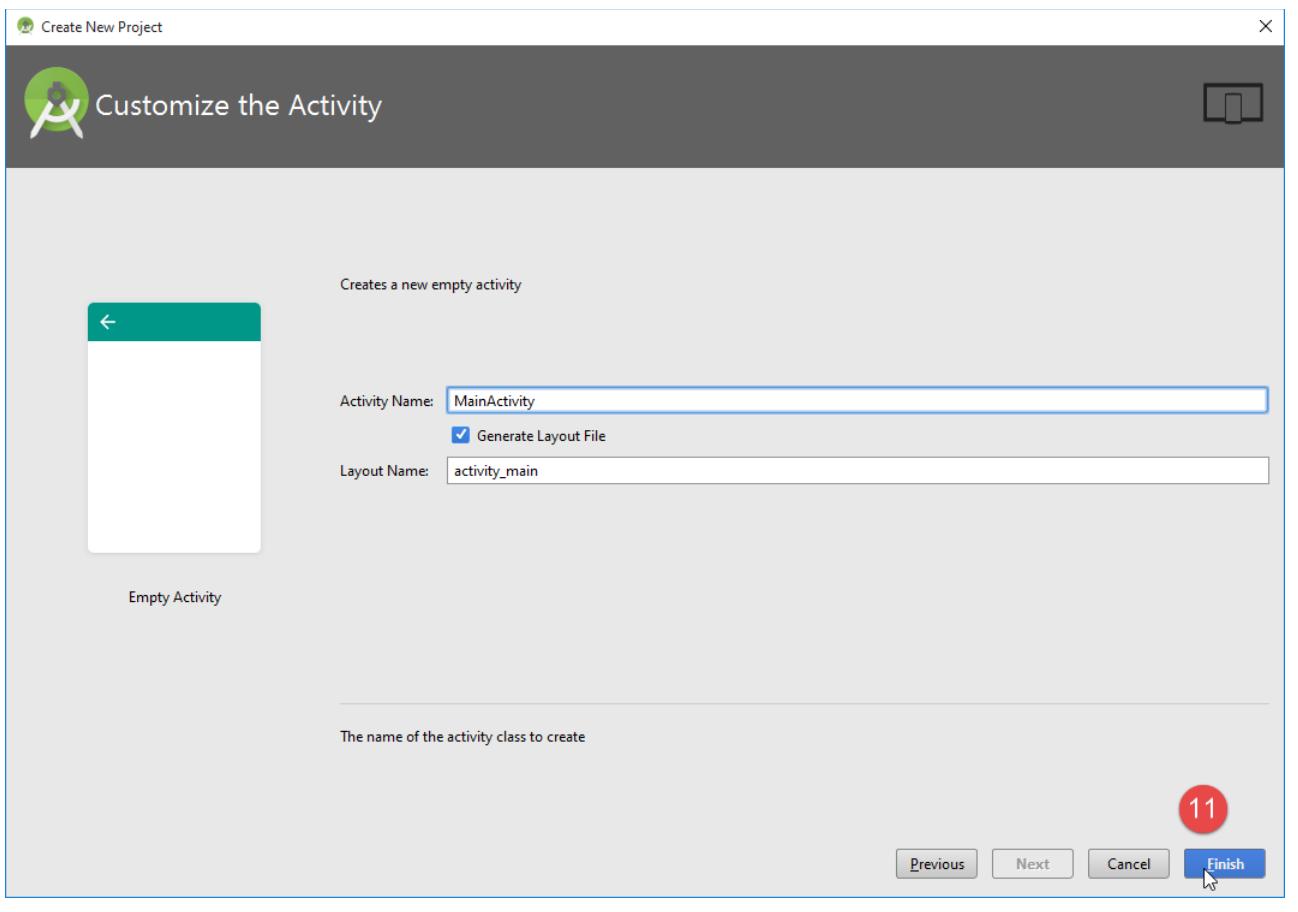
Il y a deux versions : celle dite [debug] et l'autre dite [debug-unaligned]. C'est la première qu'il faut utiliser, l'autre étant une version intermédiaire. Le binaire .apk produit en [4] peut être transféré directement sur un émulateur ou un périphérique Android. Pour le transférer sur un émulateur, il suffit de le tirer / déposer sur l'émulateur avec la souris.

1.3 Exemple-02 : un projet Android basique

Créons avec Android Studio un nouveau projet Android [1-12] :



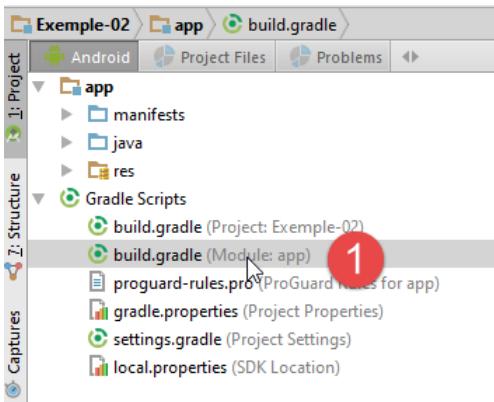




En [13], on exécute l'application. On obtient alors l'affichage [14] sur l'émulateur Genymotion.

1.3.1 Configuration Gradle

Le projet créé est configuré par le fichier [build.gradle] suivant :



```

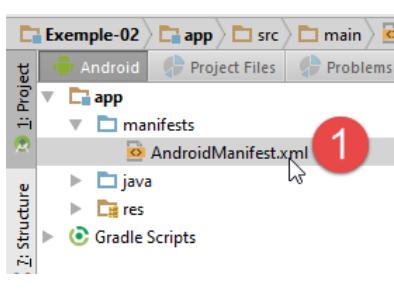
1. apply plugin: 'com.android.application'
2.
3. android {
4.     compileSdkVersion 23
5.     buildToolsVersion "23.0.3"
6.     defaultConfig {
7.         applicationId "exemples.android"
8.         minSdkVersion 15
9.         targetSdkVersion 23
10.        versionCode 1
11.        versionName "1.0"
12.    }
13.
14.    buildTypes {
15.        release {
16.            minifyEnabled false
17.            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18.        }
19.    }
20. }
21.
22. dependencies {
23.     compile fileTree(dir: 'libs', include: ['*.jar'])
24.     testCompile 'junit:junit:4.12'
25.     compile 'com.android.support:appcompat-v7:23.4.0'
26. }

```

Ce fichier a été généré par l'IDE avec les éléments de sa configuration. C'est un fichier minimal que nous allons progressivement enrichir.

- lignes 3-12 : les caractéristiques de l'application Android ;
- lignes 22-25 : ses dépendances. C'est surtout là que nous amènerons des modifications selon les exemples étudiés ;

1.3.2 Le manifeste de l'application



Le fichier [AndroidManifest.xml] [1] fixe les caractéristiques du binaire de l'application Android. Son contenu est ici le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.             package="exemples.android">
4.     <application
5.         android:allowBackup="true"
6.         android:icon="@mipmap/ic_launcher"

```

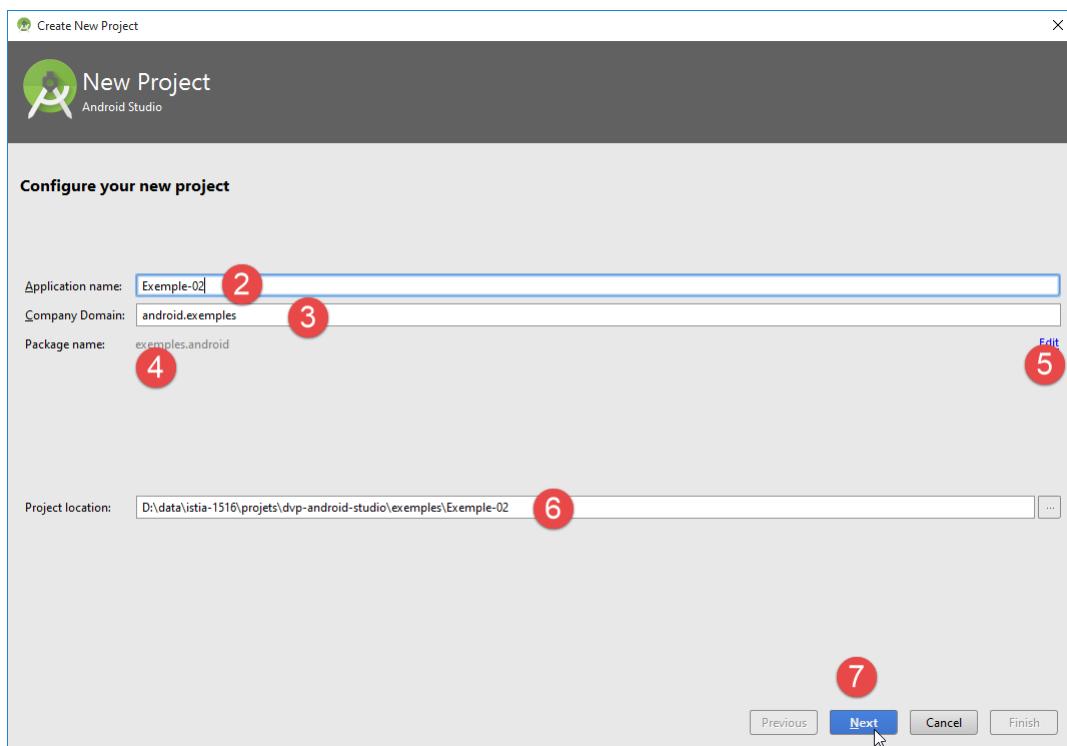
```

7.     android:label="@string/app_name"
8.     android:supportsRtl="true"
9.     android:theme="@style/AppTheme">
10.    <activity android:name=".MainActivity">
11.        <intent-filter>
12.            <action android:name="android.intent.action.MAIN"/>
13.            <category android:name="android.intent.category.LAUNCHER"/>
14.        </intent-filter>
15.    </activity>
16. </application>
17. </manifest>

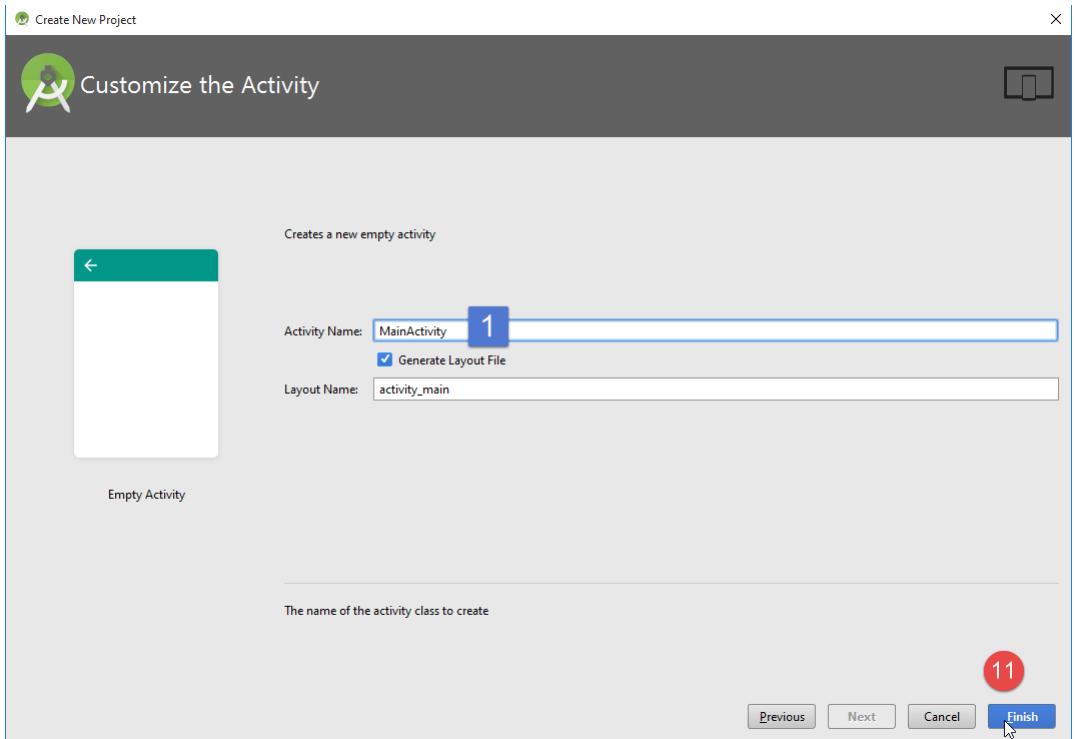
```

- ligne 3 : le paquetage du projet Android ;
- ligne 10 : le nom de l'activité ;

Ces deux renseignements viennent des saisies faites lors de la création du projet :



- la ligne 3 du manifeste (package) vient de la saisie [4] ci-dessus. Un certain nombre de classes sont automatiquement générées dans ce package ;

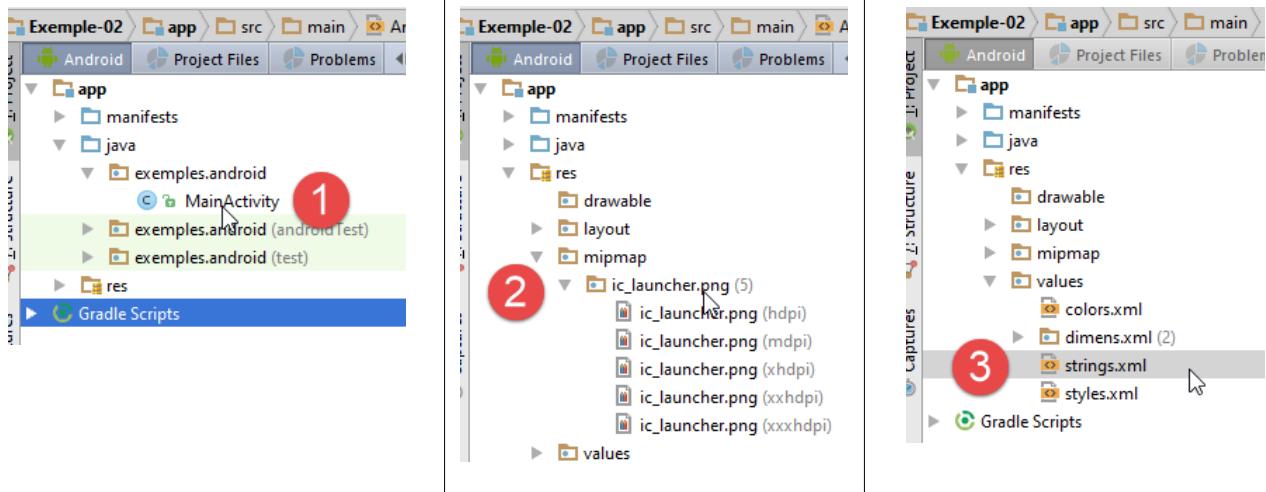


- la ligne 10 du manifeste (nom de l'activité) vient de la saisie [1] ci-dessus ;

Revenons au manifeste :

```

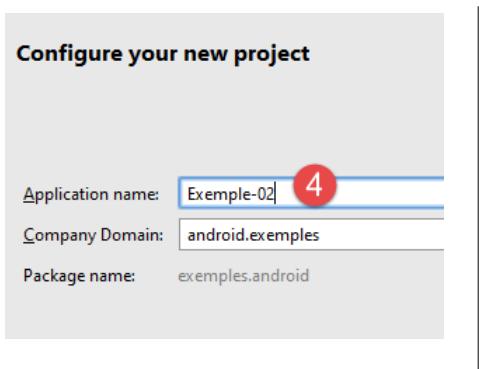
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.           package="exemples.android">
4.   <application
5.     android:allowBackup="true"
6.     android:icon="@mipmap/ic_launcher"
7.     android:label="@string/app_name"
8.     android:supportsRtl="true"
9.     android:theme="@style/AppTheme">
10.    <activity android:name=".MainActivity">
11.      <intent-filter>
12.        <action android:name="android.intent.action.MAIN"/>
13.        <category android:name="android.intent.category.LAUNCHER"/>
14.      </intent-filter>
15.    </activity>
16.  </application>
17. </manifest>
```



- ligne 10 : l'activité principale de l'application. Elle référence la classe [1] ci-dessus ;
- ligne 6 : l'icône [2] de l'application. Elle peut être changée ;
- ligne 7 : le libellé de l'application. Il se trouve dans le fichier [strings.xml] [3] :

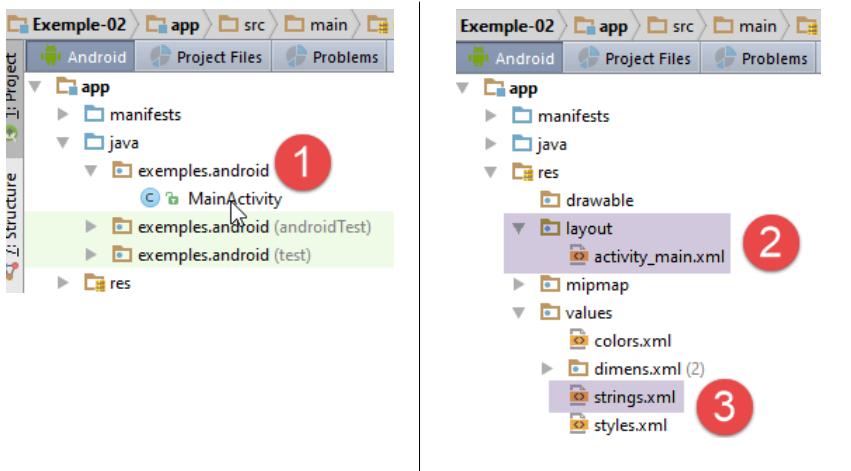
```
1. <resources>
2.   <string name="app_name">Exemple-02</string>
3. </resources>
```

Le fichier [strings.xml] contient les chaînes de caractères utilisées par l'application. Ligne 2, le nom de l'application provient de la saisie faite lors de la construction du projet [4] :



- ligne 10 : une balise d'activité. Une application Android peut avoir plusieurs activités ;
- ligne 12 : l'activité est désignée comme étant l'activité principale ;
- ligne 13 : et elle doit apparaître dans la liste des applications qu'il est possible de lancer sur l'appareil Android.

1.3.3 L'activité principale



Une application Android repose sur une ou plusieurs activités. Ici une activité [1] a été générée : [MainActivity]. Une activité peut afficher une ou plusieurs vues selon son type. La classe [MainActivity] générée est la suivante :

```

1. package exemples.android;
2.
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5.
6. public class MainActivity extends AppCompatActivity {
7.
8.     @Override
9.     protected void onCreate(Bundle savedInstanceState) {
10.         super.onCreate(savedInstanceState);
11.         setContentView(R.layout.activity_main);
12.     }
13. }
```

- ligne 6 : la classe [MainActivity] étend la classe Android [AppCompatActivity]. Ce sera le cas de toutes les activités futures ;
- ligne 9 : la méthode [onCreate] est exécutée lorsque l'activité est créée. C'est avant l'affichage de la vue associée à l'activité ;
- ligne 10 : la méthode [onCreate] de la classe parente est appelée. Il faut toujours le faire ;
- ligne 11 : le fichier [activity_main.xml] [2] est la vue associée à l'activité. La définition XML de cette vue est la suivante :

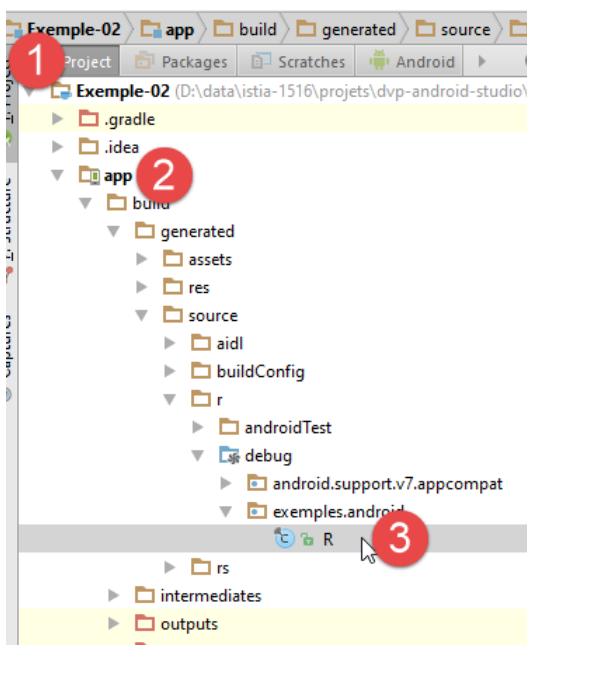
```

a) <?xml version="1.0" encoding="utf-8"?>
b) <RelativeLayout
c)     xmlns:android="http://schemas.android.com/apk/res/android"
d)     xmlns:tools="http://schemas.android.com/tools"
e)     android:layout_width="match_parent"
f)     android:layout_height="match_parent"
g)     android:paddingLeft="@dimen/activity_horizontal_margin"
h)     android:paddingRight="@dimen/activity_horizontal_margin"
i)     android:paddingTop="@dimen/activity_vertical_margin"
j)     android:paddingBottom="@dimen/activity_vertical_margin"
k)     tools:context="exemples.android.MainActivity">
l)
m)     <TextView
n)         android:text="Hello World!"
o)         android:layout_width="wrap_content"
p)         android:layout_height="wrap_content"/>
q) </RelativeLayout>
```

- lignes b-k : le gestionnaire de mise en forme. Celui qui a été choisi par défaut est le type [RelativeLayout]. Dans ce type de conteneur, les composants sont placés les uns par rapport aux autres (à droite de, à gauche de, dessous, au-dessus) ;
- lignes m-p : un composant de type [TextView] qui sert à afficher du texte ;
- ligne n : le texte affiché. Il est déconseillé de mettre du texte en dur dans les vues. Il est préférable de déplacer ces textes dans le fichier [res/values/strings.xml] [3] :

Le texte affiché sera donc [*Hello World!*]. Où sera-t-il affiché ? Le conteneur [RelativeLayout] va remplir l'écran. Le [TextView] qui est son seul et unique élément sera affiché en haut et à gauche de ce conteneur, donc en haut et à gauche de l'écran ;

Que signifie [R.layout.activity_main] ligne 11 ? Chaque ressource Android (vues, fragments, composants, ...) se voit attribuer un identifiant. Ainsi une vue [V.xml] se trouvant dans le dossier [res / layout] sera identifiée par [R.layout.V]. R est une classe générée dans le dossier [gen] :



La classe [R] est la suivante :

```

1. .....
2.     public static final class string {
3.         public static final int abc_action_bar_home_description=0x7f060000;
4.         public static final int abc_action_bar_home_description_format=0x7f060001;
5.         public static final int abc_action_bar_home_subtitle_description_format=0x7f060002;
6.         ...
7.         public static final int app_name=0x7f060014;
8.     }
9.
10.    public static final class layout {
11.        public static final int abc_action_bar_title_item=0x7f040000;
12.        public static final int abc_action_bar_up_container=0x7f040001;
13.        ...
14.        public static final int activity_main=0x7f040019;
15.        ...
16.    }
17.
18.    public static final class mipmap {
19.        public static final int ic_launcher=0x7f030000;
20.    }

```

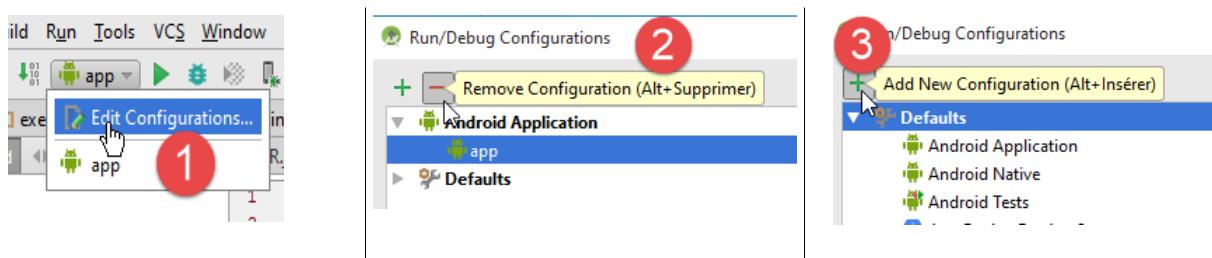
- ligne 14 : l'attribut [R.layout.activity_main] est l'identifiant de la vue [res / layout / activity_main.xml] ;
- ligne 7 : l'attribut [R.string.app_name] est l'identifiant de la chaîne [app_name] dans le fichier [res / values / string.xml] ;
- ligne 19 : l'attribut [R.mipmap.ic_launcher] est l'identifiant de l'image [res / mipmap / ic_launcher] ;

On se souviendra donc que lorsqu'on référence [R.layout.activity_main] dans le code, on référence un attribut de la classe [R]. L'IDE nous aide à connaître les différents éléments de cette classe :

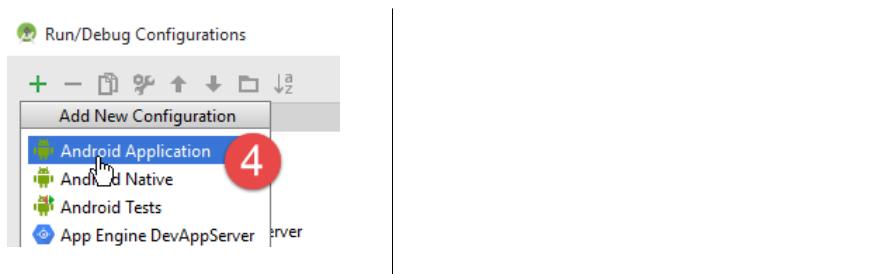


1.3.4 Exécution de l'application

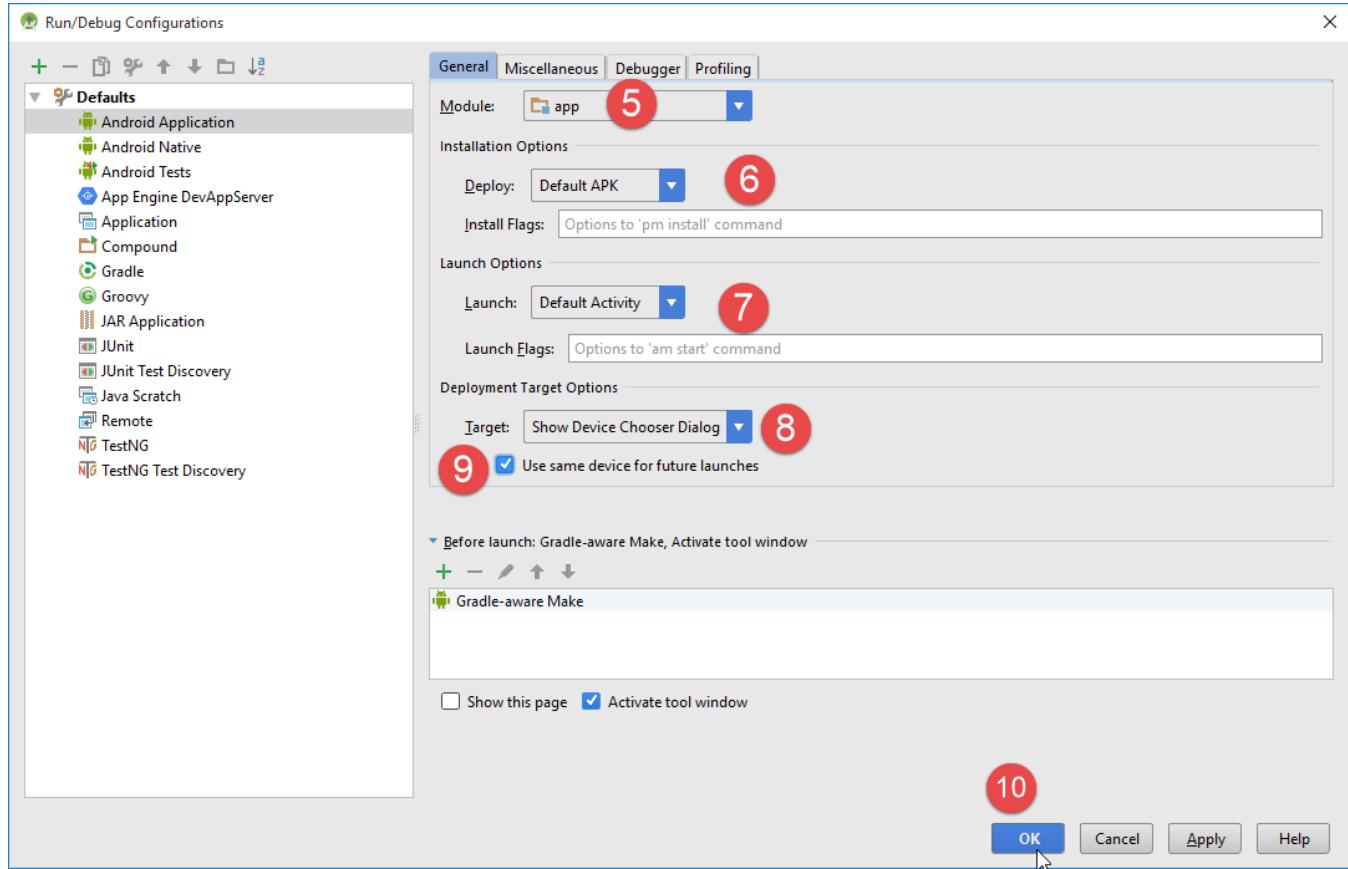
Pour exécuter une application Android, il nous faut créer une configuration d'exécution :



- en [1], choisir [Edit Configurations] ;
- le projet a été créé avec une configuration [app] que nous allons supprimer [2] pour la recréer ;
- en [3], créer une nouvelle configuration d'exécution ;



- en [4], choisir [Android Application] ;



- en [5], dans la liste déroulante choisir le module [app] ;
- en [6-8], garder les valeurs proposées par défaut ;
- en [7], l'activité par défaut est celle définie dans le fichier [AndroidManifest.xml] (ligne 1 ci-dessous) :

```

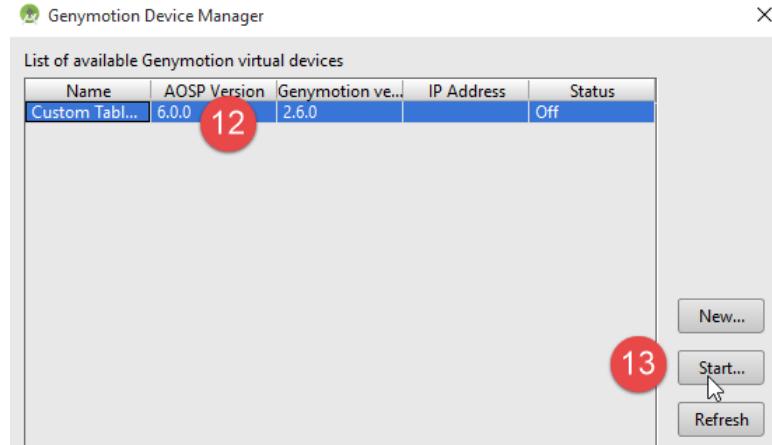
1. <activity android:name=".MainActivity">
2.   <intent-filter>
3.     <action android:name="android.intent.action.MAIN"/>
4.
5.     <category android:name="android.intent.category.LAUNCHER"/>
6.   </intent-filter>
7. </activity>

```

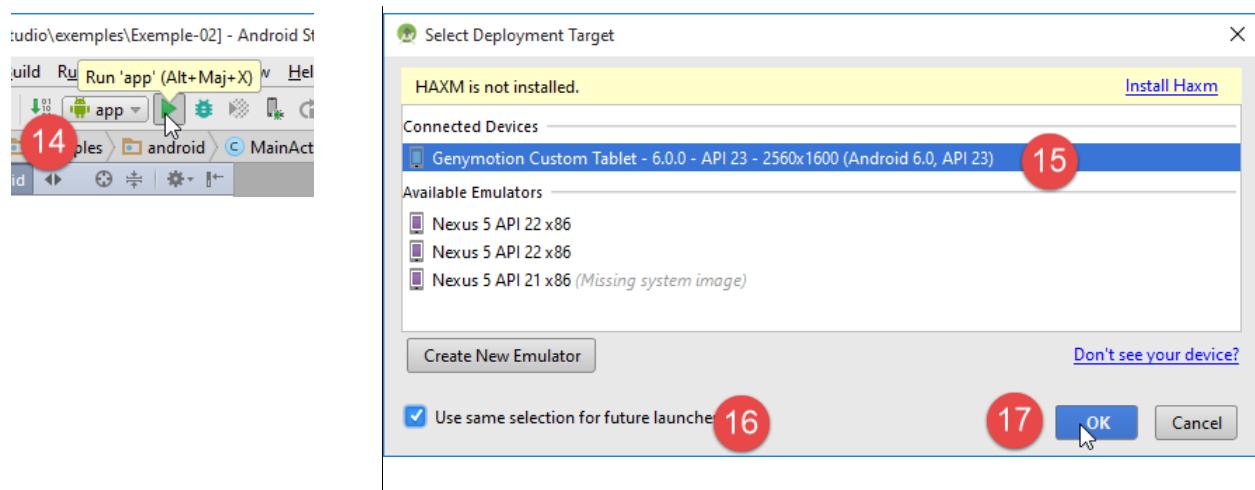
- en [8], sélectionner [Show Chooser Dialog] qui permet de choisir le périphérique d'exécution de l'application (émulateur, tablette) ;
- en [9], on indique que ce choix doit être mémorisé ;
- validez la configuration ;



- en [11], lancer le gestionnaire des émulateurs [Genymotion] (cf chapitre 6.9, page 480) ;

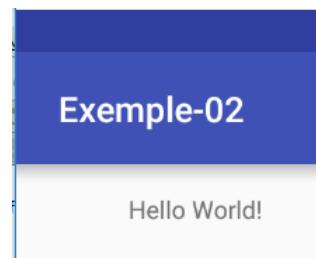


- en [12], sélectionner un émulateur de tablette et lancez le [13] ;



- en [14], exécutez la configuration d'exécution [app] ;
- en [15] est présenté le formulaire de choix du périphérique d'exécution. Un seul est ici disponible : l'émulatuer [Genymotion] lancé précédemment ;

L'émulateur logiciel affiche au bout d'un moment la vue suivante :



1.3.5 Le cycle de vie d'une activité

Rvenons sur le code de l'activité [MainActivity] :

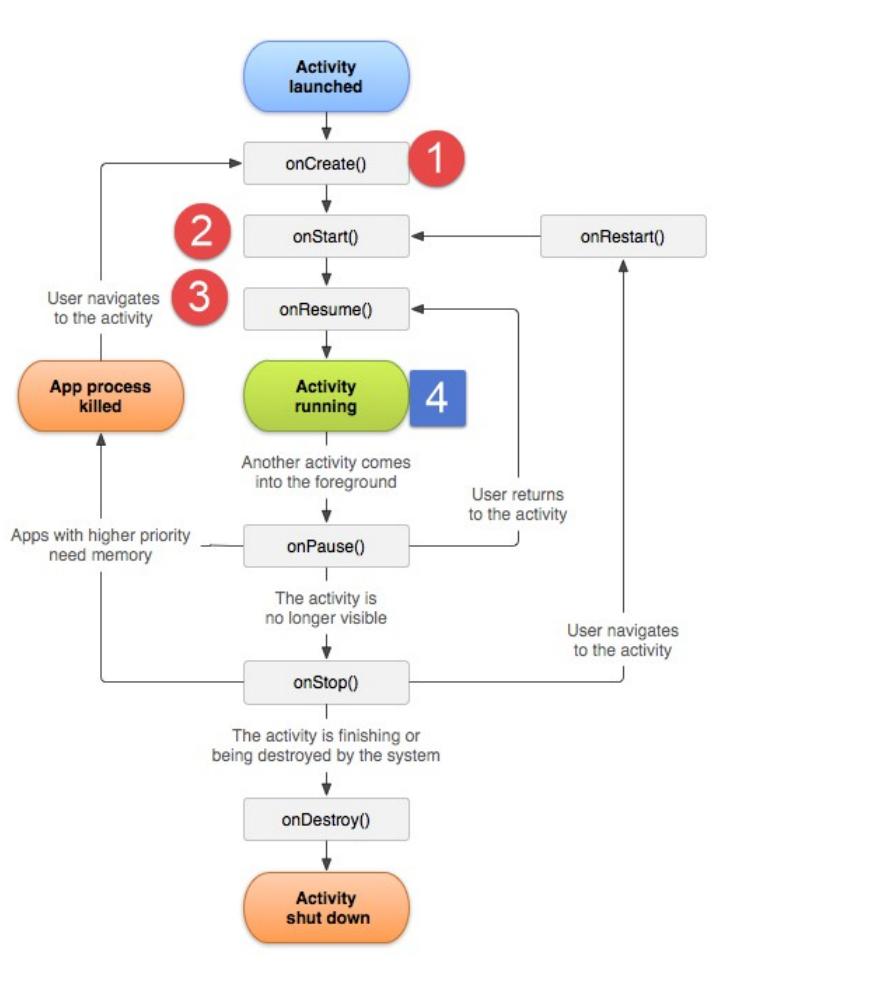
```

1. package exemples.android;
2.
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
```

```

5.
6. public class MainActivity extends AppCompatActivity {
7.
8.     @Override
9.     protected void onCreate(Bundle savedInstanceState) {
10.         super.onCreate(savedInstanceState);
11.         setContentView(R.layout.activity_main);
12.     }
13. }
```

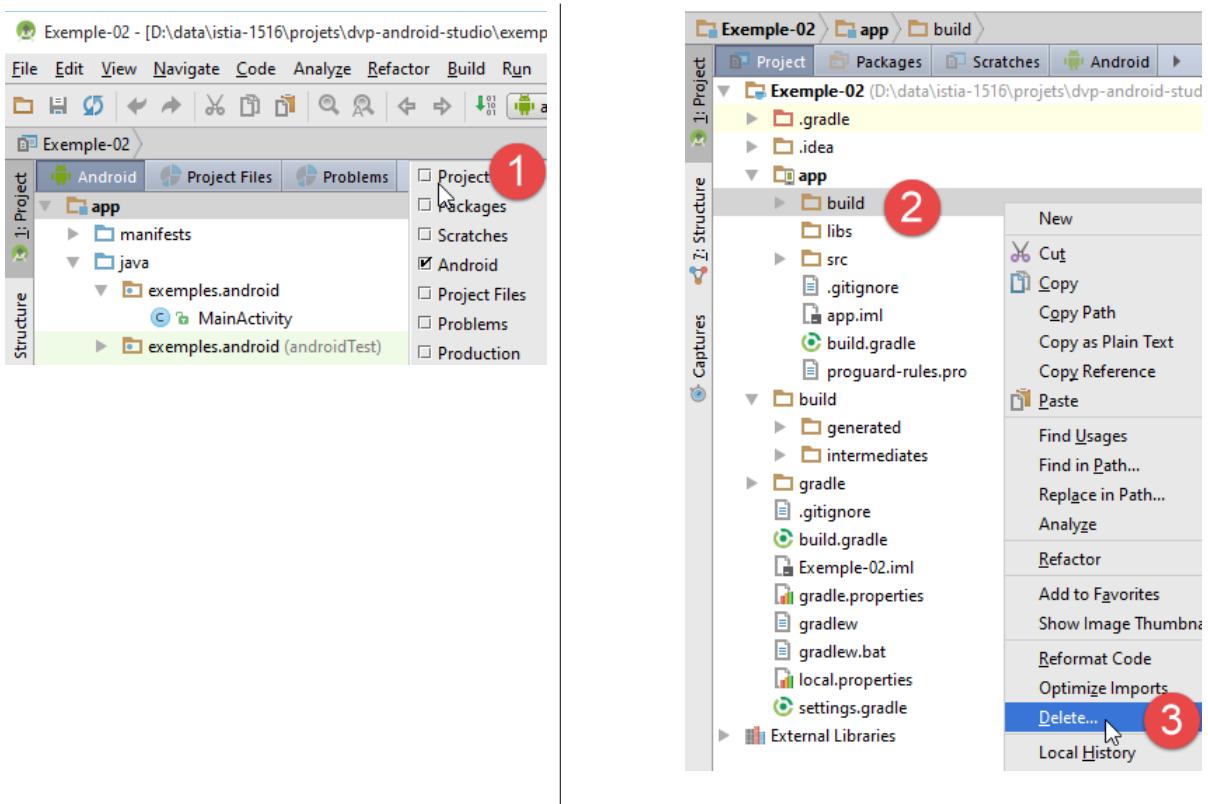
La méthode [onCreate] des lignes 8-12 fait partie des méthodes qui peuvent être appelées au cours du cycle de vie d'une activité. La documentation Android donne la liste de celles-ci :



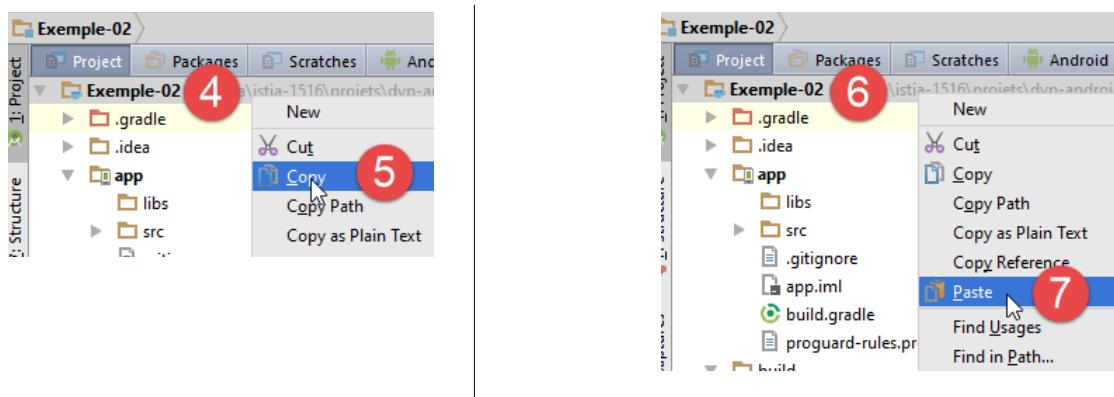
- [1] : la méthode [onCreate] est appelée au démarrage de l'activité. C'est dans cette méthode qu'on associe l'activité à une vue et qu'on récupère les références des composants de celle-ci ;
- [2-3] : les méthodes [onStart, onResume] sont ensuite appelées. On voit que la méthode [onResume] est la dernière méthode à être exécutée avant d'arriver à l'état [4] de l'activité en cours d'exécution ;

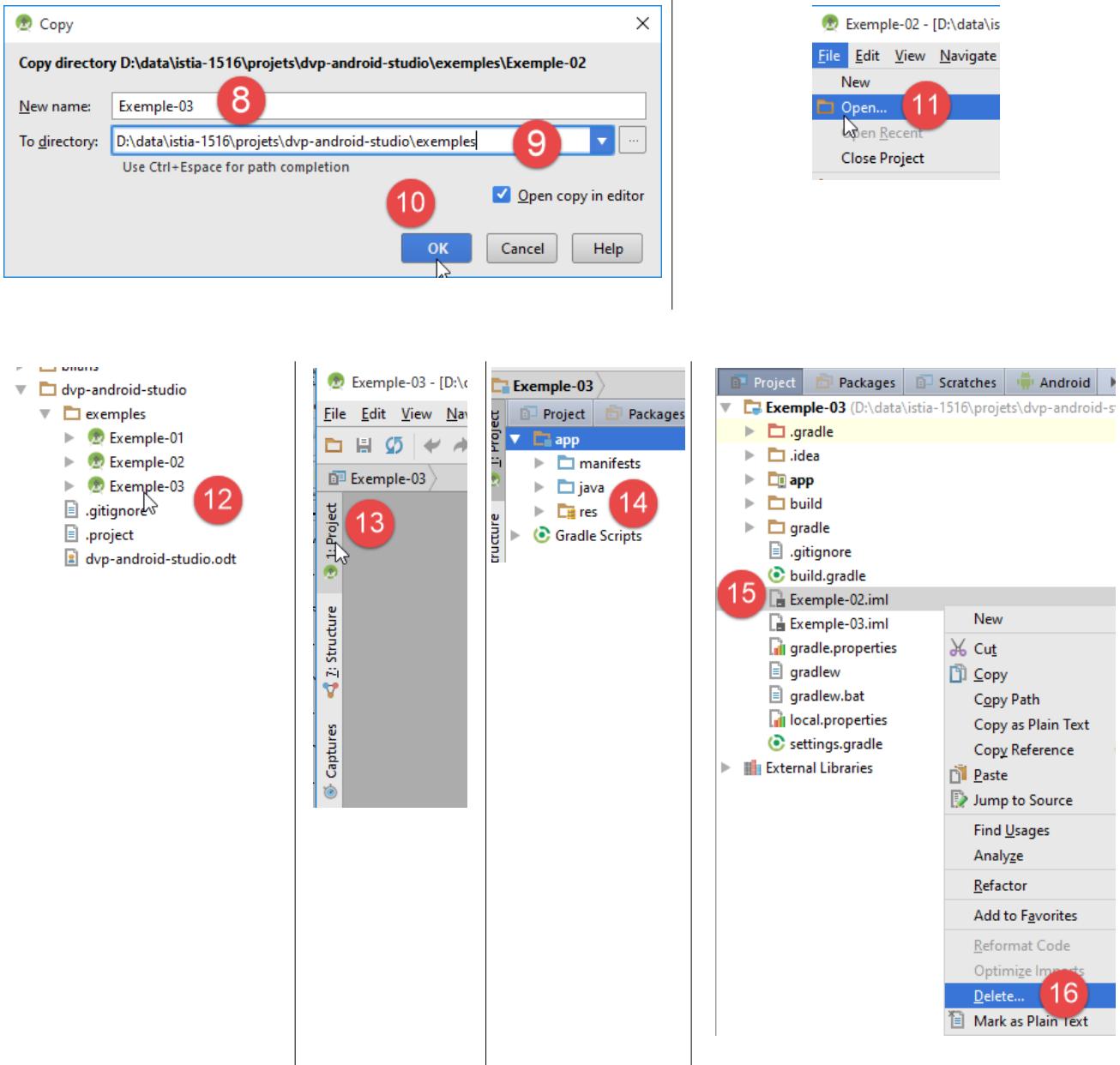
1.4 Exemple-03 : réécriture du projet [Exemple-02] avec la bibliothèque [Android Annotations]

Nous allons maintenant introduire la bibliothèque [[Android Annotations](#)] qui facilite l'écriture des applications Android. Pour cela on duplique l'exemple [Exemple-02] dans [Exemple-03] en suivant la procédure [1-16].



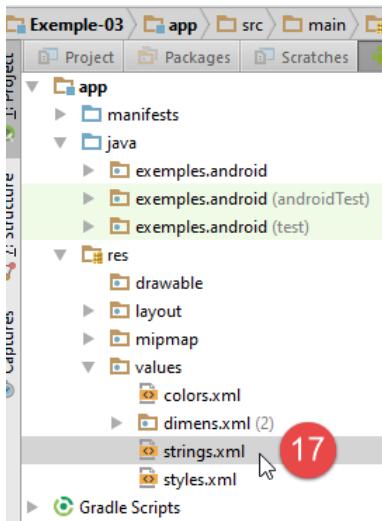
- en [1], prenez la perspective [Project] pour voir la totalité du projet Android ;





Note : entre [14] et [15], on est passé d'une perspective [Android] à une perspective [Project] (cf paragraphe 1.2.2.1, page 16).

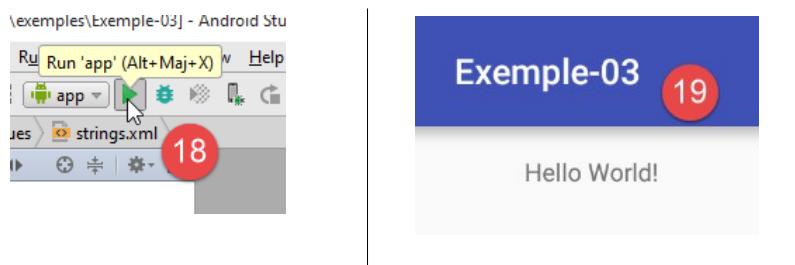
Nous modifions ensuite le fichier [res / values / strings.xml] [17] :



Le fichier [strings.xml] et modifié de la façon suivante :

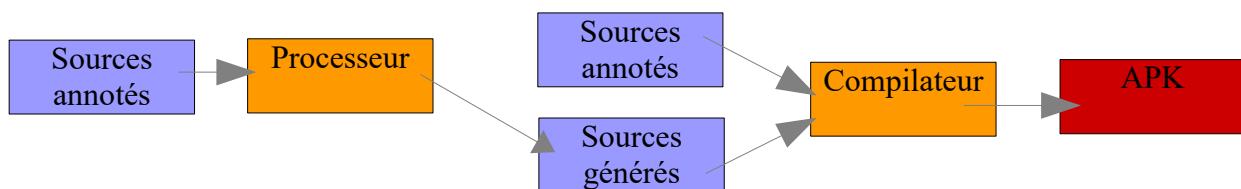
```
1. <resources>
2.   <string name="app_name">Exemple-03</string>
3. </resources>
```

Maintenant, nous exécutons la nouvelle application qui a repris toute la configuration de [Exemple-02] :



En [19], nous obtenons le même résultat qu'avec [Exemple-02] mais avec un nouveau nom.

Nous allons maintenant introduire la bibliothèque [Android Annotations] que nous appellerons par facilité **AA**. Cette bibliothèque introduit de nouvelles classes pour annoter les sources Android. Ces annotations vont être utilisées par un processeur qui va créer de nouvelles classes Java dans le module, classes qui participeront à la compilation de celui-ci au même titre que les classes écrites par le développeur. On a ainsi la chaîne de compilation suivante :



Nous allons tout d'abord mettre dans le fichier [build.gradle] les dépendances sur le compilateur d'annotations AA (processeur ci-dessus) :

```
1. def AAVersion = '4.0.0'
2.
3. dependencies {
4.     apt "org.androidannotations:androidannotations:$AAVersion"
5.     compile "org.androidannotations:androidannotations-api:$AAVersion"
6.     compile 'com.android.support:appcompat-v7:23.4.0'
7.     compile fileTree(dir: 'libs', include: ['*.jar'])
```

```
8. }
```

- les lignes 4-5 ajoutent les deux dépendances qui forment la bibliothèque AA ;

Le fichier [build.gradle] est modifié de nouveau pour utiliser un plugin appelé [android-apt] qui modifie le processus de compilation en deux étapes :

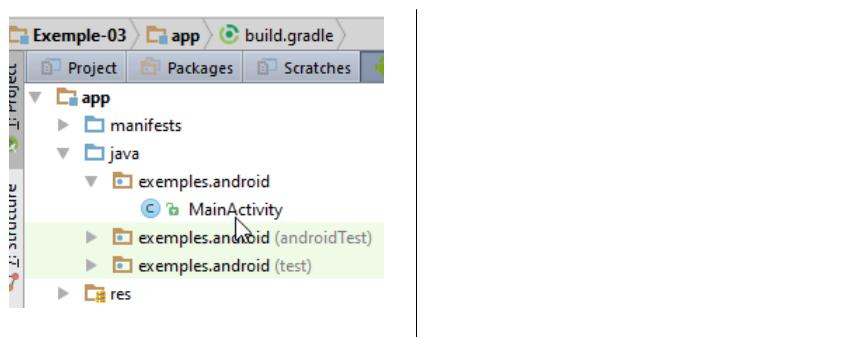
- traitement des annotations Android, ce qui donne naissance à de nouvelles classes ;
- compilation de l'ensemble des classes du projet ;

```
1. buildscript {
2.     repositories {
3.         mavenCentral()
4.     }
5.
6.     dependencies {
7.         // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
8.         classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
9.     }
10. }
11.
12. apply plugin: 'com.android.application'
13. apply plugin: 'android-apt'
```

- ligne 8 : version du plugin [android-apt] qui sera cherchée dans le dépôt Maven central (ligne 3) ;
- ligne 13 : activation de ce plugin ;

A ce stade, vérifiez que la configuration d'exécution [app] fonctionne toujours.

Nous allons maintenant introduire une première annotation de la bibliothèque AA dans la classe [MainActivity] :



La classe [MainActivity] est pour l'instant la suivante :

```
1. package exemples.android;
2.
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5.
6. public class MainActivity extends AppCompatActivity {
7.
8.     @Override
9.     protected void onCreate(Bundle savedInstanceState) {
10.         super.onCreate(savedInstanceState);
11.         setContentView(R.layout.activity_main);
12.     }
13. }
```

Nous avons déjà expliqué ce code au paragraphe 1.3.3 page 29. Nous le modifions de la façon suivante :

```
1. package exemples.android;
2.
3. import android.os.Bundle;
4. import android.support.v7.app.AppCompatActivity;
5. import org.androidannotations.annotations.EActivity;
6.
7. @EActivity(R.layout.activity_main)
8. public class MainActivity extends AppCompatActivity {
9.
10.    @Override
11.    protected void onCreate(Bundle savedInstanceState) {
12.        super.onCreate(savedInstanceState);
13.    }
14. }
```

14. }

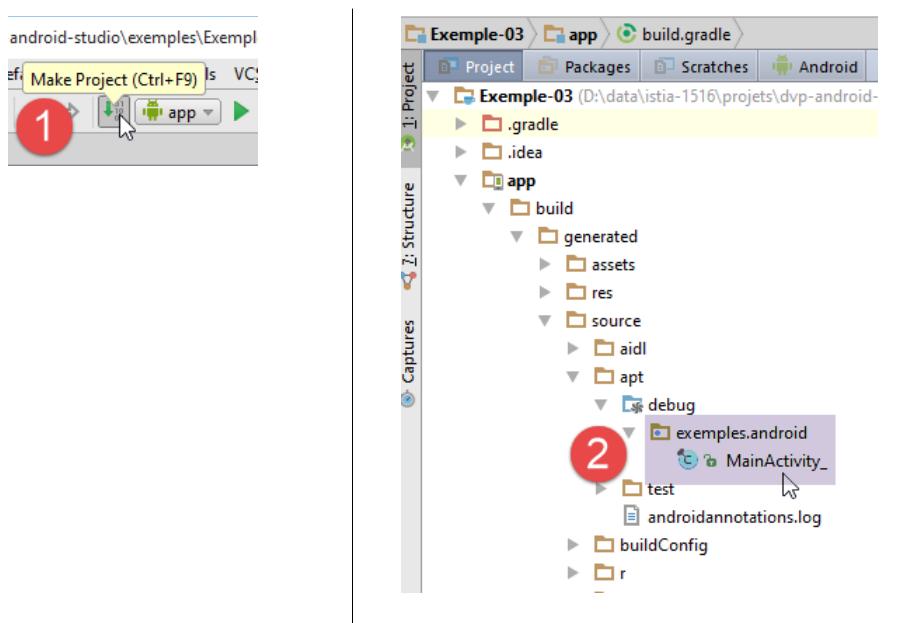
- ligne 7 : l'annotation `[@EActivity]` est une annotation AA (ligne 3). Son paramètre est la vue associée à l'activité ;

Cette annotation va produire une classe `[MainActivity_]` dérivée de la classe `[MainActivity]` et c'est cette classe qui sera la véritable activité. Nous devons donc modifier le manifeste du projet `[AndroidManifest.xml]` de la façon suivante :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.           package="exemples.android">
4.
5.     <application
6.         android:allowBackup="true"
7.         android:icon="@mipmap/ic_launcher"
8.         android:label="@string/app_name"
9.         android:supportsRtl="true"
10.        android:theme="@style/AppTheme">
11.        <activity android:name=".MainActivity_">
12.            <intent-filter>
13.                <action android:name="android.intent.action.MAIN"/>
14.
15.                <category android:name="android.intent.category.LAUNCHER"/>
16.            </intent-filter>
17.        </activity>
18.    </application>
19.
20. </manifest>
```

- ligne 11 : la nouvelle activité ;

Ceci fait, nous pouvons compiler le projet [1] :



- en [2], on voit la classe `[MainActivity_]` générée dans le dossier `[app / build / generated / source / apt / debug]` ;

La classe `[MainActivity_]` générée est la suivante :

```
1. /**
2.  * DO NOT EDIT THIS FILE.
3.  * Generated using AndroidAnnotations 4.0.0.
4.  */
5. // You can create a larger work that contains this file and distribute that work under terms of your choice.
6. //
7.
8.
9. package exemples.android;
10.
11. import android.app.Activity;
12. import android.content.Context;
13. import android.os.Build.VERSION;
14. import android.os.Build.VERSION_CODES;
```

```

15. import android.os.Bundle;
16. import android.support.v4.app.ActivityCompat;
17. import android.view.View;
18. import android.view.ViewGroup.LayoutParams;
19. import org.androidannotations.api.builder.ActivityIntentBuilder;
20. import org.androidannotations.api.builder.PostActivityStarter;
21. import org.androidannotations.api.view.HasViews;
22. import org.androidannotations.api.view.OnViewChangedNotifier;
23.
24. public final class MainActivity_
25.     extends MainActivity
26.     implements HasViews
27. {
28.     private final OnViewChangedNotifier onViewChangedNotifier_ = new OnViewChangedNotifier();
29.
30.     @Override
31.     public void onCreate(Bundle savedInstanceState) {
32.         OnViewChangedNotifier previousNotifier = OnViewChangedNotifier.replaceNotifier(onViewChangedNotifier_);
33.         init_(savedInstanceState);
34.         super.onCreate(savedInstanceState);
35.         OnViewChangedNotifier.replaceNotifier(previousNotifier);
36.         setContentView(R.layout.activity_main);
37.     }
38. ...

```

- lignes 24-25 : la classe [MainActivity_] étend la classe [MainActivity] ;

Nous ne chercherons pas à expliquer le code des classes générées par AA. Elles gèrent la complexité que les annotations cherchent à cacher. Mais il peut être parfois bon de l'examiner lorsqu'on veut comprendre comment sont 'traduites' les annotations qu'on utilise.

On peut désormais exécuter de nouveau la configuration [app]. On obtient le même résultat qu'auparavant. Nous allons désormais partir de ce projet que nous dupliqueraisons pour présenter les notions importantes de la programmation Android.

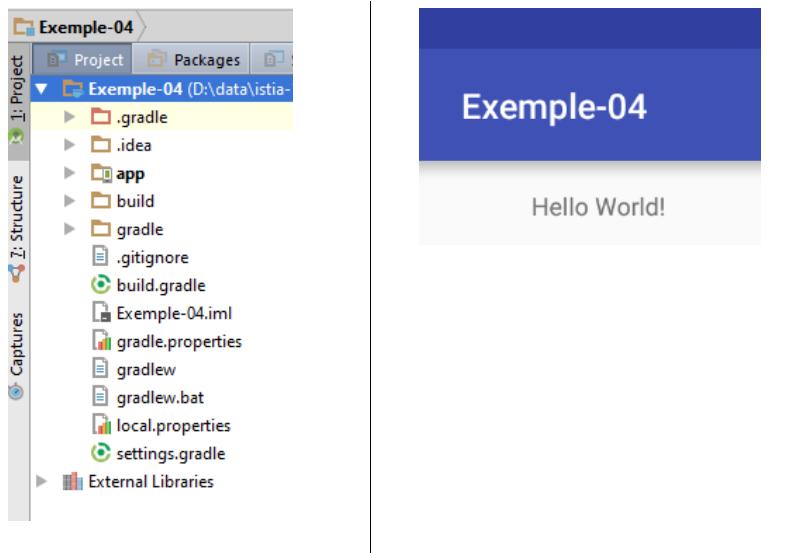
1.5 Exemple-04 : vues et événements

1.5.1 Cration du projet

On suivra la procure dcrite pour dupliquer [Exemple-02] dans [Exemple-03] au paragraphe 1.4, page 36 :

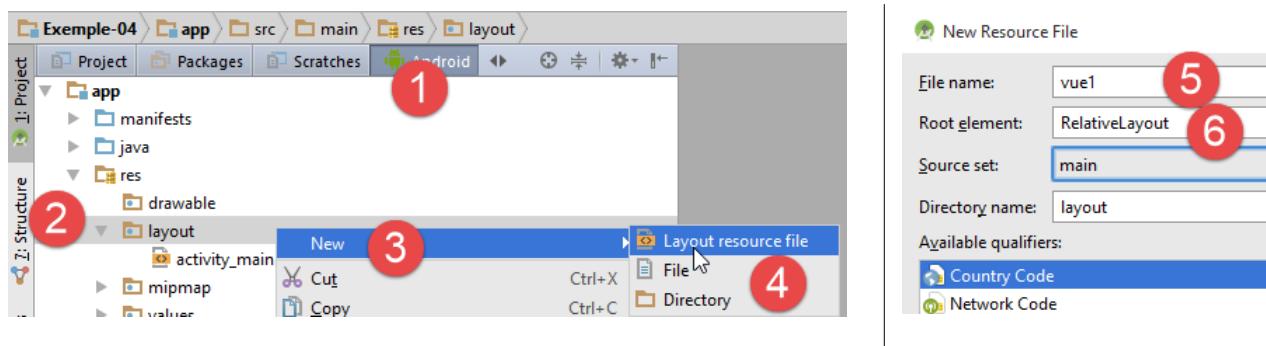
Nous :

- dupliquons le projet [Exemple-03] dans [Exemple-04] (aprs avoir supprim le dossier [app / build] de [Exemple-03]) ;
- chargeons le projet [Exemple-04] ;
- changeons le nom du projet dans le fichier [app / res / values / strings.xml] (perspective Android) ;
- supprimons le fichier [Exemple-04 / Exemple-04.iml] (perspective Project) ;
- compilons puis exutons le projet ;

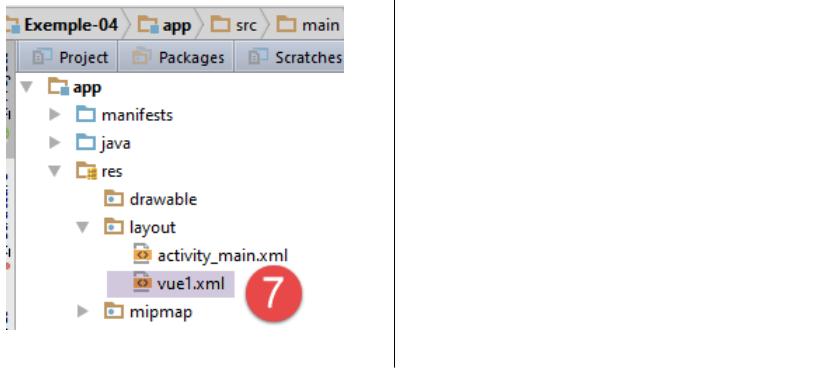


1.5.2 Construire une vue

Nous allons maintenant modifier, avec l'iteur graphique, la vue affiche par le projet [Exemple-04] :



- en [1-4], crez une nouvelle vue XML ;
- en [5], nommez la vue ;
- en [6], indiquez la balise racine de la vue. Ici, nous choisissons un conteneur [RelativeLayout]. Dans ce conteneur de composants, ceux-ci sont placs les uns par rapport aux autres : " droite de ", " gauche de ", "au-dessous de ", "au-dessus de " ;

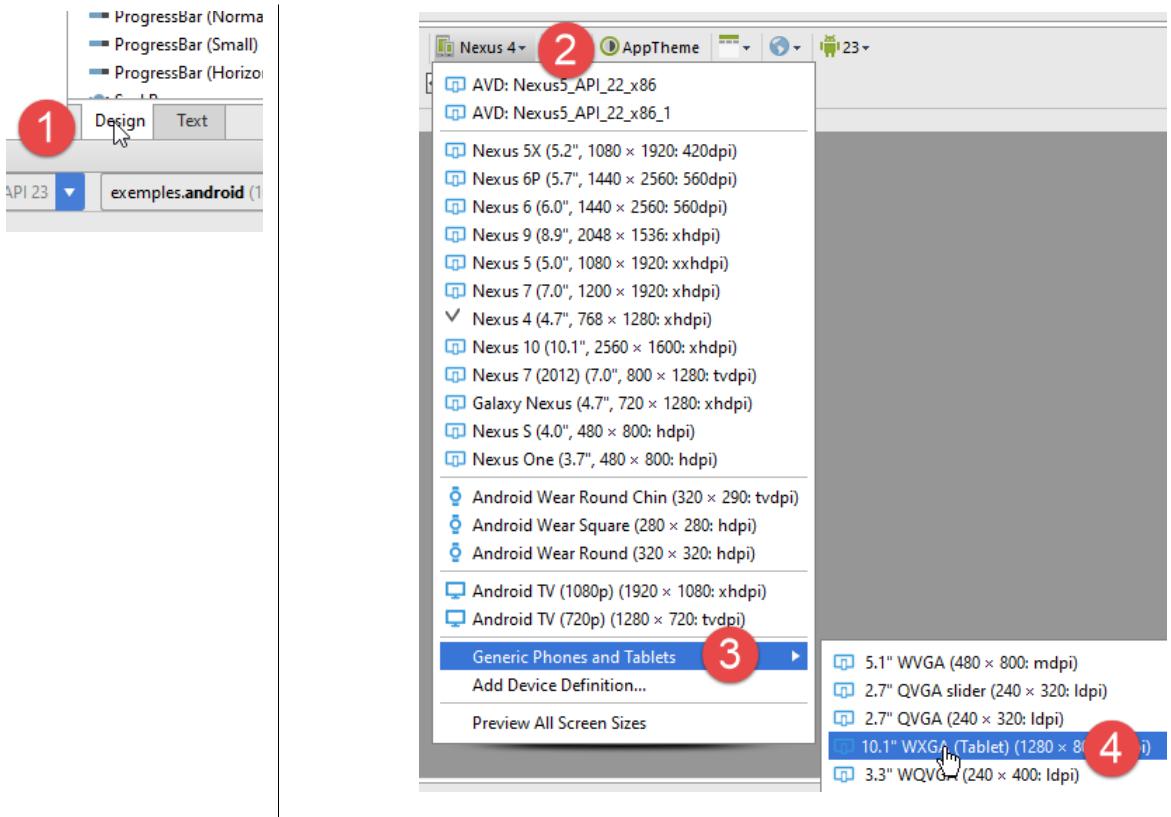


Le fichier [vue1.xml] généré [7] est le suivant :

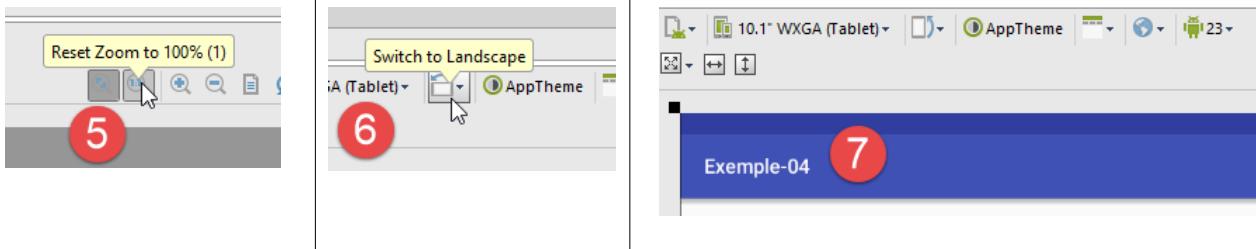
```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent">
5.
6. </RelativeLayout>
```

- ligne 2 : un conteneur [RelativeLayout] vide qui occupera toute la largeur de la tablette (ligne 3) et toute sa hauteur (ligne 4) ;



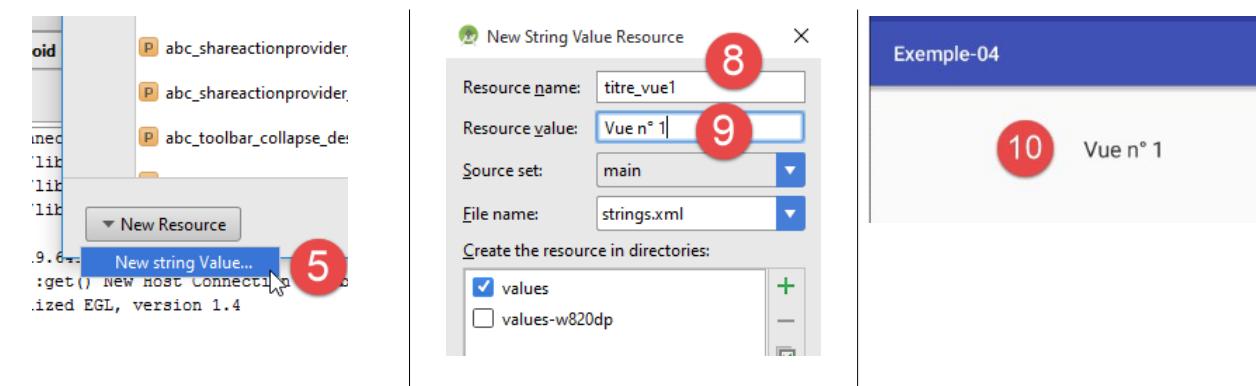
- en [1], sélectionnez l'onglet [Design] dans la vue [vue1.xml] affichée ;
- en [2-4], mettez-vous en mode tablette ;



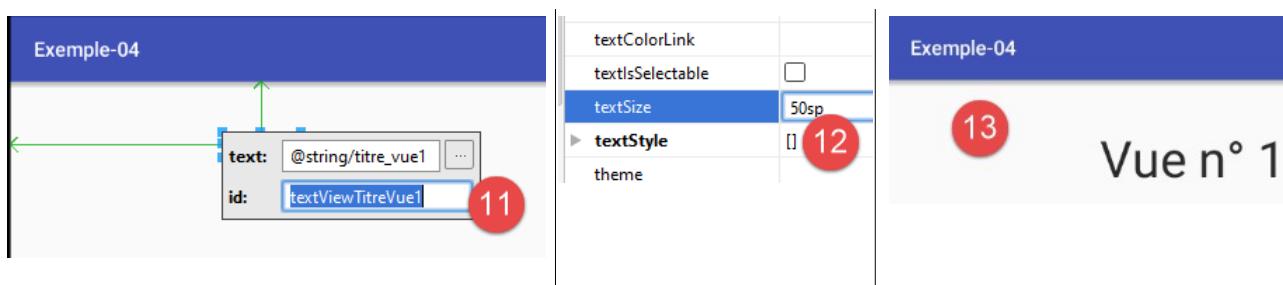
- en [5], mettez-vous à l'échelle 1 de la tablette ;
- en [6], choisissez le mode 'paysage' pour la tablette ;
- la copie d'écran [7] résume les choix faits.



- en [1], prendre un [Large Text] et le tirer sur la vue [2] ;
- en [3], double-cliquer sur le composant ;
- en [4], modifier le texte affiché. Plutôt que de le mettre en 'dur' dans la vue XML, nous allons l'externaliser dans le fichier [res / values / string.xml]



- en [5], on ajoute une nouvelle valeur dans le fichier [strings.xml] ;
- en [8], on donne un identifiant à la chaîne ;
- en [9], on donne la valeur de la chaîne ;
- en [10], la nouvelle vue après validation de l'étape précédente ;



- après un double clic sur le composant, on change son identifiant [11] ;
- en [12], dans les propriétés du composant, on change la taille des caractères [50sp] ;

- en [13], la nouvelle vue ;

Le fichier [vue1.xml] a évolué comme suit :

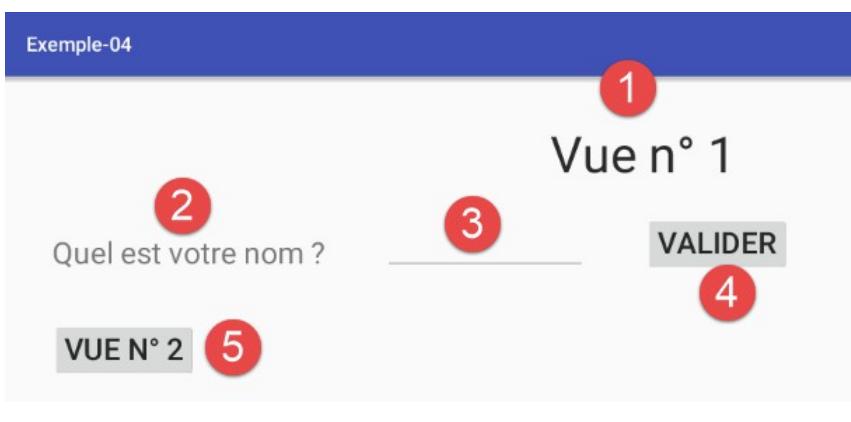
```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent">
5.
6.     <TextView
7.         android:layout_width="wrap_content"
8.         android:layout_height="wrap_content"
9.         android:textAppearance="?android:attr/textAppearanceLarge"
10.        android:text="@string/titre_vue1"
11.        android:id="@+id/textViewTitreVue1"
12.        android:layout_marginLeft="213dp" android:layout_marginStart="213dp"
13.        android:layout_marginTop="50dp" android:layout_alignParentTop="true" android:layout_alignParentLeft="true"
14.        android:layout_alignParentStart="true" android:textSize="50sp"/>
15. </RelativeLayout>
```

- les modifications faites dans l'interface graphique sont aux lignes 10, 11 et 14. Les autres attributs du [TextView] sont des valeurs par défaut ou bien découlent du positionnement du composant dans la vue ;
- lignes 7-8 : la taille du composant est celle du texte qu'elle contient (wrap_content) en hauteur et largeur ;
- ligne 13 : le haut du composant est aligné avec le haut de la vue (ligne 13), 50 pixels dessous (ligne 13) ;
- ligne 12 : le côté gauche du composant est aligné avec la gauche de la vue (ligne 13), 213 pixels à droite (ligne 12) ;

En général, les tailles exactes des marges gauche, droite, haute et basse seront fixées directement dans le XML.

En procédant de la même façon, créez la vue suivante [1] :



Les composants sont les suivants :

Nº	Id	Type	Rôle
1	textViewTitreVue1	TextView	Titre de la vue
2	textView1	TextView	une question
3	editTextNom	EditText	saisie d'un nom
4	buttonValider	Button	pour valider la saisie
5	buttonVue2	Button	pour passer à la vue n° 2

Placer les composants les uns par rapport aux autres est un exercice qui peut s'avérer frustrant, les réactions de l'éditeur graphique étant parfois surprenantes. Il peut être préférable d'utiliser les propriétés des composants :

Le composant [textView1] doit être placé à 50 pixels sous le titre et à 50 pixels du bord gauche du conteneur :

layout:alignComponent [top:bottom]		layout:alignParent [left]		layout:margin [?, 50dp, 50dp, ?, ?, ?, ?]
top:top		top	<input type="checkbox"/>	all
top:bottom	Ab textViewTitreVue1 - @	left	<input checked="" type="checkbox"/>	left 50dp
left:left		bottom	<input type="checkbox"/>	top 50dp
left:right	1	right	<input type="checkbox"/>	right
bottom:bottom				bottom

- en [1], le bord supérieur (top) du composant est aligné par rapport au bord inférieur (bottom) du composant [textViewTitreVue1] à une distance de 50 pixels [3] (top) ;
- en [2], le bord gauche (left) du composant est aligné par rapport au bord gauche du conteneur à une distance de 50 pixels [3] (left) ;

Le composant [editTextNom] doit être placé à 60 pixels à droite du composant [textView1] et aligné par le bas sur ce même composant ;

layout:alignComponent [left:right, bottom:bottom]	layout:margin [?, 60dp, ?, ?, ?, ?, ?]
top:top	all
top:bottom	1
left:left	left 60dp
left:right	Ab textView1 - @
bottom:bottom	Ab textView1 - @

- en [1], le bord gauche (left) du composant est aligné par rapport au bord droit (right) du composant [textView1] à une distance de 60 pixels [2] (left). Il est aligné sur le bord inférieur (bottom:bottom) du composant [textView1] [1] ;

Le composant [buttonValidator] doit être placé à 60 pixels à droite du composant [editTextNom] et aligné par le bas sur ce même composant ;

layout:alignComponent [left:right, bottom:bottom]	layout:margin [?, 60dp, ?, ?, ?, ?, ?]
top:top	all
top:bottom	1
left:left	left 60dp
left:right	Ab editTextNom
bottom:bottom	Ab editTextNom

- en [1], le bord gauche (left) du composant est aligné par rapport au bord droit (right) du composant [editTextNom] à une distance de 60 pixels [2] (left). Il est aligné sur le bord inférieur du composant (bottom:bottom) [editTextNom] [1] ;

Le composant [buttonVue2] doit être placé à 50 pixels sous le composant [textView1] et aligné par la gauche sur ce même composant ;

layout:alignComponent [top:bottom, left:left]	layout:margin [?, ?, 50dp, ?, ?, ?, ?]
top:top	all
top:bottom	Ab textView1 - @
left:left	left 50dp
left:right	1
	top
	right
	bottom

- en [1], le bord gauche (left) du composant est aligné par rapport au bord gauche (left) du composant [*textView1*] et est placé dessous (top:bottom) à une distance de 50 pixels [2] (top) ;

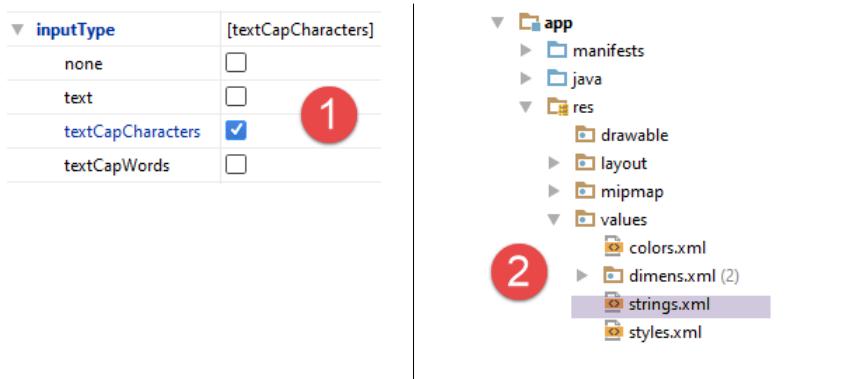
Le fichier XML généré est le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2.
3. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
4.         android:layout_width="match_parent"
5.         android:layout_height="match_parent">
6.
7.     <TextView
8.         android:layout_width="wrap_content"
9.         android:layout_height="wrap_content"
10.        android:textAppearance="?android:attr/textAppearanceLarge"
11.        android:text="@string/titre_vue1"
12.        android:id="@+id/textViewTitreVue1"
13.        android:layout_marginTop="49dp"
14.        android:textSize="50sp"
15.        android:layout_gravity="center|left"
16.        android:layout_alignParentTop="true"
17.        android:layout_centerHorizontal="true"/>
18.
19.    <TextView
20.        android:layout_width="wrap_content"
21.        android:layout_height="wrap_content"
22.        android:text="@string/txt_nom"
23.        android:id="@+id/textView1"
24.        android:layout_below="@+id/textViewTitreVue1"
25.        android:layout_alignParentLeft="true"
26.        android:layout_marginLeft="50dp"
27.        android:layout_marginTop="50dp"
28.        android:textSize="30sp"/>
29.
30.    <EditText
31.        android:layout_width="wrap_content"
32.        android:layout_height="wrap_content"
33.        android:id="@+id/editTextNom"
34.        android:minWidth="200dp"
35.        android:layout_toRightOf="@+id/textView1"
36.        android:layout_marginLeft="60dp"
37.        android:layout_alignBottom="@+id/textView1"
38.        android:inputType="textCapCharacters"/>
39.
40.    <Button
41.        android:layout_width="wrap_content"
42.        android:layout_height="wrap_content"
43.        android:text="@string/btn_valider"
44.        android:id="@+id/buttonValider"
45.        android:layout_alignBottom="@+id/editTextNom"
46.        android:layout_toRightOf="@+id/editTextNom"
47.        android:textSize="30sp"
48.        android:layout_marginLeft="60dp"/>
49.
50.    <Button
51.        android:layout_width="wrap_content"
52.        android:layout_height="wrap_content"
53.        android:text="@string/btn_vue2"
54.        android:id="@+id/buttonVue2"
55.        android:layout_below="@+id/textView1"
56.        android:layout_alignLeft="@+id/textView1"
57.        android:layout_marginTop="50dp"
58.        android:textSize="30sp"/>
59.
60. </RelativeLayout>
```

On y retrouve tout ce qui a été fait de façon graphique. Une autre façon de créer une vue est alors d'écrire directement ce fichier. Lorsqu'on est habitué, cela peut être plus rapide que d'utiliser l'éditeur graphique.

- ligne 38, on trouve une information que nous n'avons pas montrée. Elle est donnée via les propriétés du composant [*editTextNom*] [1] :



Tous les textes proviennent du fichier [strings.xml] [2] suivant :

```

1. <resources>
2.   <string name="app_name">Exemple-04</string>
3.   <string name="titre_vue1">Vue n° 1</string>
4.   <string name="txt_nom">Quel est votre nom ?</string>
5.   <string name="btn_valider">Valider</string>
6.   <string name="btn_vue2">Vue n° 2</string>
7. </resources>

```

Maintenant, modifions l'activité [MainActivity] pour que cette vue soit affichée au démarrage de l'application :

```

1. package exemples.android;
2.
3. import android.os.Bundle;
4. import android.support.v7.app.AppCompatActivity;
5. import org.androidannotations.annotations.EActivity;
6.
7. @EActivity(R.layout.vue1)
8. public class MainActivity extends AppCompatActivity {
9.
10.    @Override
11.    protected void onCreate(Bundle savedInstanceState) {
12.        super.onCreate(savedInstanceState);
13.    }
14. }

```

- ligne 7 : c'est la vue [vue1.xml] qui est désormais affichée par l'activité ;

Modifiez le fichier [AndroidManifest.xml] de la façon suivante :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.           package="exemples.android">
4.   <application
5.     android:allowBackup="true"
6.     android:icon="@mipmap/ic_launcher"
7.     android:label="@string/app_name"
8.     android:supportsRtl="true"
9.     android:theme="@style/AppTheme">
10.    <activity
11.      android:name=".MainActivity_"
12.      android:windowSoftInputMode="stateHidden">
13.      <intent-filter>
14.        <action android:name="android.intent.action.MAIN"/>
15.        <category android:name="android.intent.category.LAUNCHER"/>
16.      </intent-filter>
17.    </activity>
18.  </application>
19. </manifest>

```

- ligne 12 : cette ligne de configuration empêche le clavier d'apparaître dès l'affichage de la vue [vue1]. En effet celle-ci a un champ de saisie qui a le focus lors de l'affichage de la vue. Ce focus fait apparaître par défaut le clavier virtuel ;

Exécutez l'application et vérifiez que c'est bien la vue [vue1.xml] qui est affichée :

Vue n° 1

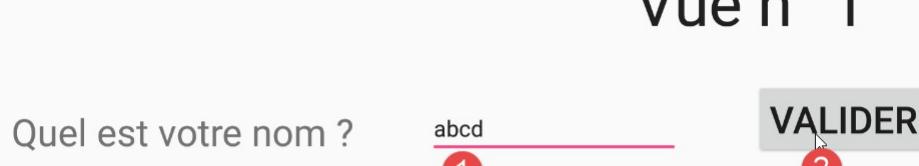
Quel est votre nom ?

VALIDER

VUE N° 2

1.5.3 Gestion des événements

Gérons maintenant le clic sur le bouton [Valider] de la vue [Vue1] :



VUE N° 2

3

Bonjour abcd

Le code de [MainActivity] évolue comme suit :

```

1. package exemples.android;
2.
3. import android.os.Bundle;
4. import android.support.v7.app.AppCompatActivity;
5. import android.util.Log;
6. import android.widget.EditText;
7. import android.widget.Toast;
8. import org.androidannotations.annotations.AfterViews;
9. import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EActivity;
11. import org.androidannotations.annotations.ViewById;
12.
13. @EActivity(R.layout.vue1)
14. public class MainActivity extends AppCompatActivity {
15.

```

```

16. // les éléments de l'interface visuelle
17. @ViewById(R.id.editTextNom)
18. protected EditText editTextNom;
19.
20. @Override
21. protected void onCreate(Bundle savedInstanceState) {
22.     Log.d("MainActivity", "onCreate");
23.     super.onCreate(savedInstanceState);
24. }
25.
26. @AfterViews
27. protected void afterViews(){
28.     Log.d("MainActivity", "afterViews");
29. }
30.
31. // gestionnaire d'évt
32. @Click(R.id.buttonValider)
33. protected void doValidator() {
34.     // on affiche le nom saisi
35.     Toast.makeText(this, String.format("Bonjour %s", editTextNom.getText().toString()), Toast.LENGTH_LONG).show();
36. }
37.
38. }

```

- lignes 17-18 : on associe le champ [**protected** EditText editTextNom] au composant d'identifiant [R.id.editTextNom] de l'interface visuelle. Le champ associé au composant doit être accessible dans la classe dérivée [MainActivity_] et pour cette raison ne peut être de portée [private]. Le champ identifié par [R.id.editTextNom] provient de la vue [vue1.xml] :

```

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editTextNom"
    android:minWidth="200dp"
    android:layout_toRightOf="@+id/textView1"
    android:layout_marginLeft="60dp"
    android:layout_alignBottom="@+id/textView1"
    android:inputType="textCapCharacters"/>

```

Note : ne pas mettre de caractères accentués dans les identifiants [id]. AA ne les gère pas correctement.

- ligne 32 : l'annotation [**@Click(R.id.buttonValider)**] désigne la méthode qui gère l'événement 'Click' sur le bouton d'identifiant [R.id.buttonValider]. Cet identifiant provient également de la vue [vue1.xml] :

```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn_valider"
    android:id="@+id/buttonValider"
    android:layout_alignBottom="@+id/editTextNom"
    android:layout_toRightOf="@+id/editTextNom"
    android:textSize="30sp"
    android:layout_marginLeft="60dp"/>

```

- ligne 35 : affiche le nom saisi :
 - Toast.makeText(...).show()* : affiche un texte à l'écran,
 - le 1^{er} paramètre de *makeText* est l'activité,
 - le second paramètre est le texte à afficher dans la boîte qui va être affichée par *makeText*,
 - le troisième paramètre est la durée de vie de la boîte affichée : *Toast.LENGTH_LONG* ou *Toast.LENGTH_SHORT*;
- ligne 26, l'annotation [**@AfterViews**] annote la méthode à exécuter lorsque tous les champs annotés par [**@ViewById**] sont initialisés. Il est important de savoir quand ces champs sont initialisés. Par exemple, est-ce que dans la méthode [onCreate] on peut utiliser la référence de la ligne 18 ? Pour répondre à cette question, nous avons mis des logs ;

Exécutez le projet [Exemple-04] et vérifiez qu'il se passe quelque chose lorsque vous cliquez sur le bouton [Valider]. Nous obtenons les logs suivants :

```

1. 05-28 09:06:23.751 571-571/exemples.android D/MainActivity: onCreate
2. 05-28 09:06:23.841 571-571/exemples.android D/MainActivity: afterViews

```

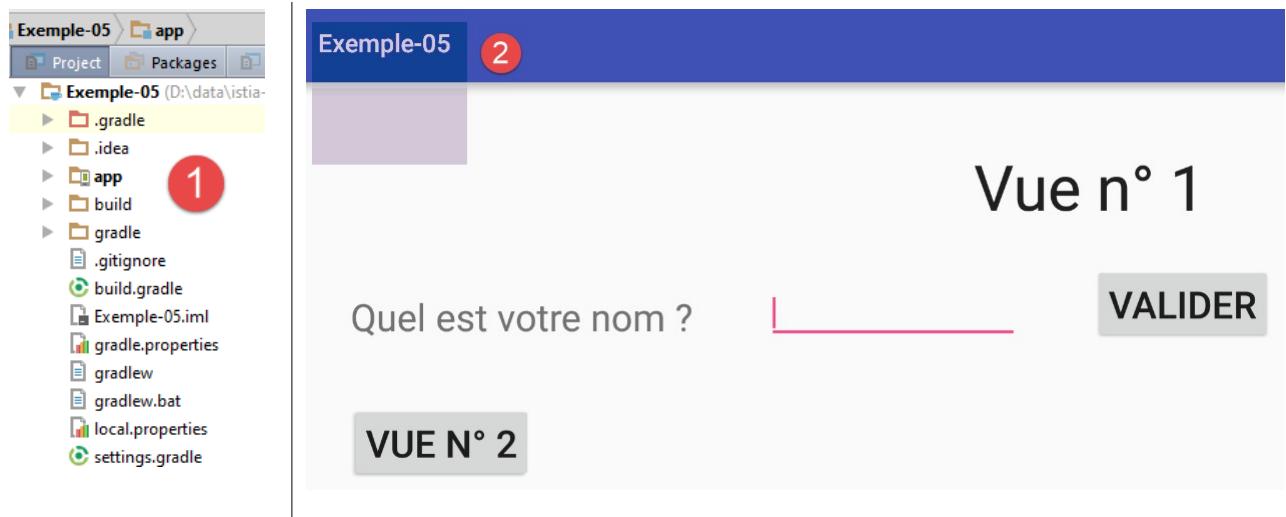
On en conclut que lorsque la méthode [onCreate] s'exécute, les champs annotés par [**@ViewById**] ne sont pas encore initialisés. De nouveau, le lecteur débutant est encouragé à mettre ce type de logs dans les méthodes qui gèrent le cycle de vie de l'application.

1.6 Exemple-05 : navigation entre vues

Dans le projet précédent, le bouton [Vue n° 2] n'a pas été exploité. On se propose de l'exploiter en créant une seconde vue et en montrant comment naviguer d'une vue à l'autre. Il y a plusieurs façons de résoudre ce problème. Celle qui est proposée ici est d'associer chaque vue à une activité. Une autre méthode est d'avoir une unique activité de type [AppCompatActivity] qui affiche des vues de type [Fragment]. Ce sera la méthode utilisée dans des applications à venir.

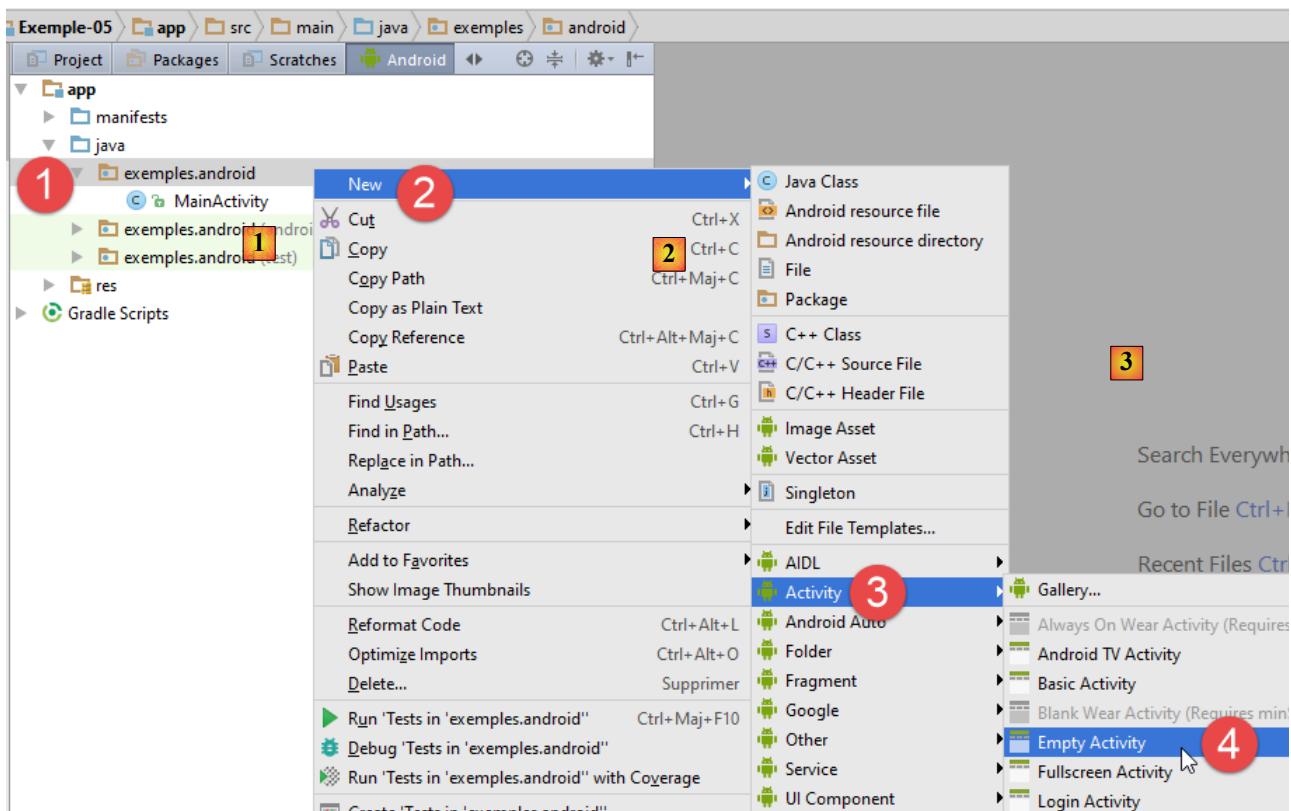
1.6.1 Crédit du projet

On duplique le projet [Exemple-04] dans [Exemple-05]. Pour cela, on suivra la procédure décrite pour dupliquer [Exemple-02] dans [Exemple-03] au paragraphe 1.4, page 36 et qui a été reproduite dans le paragraphe 1.5, page 42.

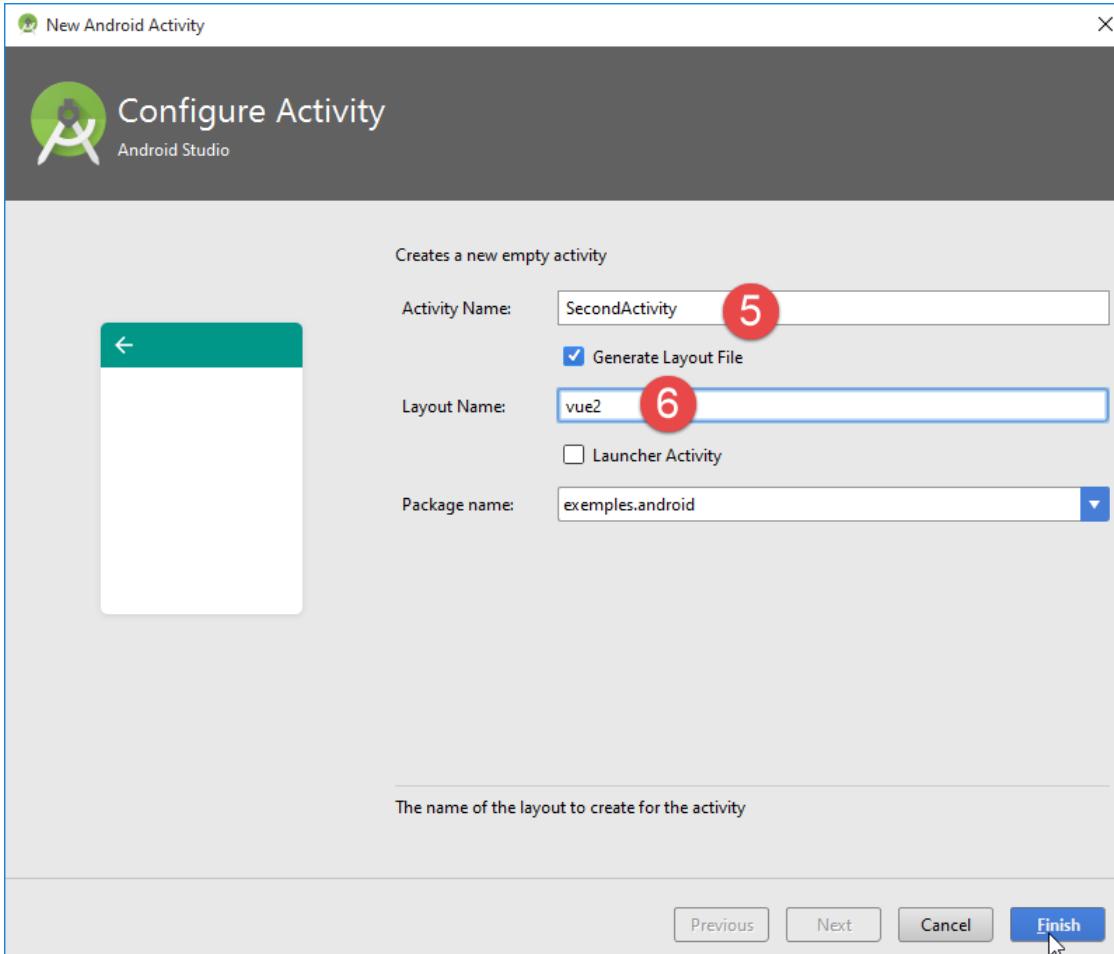


1.6.2 Ajout d'une seconde activité

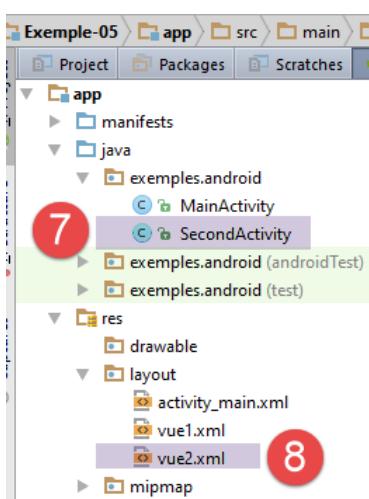
Pour gérer une seconde vue, nous allons créer une seconde activité. C'est elle qui gèrera la vue n° 2. On est là dans un modèle **une vue = une activité**. Il y a d'autres modèles possibles.



- en [1-4], on crée un nouvelle activité ;



- en [5], le nom de la classe qui sera générée ;
- en [6], le nom de la vue (vue2.xml) associée à la nouvelle activité ;



- en [7-8], les fichiers impactés par la configuration précédente ;

L'activité [SecondActivity] est la suivante :

```
1. package exemples.android;
2.
3. import android.support.v7.app.AppCompatActivity;
```

```

4. import android.os.Bundle;
5.
6. public class SecondActivity extends AppCompatActivity {
7.
8.     @Override
9.     protected void onCreate(Bundle savedInstanceState) {
10.         super.onCreate(savedInstanceState);
11.         setContentView(R.layout.vue2);
12.     }
13. }
```

- ligne 11 : l'activité est associée à la vue [vue2.xml] ;

La vue [vue2.xml] est la suivante :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     android:paddingLeft="@dimen/activity_horizontal_margin"
8.     android:paddingRight="@dimen/activity_horizontal_margin"
9.     android:paddingTop="@dimen/activity_vertical_margin"
10.    android:paddingBottom="@dimen/activity_vertical_margin"
11.    tools:context="exemples.android.SecondActivity">
12.
13. </RelativeLayout>
```

C'est une vue pour l'instant vide avec un gestionnaire de disposition de type [RelativeLayout] (ligne 2). Ligne 11, on voit qu'elle a été associée à la nouvelle activité.

Le manifeste du module Android [AndroidManifest.xml] a évolué de la façon suivante :

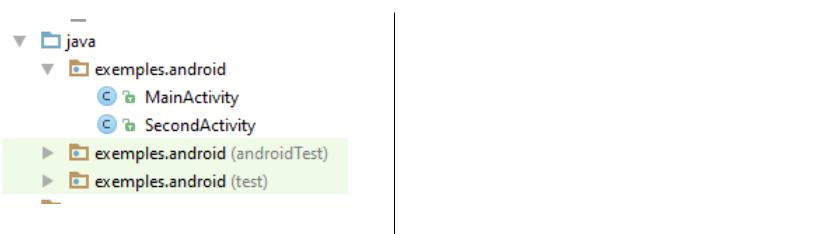
```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.             package="exemples.android">
4.
5.     <application
6.         android:allowBackup="true"
7.         android:icon="@mipmap/ic_launcher"
8.         android:label="@string/app_name"
9.         android:supportsRtl="true"
10.        android:theme="@style/AppTheme">
11.            <activity
12.                android:name=".MainActivity"
13.                android:windowSoftInputMode="stateHidden">
14.                    <intent-filter>
15.                        <action android:name="android.intent.action.MAIN"/>
16.
17.                        <category android:name="android.intent.category.LAUNCHER"/>
18.                    </intent-filter>
19.            </activity>
20.            <activity android:name=".SecondActivity">
21.            </activity>
22.        </application>
23.
24. </manifest>
```

Ligne 20 , une seconde activité a été enregistrée.

1.6.3 Navigation de la vue n° 1 à la vue n° 2

Revenons au code de la classe [MainActivity] qui affiche la vue n° 1. Le passage à la vue n° 2 n'est actuellement pas géré :



Nous la gérons de la façon suivante :

```

1. // naviguer vers la vue n° 2
2. @Click(R.id.buttonVue2)
3. protected void navigateToView2() {
4.     // on navigue vers la vue n° 2 en lui passant le nom saisi dans la vue n° 1
5.     // on crée un Intent
6.     Intent intent = new Intent();
7.     // on associe cet Intent à une activité
8.     intent.setClass(this, SecondActivity.class);
9.     // on associe des informations à cet Intent
10.    intent.putExtra("NOM", editTextNom.getText().toString().trim());
11.    // on lance l'Intent, ici une activité de type [SecondActivity]
12.    startActivity(intent);
13. }

```

- lignes 2-3 : la méthode [navigateToView2] gère le 'clic' sur le bouton identifié par [R.id.buttonVue2] défini dans la vue [vue1.xml] :

```

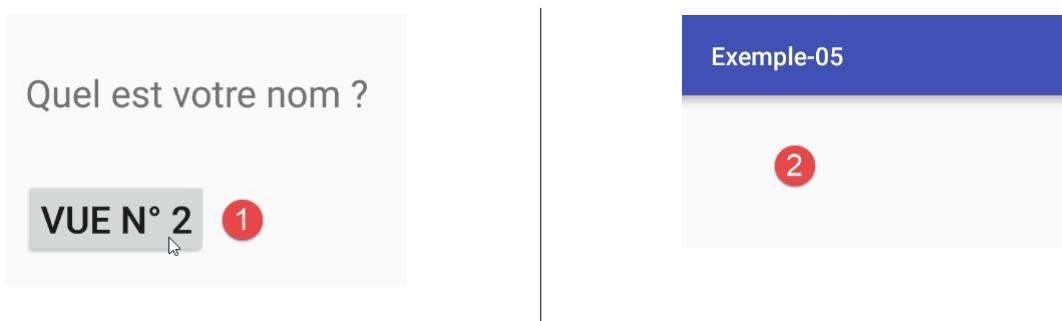
1. <Button
2.     android:layout_width="wrap_content"
3.     android:layout_height="wrap_content"
4.     android:text="@string/btn_vue2"
5.     android:id="@+id/buttonVue2"
6.     android:layout_below="@+id/textView1"
7.     android:layout_alignLeft="@+id/textView1"
8.     android:layout_marginTop="50dp"
9.     android:textSize="30sp"/>

```

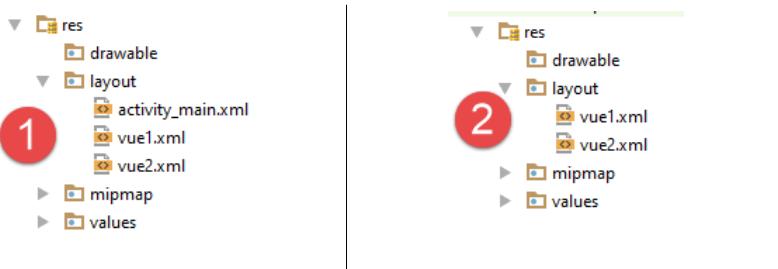
Les commentaires décrivent les étapes à réaliser pour le changement de vue :

- ligne 6 : créer un objet de type [Intent]. Cet objet va permettre de préciser et l'activité à lancer et les informations à lui passer ;
- ligne 8 : associer l'Intent à une activité, ici une activité de type [SecondActivity] qui sera chargée d'afficher la vue n° 2. Il faut se souvenir que l'activité [MainActivity] affiche elle la vue n° 1. Donc on a **une vue = une activité**. Il nous faudra définir le type [SecondActivity] ;
- ligne 10 : de façon facultative, mettre des informations dans l'objet [Intent]. Celles-ci sont destinées à l'activité [SecondActivity] qui va être lancée. Les paramètres de [Intent.putExtra] sont (Object clé , Object valeur). On notera que la méthode [EditText.getText()] qui rend le texte saisi dans la zone de saisie ne rend pas un type [String] mais un type [Editable]. Il faut utiliser la méthode [toString] pour avoir le texte saisi ;
- ligne 12 : lancer l'activité définie par l'objet [Intent].

Exécutez le projet [Exemple-05] et vérifiez que vous obtenez bien la vue n° 2 (vide pour l'instant) :



1.6.4 Construction de la vue n° 2



- en [1-2], nous supprimons la vue [main.xml] qui ne nous sert plus, puis nous modifions la vue [vue2.xml] de la façon suivante :



Les composants sont les suivants :

Nº	Id	Type	Rôle
1	<code>textViewTitreVue2</code>	TextView	Titre de la vue
2	<code>textViewBonjour</code>	TextView	un texte
5	<code>btn_vue1</code>	Button	pour passer à la vue n° 1

Le fichier XML [vue2.xml] est le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout
3.   xmlns:android="http://schemas.android.com/apk/res/android"
4.   xmlns:tools="http://schemas.android.com/tools"
5.   android:layout_width="match_parent"
6.   android:layout_height="match_parent"
7.   android:paddingLeft="@dimen/activity_horizontal_margin"
8.   android:paddingRight="@dimen/activity_horizontal_margin"
9.   android:paddingTop="@dimen/activity_vertical_margin"
10.  android:paddingBottom="@dimen/activity_vertical_margin"
11.  tools:context="exemples.android.SecondActivity">
12.
13.
14.  <TextView
15.    android:layout_width="wrap_content"
16.    android:layout_height="wrap_content"
17.    android:textAppearance="?android:attr/textAppearanceLarge"
18.    android:text="@string/titre_vue2"
19.    android:id="@+id/textViewTitreVue2"
20.    android:layout_marginTop="50dp"
21.    android:textSize="50sp"
22.    android:layout_gravity="center/left"
23.    android:layout_alignParentTop="true"
24.    android:layout_centerHorizontal="true"/>
25.
26.  <TextView
27.    android:layout_width="wrap_content"
28.    android:layout_height="wrap_content"
29.    android:id="@+id/textViewBonjour"
30.    android:layout_centerVertical="true"
31.    android:layout_alignParentLeft="true"
32.    android:layout_below="@+id/textViewTitreVue2"
33.    android:layout_marginTop="50dp"
34.    android:layout_marginLeft="50dp"
35.    android:textSize="30sp"
36.    android:text="Bonjour !"
37.    android:textColor="#ffffb91b"/>
38.
39.  <Button
40.    android:layout_width="wrap_content"
41.    android:layout_height="wrap_content"
42.    android:text="@string/btn_vue1"
43.    android:id="@+id/buttonVue1"
44.    android:layout_marginTop="50dp"
45.    android:textSize="30sp"
46.    android:layout_alignLeft="@+id/textViewBonjour"
47.    android:layout_below="@+id/textViewBonjour"/>
48.
49. </RelativeLayout>
```

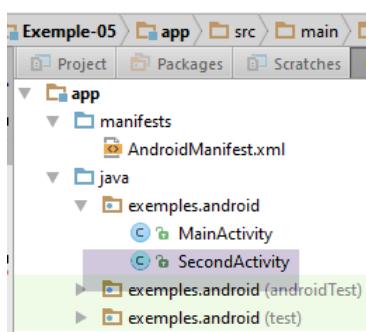
Exécutez le projet [Exemple-05] et vérifiez que vous obtenez bien la nouvelle vue en cliquant sur le bouton [Vue n° 2].

1.6.5 L'activité [SecondActivity]

Dans [MainActivity], nous avons écrit le code suivant :

```
1.     // naviguer vers la vue n° 2
2.     protected void navigateToView2() {
3.         // on navigue vers la vue n° 2 en lui passant le nom saisi dans la vue n° 1
4.         // on crée un Intent
5.         Intent intent = new Intent();
6.         // on associe cet Intent à une activité
7.         intent.setClass(this, SecondActivity.class);
8.         // on associe des informations à cet Intent
9.         intent.putExtra("NOM", edtNom.getText().toString().trim());
10.        // on lance l'Intent, ici une activité de type [SecondActivity]
11.        startActivity(intent);
12.    }
```

Ligne 9, nous avons mis pour [SecondActivity] des informations qui n'ont pas été exploitées. Nous les exploitons maintenant et cela se passe dans le code de [SecondActivity] :



Le code de [SecondActivity] évolue comme suit :

```
1. package exemples.android;
2.
3. import android.content.Intent;
4. import android.support.v7.app.AppCompatActivity;
5. import android.os.Bundle;
6. import android.widget.TextView;
7. import org.androidannotations.annotations.AfterViews;
8. import org.androidannotations.annotations.EActivity;
9. import org.androidannotations.annotations.ViewById;
10.
11. @EActivity(R.layout.vue2)
12. public class SecondActivity extends AppCompatActivity {
13.
14.     // composants de l'interface visuelle
15.     @ViewById
16.     protected TextView textViewBonjour;
17.
18.     @Override
19.     protected void onCreate(Bundle savedInstanceState) {
20.         super.onCreate(savedInstanceState);
21.     }
22.
23.     @AfterViews
24.     protected void afterViews() {
25.         // on récupère l'intent s'il existe
26.         Intent intent = getIntent();
27.         if (intent != null) {
28.             Bundle extras = intent.getExtras();
29.             if (extras != null) {
30.                 // on récupère le nom
31.                 String nom = extras.getString("NOM");
32.                 if (nom != null) {
33.                     // on l'affiche
34.                     textViewBonjour.setText(String.format("Bonjour %s !", nom));
35.                 }
36.             }
37.         }
38.     }
39.
40. }
```

- ligne 11 : on utilise l'annotation `[@EActivity]` pour indiquer que la classe `[SecondActivity]` est une activité associée à la vue `[vue2.xml]` ;
- lignes 15-16 : on récupère une référence sur le composant `[TextView]` identifié par `[R.id.textViewBonjour]`. Ici, on n'a pas écrit `[@ViewById(R.id.textViewBonjour)]`. Dans ce cas, AA suppose que l'identifiant du composant est identique au champ annoté, ici le champ `[textViewBonjour]` ;
- ligne 23 : l'annotation `[@AfterViews]` annote une méthode qui doit être exécutée après que les champs annotés par `[@ViewById]` ont été initialisés. Dans la méthode `[OnCreate]` (ligne 19), on ne peut pas utiliser ces champs car ils n'ont pas encore été initialisés. Dans le projet `[Exemple-05]`, on bascule d'une activité à une autre et il n'était a priori pas clair si la méthode annotée `[@AfterViews]` allait être exécutée une fois à l'instanciation initiale de l'activité ou à chaque fois que l'activité est démarrée. Les tests ont montré que la seconde hypothèse était vérifiée ;
- ligne 26 : la classe `[AppCompatActivity]` a une méthode `[getIntent]` qui rend l'objet `[Intent]` associé à l'activité ;
- ligne 28 : la méthode `[Intent.getExtras]` rend un type `[Bundle]` qui est une sorte de dictionnaire contenant les informations associées à l'objet `[Intent]` de l'activité ;
- ligne 31 : on récupère le nom placé dans l'objet `[Intent]` de l'activité ;
- ligne 34 : on l'affiche.

Rappel : les champs annotés par l'annotation `[@ViewById]` **ne doivent pas comporter de caractères accentués**.

Revenons sur la classe `[SecondActivity]`. Parce que nous avons écrit :

```
1. @EActivity(R.layout.vue2)
2. public class SecondActivity extends AppCompatActivity {
```

AA va générer une classe `[SecondActivity_]` dérivée de `[SecondActivity]` et c'est cette classe qui sera la véritable activité. Cela nous amène à faire des modifications dans :

[MainActivity]

```
1. // naviguer vers la vue n° 2
2. @Click(R.id.buttonVue2)
3. protected void navigateToView2() {
4. ..
5. // on associe cet Intent à une activité
6. intent.setClass(this, SecondActivity_.class);
7. ...
8. }
```

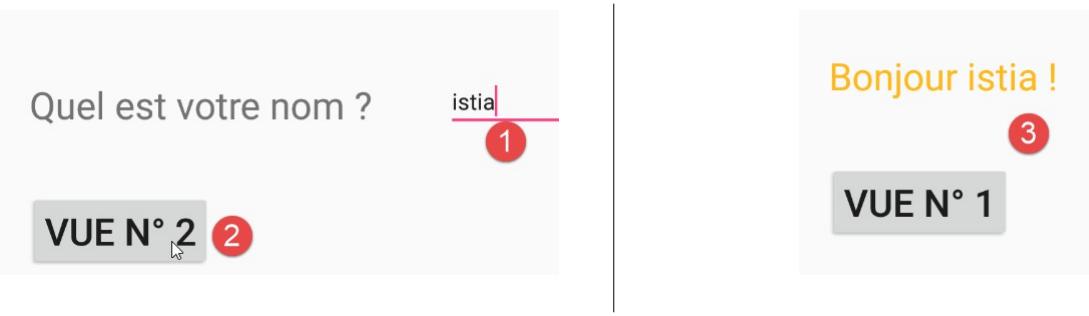
- ligne 6, on doit remplacer `[SecondActivity]` par `[SecondActivity_]` ;

[AndroidManifest.xml]

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.           package="exemples.android">
4.
5.   <application
6.     android:allowBackup="true"
7.     android:icon="@mipmap/ic_launcher"
8.     android:label="@string/app_name"
9.     android:supportsRtl="true"
10.    android:theme="@style/AppTheme">
11.      <activity
12.        android:name=".MainActivity_"
13.        android:windowSoftInputMode="stateHidden">
14.          <intent-filter>
15.            <action android:name="android.intent.action.MAIN"/>
16.
17.            <category android:name="android.intent.category.LAUNCHER"/>
18.          </intent-filter>
19.        </activity>
20.        <activity android:name=".SecondActivity_">
21.        </activity>
22.      </application>
23.
24. </manifest>
```

- ligne 20, on doit remplacer `[SecondActivity]` par `[SecondActivity_]` ;

Testez cette nouvelle version. Tapez un nom dans la vue n° 1 et vérifiez que la vue n° 2 l'affiche bien.



1.6.6 Navigation de la vue n° 2 vers la vue n° 1

Pour naviguer de la vue n° 2 à la vue n° 1 nous allons suivre la procédure vue précédemment :

- mettre le code de navigation dans l'activité [SecondActivity] qui affiche la vue n° 2 ;
- écrire la méthode [@AfterViews] dans l'activité [MainActivity] qui affiche la vue n° 1 ;

Le code de [SecondActivity] évolue comme suit :

```

1.  @Click(R.id.buttonVue1)
2.  protected void navigateToView1() {
3.      // on crée un Intent pour l'activité [MainActivity]
4.      Intent intent1 = new Intent();
5.      intent1.setClass(this, MainActivity_.class);
6.      // on récupère l'Intent de l'activité courante [SecondActivity]
7.      Intent intent2 = getIntent();
8.      if (intent2 != null) {
9.          Bundle extras2 = intent2.getExtras();
10.         if (extras2 != null) {
11.             // on met le nom dans l'Intent de [MainActivity]
12.             intent1.putExtra("NOM", extras2.getString("NOM"));
13.         }
14.         // on lance [MainActivity]
15.         startActivity(intent1);
16.     }
17. }
```

- lignes 1-2 : on associe la méthode [navigateToView1] au clic sur le bouton [btn_vue1] ;
- ligne 4 : on crée un nouvel [Intent] ;
- ligne 5 : associé à l'activité [MainActivity_] ;
- ligne 7 : on récupère l'Intent associé à [SecondActivity] ;
- ligne 9 : on récupère les informations de cet Intent ;
- ligne 12 : la clé [NOM] est récupérée dans [intent2] pour être mise dans [intent1] avec la même valeur associée ;
- ligne 15 : l'activité [MainActivity_] est lancée.

Dans le code de [MainActivity] on ajoute la méthode [@AfterViews] suivante :

```

1.  @AfterViews
2.  protected void afterViews() {
3.      // on récupère l'intent s'il existe
4.      Intent intent = getIntent();
5.      if (intent != null) {
6.          Bundle extras = intent.getExtras();
7.          if (extras != null) {
8.              // on récupère le nom
9.              String nom = extras.getString("NOM");
10.             if (nom != null) {
11.                 // on l'affiche
12.                 editTextNom.setText(nom);
13.             }
14.         }
15.     }
16. }
```

Faites ces modifications et testez votre application. Maintenant quand on revient de la vue n° 2 à la vue n° 1, on doit retrouver le nom saisi initialement, ce qui n'était pas le cas jusqu'à maintenant.

Bonjour android !

VUE N° 1

1

Quel est votre nom ?

android

2

VUE N° 2

1.6.7 Cycle de vie des activités

Nous avons présenté au paragraphe 1.3.5, page 34, le cycle de vie d'une activité. Nous avons ici deux activités et on bascule de l'une à l'autre pendant l'exécution. Ces activités contiennent deux méthodes dont on se sait pas très bien quand elles sont appelées l'une par rapport à l'autre : [onCreate] et [afterViews]. Il est important de le savoir. Pour cela, nous ajoutons des logs dans les deux activités :

Ainsi dans la classe [MainActivity], nous écrivons :

```
1.     // constructeur
2.     public MainActivity() {
3.         Log.d("MainActivity", "constructor");
4.     }
5.
6.     @Override
7.     protected void onCreate(Bundle savedInstanceState) {
8.         Log.d("MainActivity", "onCreate");
9.         ...
10.    }
11.
12.    @AfterViews
13.    protected void afterViews() {
14.        Log.d("MainActivity", "afterViews");
15.        ...
16.    }
17. }
```

- lignes 2-4 : nous voulons savoir si la classe [MainActivity] est instanciée une ou plusieurs fois ;
- ligne 8 : nous voulons savoir si la méthode [onCreate] est appelée une ou plusieurs fois ;
- ligne 14 : nous voulons savoir si la méthode [afterViews] est appelée une ou plusieurs fois ;

Nous faisons exactement de même dans la classe [SecondActivity].

Au démarrage de l'application, nous avons les logs suivants :

```
1. 05-28 09:38:09.429 26711-26711/exemples.android D/MainActivity: constructor
2. 05-28 09:38:09.449 26711-26711/exemples.android D/MainActivity: onCreate
3. 05-28 09:38:09.600 26711-26711/exemples.android D/MainActivity: afterViews
```

Les méthodes [onCreate, afterViews] de la première activité ont été exécutées dans cet ordre. Lorsqu'on clique sur le bouton [Vue n° 2], les nouveaux logs sont les suivants :

```
1. 05-28 09:39:26.607 26711-26711/exemples.android D/SecondActivity: constructor
2. 05-28 09:39:26.608 26711-26711/exemples.android D/SecondActivity: onCreate
3. 05-28 09:39:26.617 26711-26711/exemples.android D/SecondActivity: afterViews
```

Les méthodes [onCreate, afterViews] de la seconde activité ont été exécutées dans cet ordre. Lorsqu'on clique sur le bouton [Vue n° 1], les nouveaux logs sont les suivants :

```
1. 05-28 09:39:56.393 26711-26711/exemples.android D/MainActivity: constructor
2. 05-28 09:39:56.394 26711-26711/exemples.android D/MainActivity: onCreate
3. 05-28 09:39:56.400 26711-26711/exemples.android D/MainActivity: afterViews
```

La classe [MainActivity] est donc de nouveau instanciée. Lorsqu'on clique sur le bouton [Vue n° 2], les nouveaux logs sont les suivants :

```
1. 05-28 09:40:59.099 26711-26711/exemples.android D/SecondActivity: constructor
2. 05-28 09:40:59.102 26711-26711/exemples.android D/SecondActivity: onCreate
```

3. 05-28 09:40:59.113 26711-26711/exemples.android D/SecondActivity: afterViews

La classe [SecondActivity] est donc de nouveau instanciée.

Les deux activités sont donc systématiquement recréées lorsqu'on change d'activité.

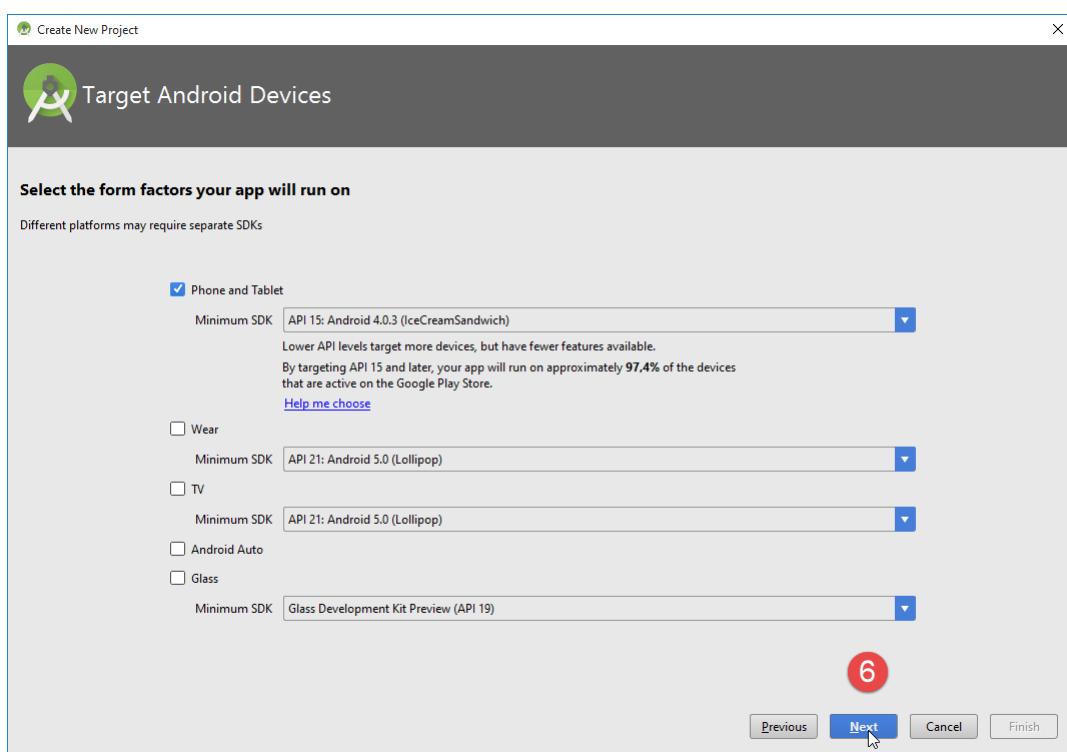
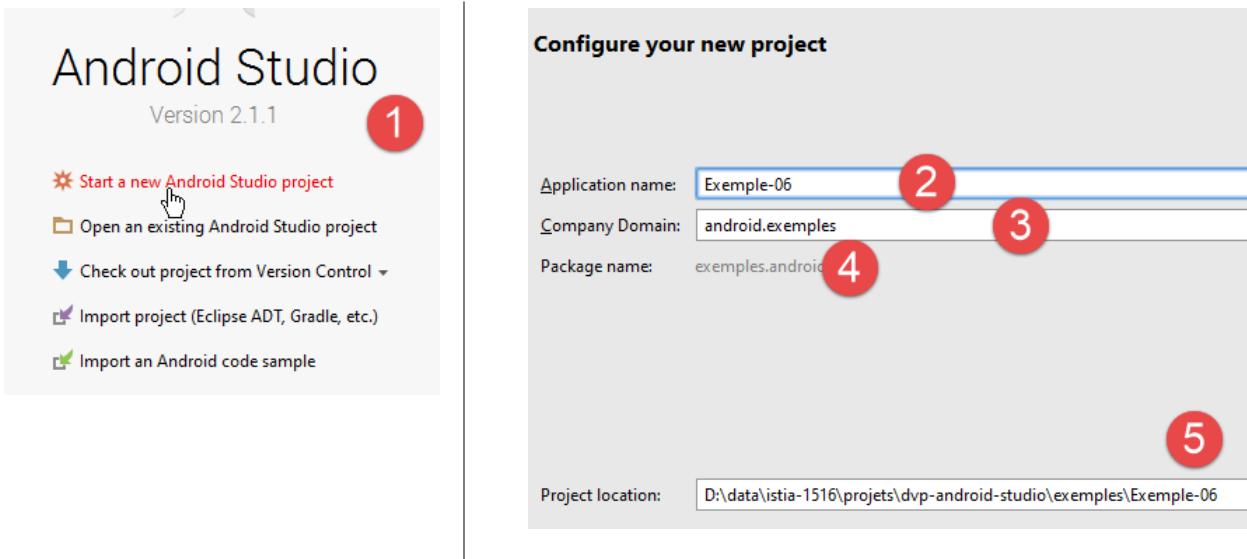
Nous allons découvrir maintenant une architecture avec une unique activité pouvant gérer plusieurs vues appelées **fragments**. L'activité et les vues ne seront instanciées qu'une fois contrairement à la méthode précédente où une activité pouvait être instanciée plusieurs fois.

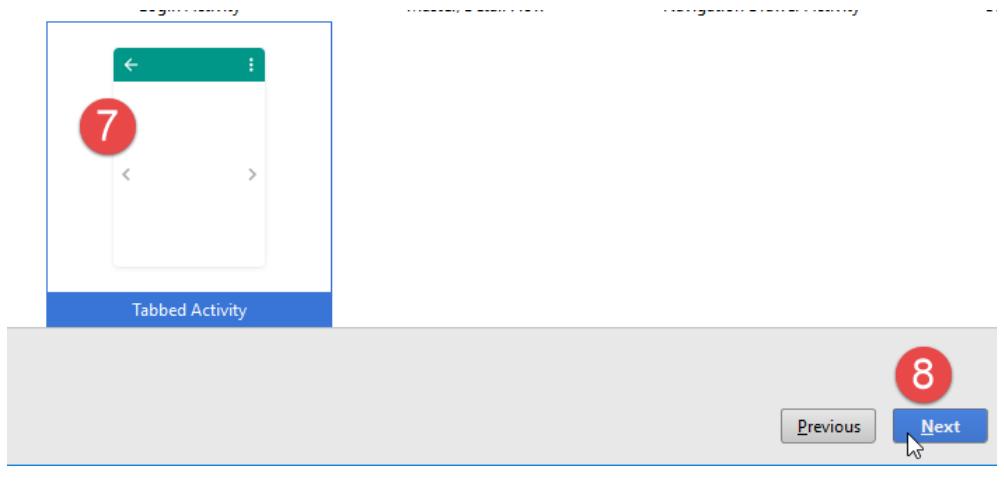
1.7 Exemple-06 : navigation par onglets

Nous allons maintenant explorer les interfaces à onglets. L'exemple est complexe mais introduit tous les éléments que nous allons utiliser par la suite : activité unique, gestionnaire de fragments (vues), conteneur de fragments, navigation entre fragments. La notion d'onglets est différente de celle des fragments et est mineure dans ce qu'on veut montrer dans cet exemple.

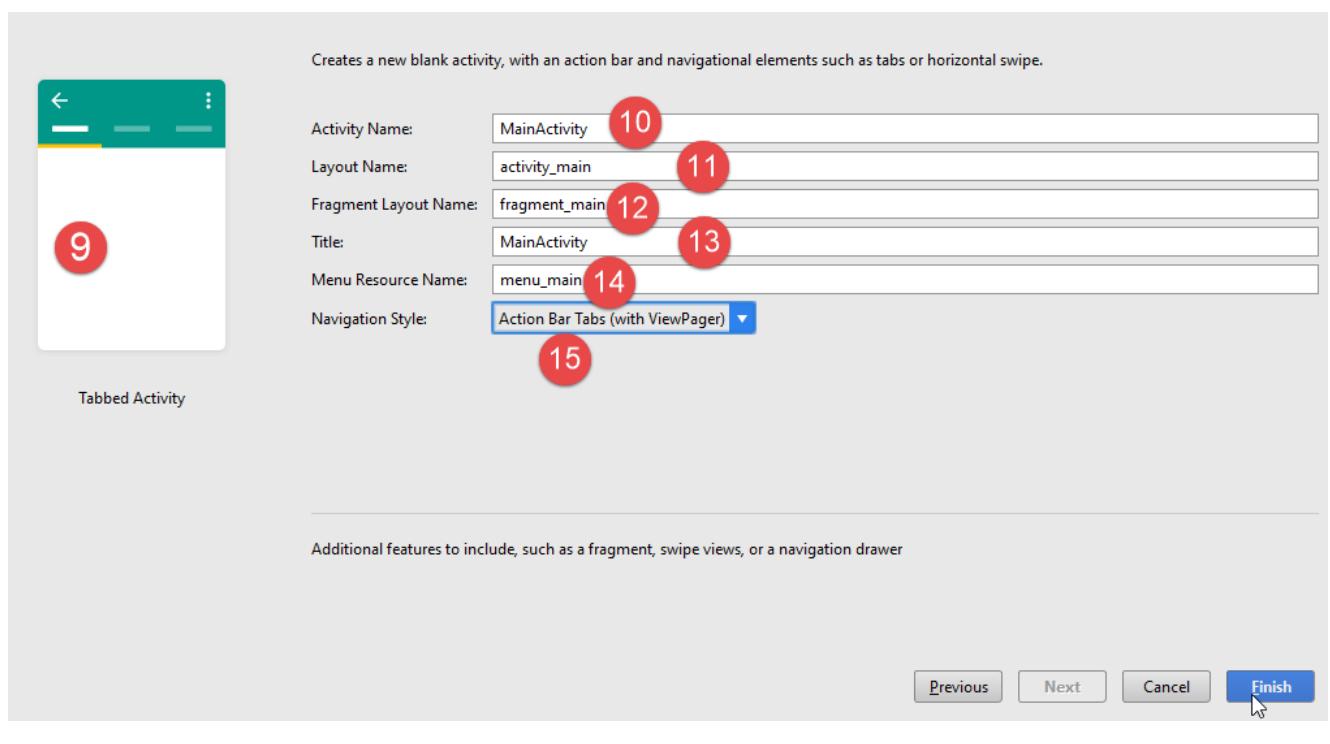
1.7.1 Crédation du projet

Nous créons un nouveau projet :



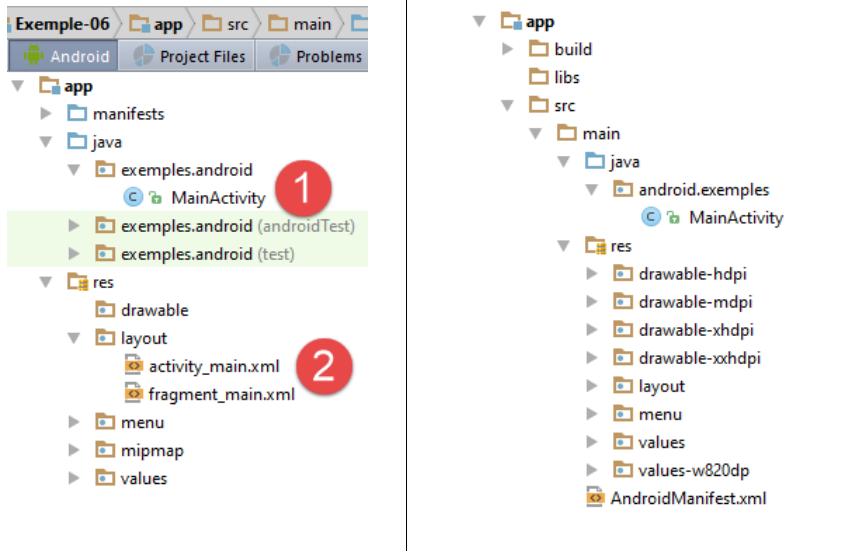


- en [7], on choisit une activité avec onglets (Tabbed Activity) ;



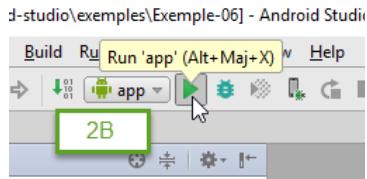
- en [10-14], on garde les valeurs proposées par défaut ;
- en [15], on choisit des onglets avec une barre de titres ;

Le projet créé est alors le suivant :

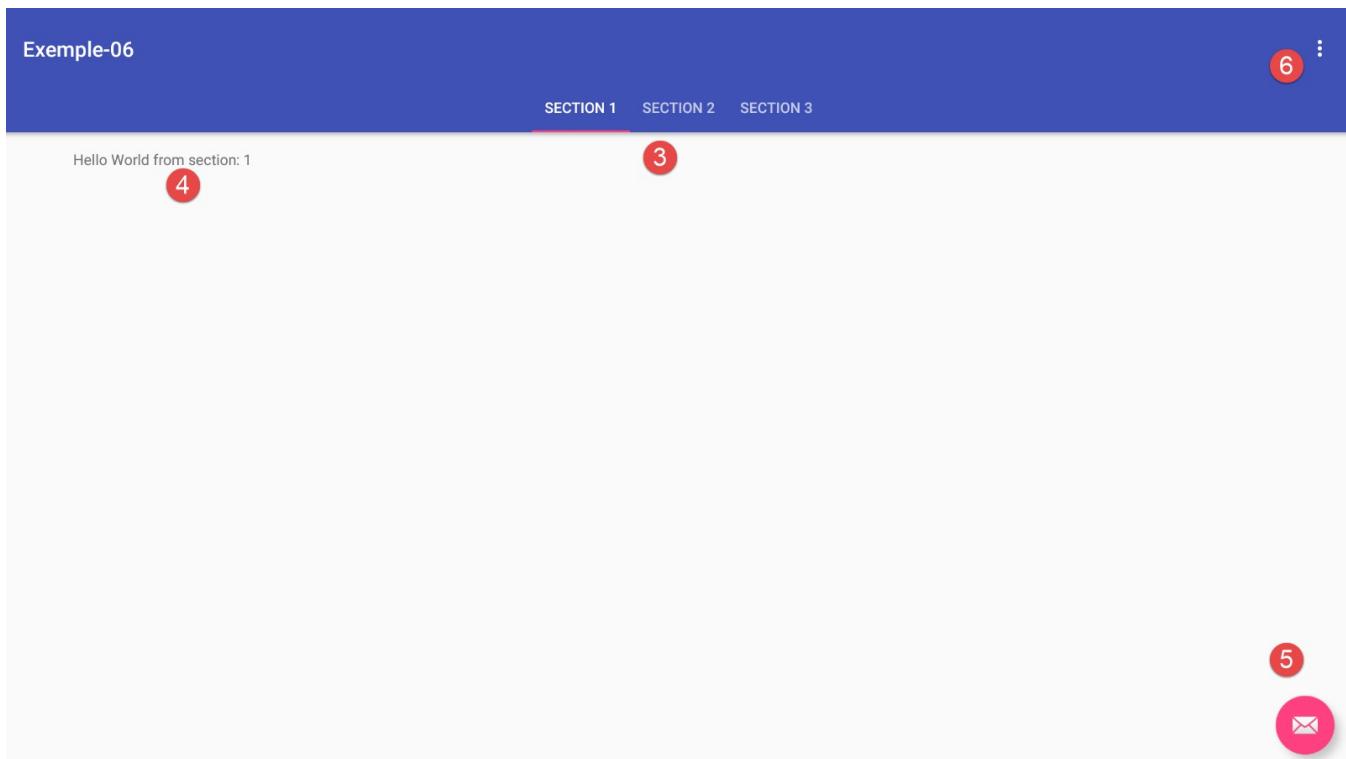


- en [1], l'activité ;
- en [2], les vues ;

Une configuration d'exécution [app], du nom du module, a été automatiquement créée [2b] :

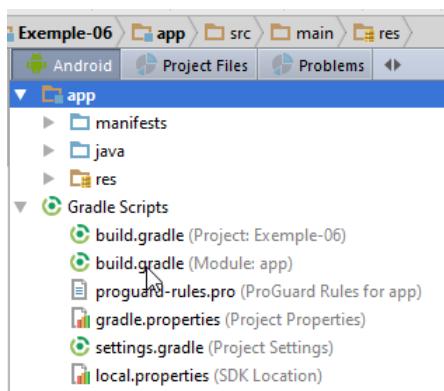


On peut l'exécuter. S'affiche alors une fenêtre avec trois onglets [3-6] :



1.7.2 Configuration Gradle

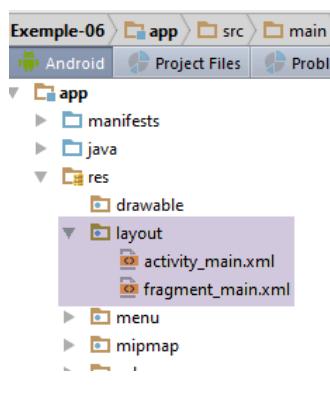
Le projet [Exemple-06] a été généré avec le fichier [build.gradle] suivant :



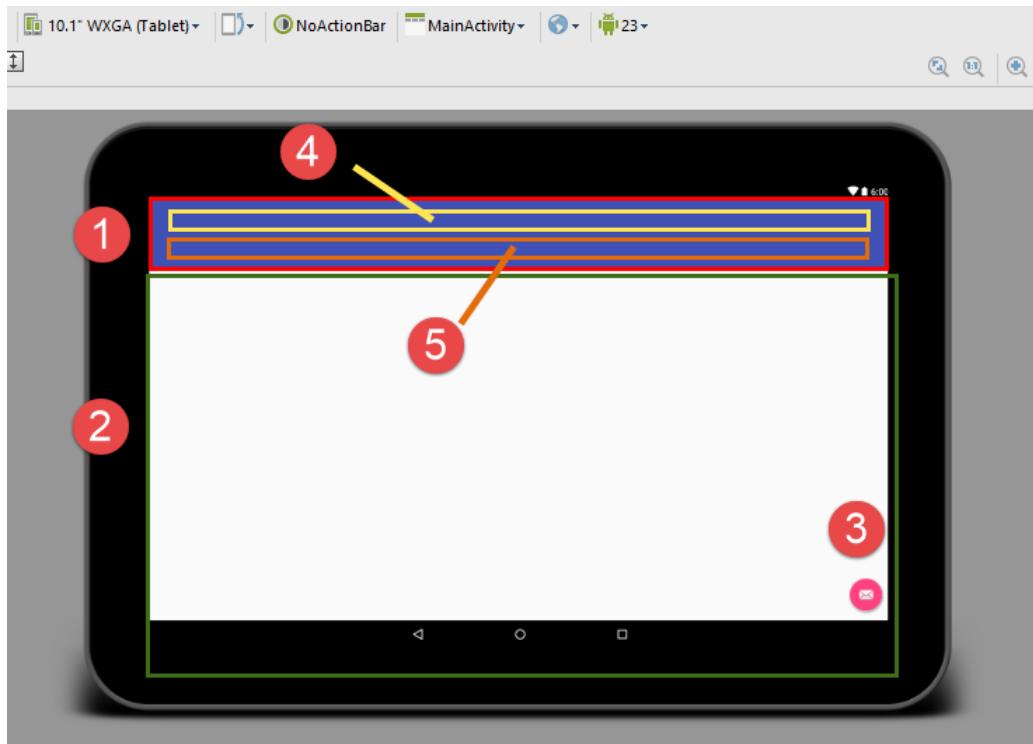
```
1. apply plugin: 'com.android.application'
2.
3. android {
4.     compileSdkVersion 23
5.     buildToolsVersion "23.0.3"
6.     defaultConfig {
7.         applicationId "exemples.android"
8.         minSdkVersion 15
9.         targetSdkVersion 23
10.        versionCode 1
11.        versionName "1.0"
12.    }
13.    buildTypes {
14.        release {
15.            minifyEnabled false
16.            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
17.        }
18.    }
19. }
20.
21. dependencies {
22.     compile fileTree(dir: 'libs', include: ['*.jar'])
23.     testCompile 'junit:junit:4.12'
24.     compile 'com.android.support:appcompat-v7:23.4.0'
25.     compile 'com.android.support:design:23.4.0'
26. }
```

Il y a une nouveauté par rapport à ce qui a déjà été rencontré : la ligne 25. Cette bibliothèque est nécessaire aux nouveaux composants utilisés par l'application générée.

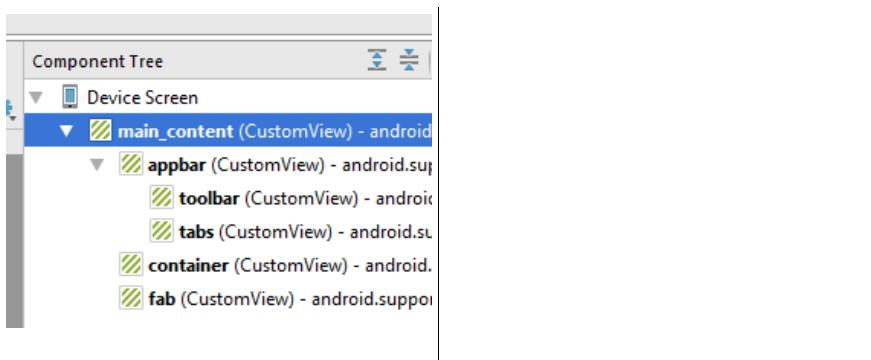
1.7.3 La vue [activity_main]



La vue [activity_main] est la vue associée à l'activité [MainActivity] du projet. En mode [design], la vue est la suivante :



Elle contient les composants suivants :



- [main_content] est la totalité de la vue ;
- [appbar] (encadré rouge, 1) est la barre d'application. Elle contient deux composants :
 - [toolbar] (encadré jaune 4) est la barre d'outils ;
 - [tabs] (encadré orange 5) est la barre de titre des onglets ;
- [container] (encadré vert, 2) peut accueillir divers fragments. Un fragment est une vue. Ainsi, la même activité va pouvoir afficher plusieurs vues (fragments) dans ce conteneur ;
- [fab] (composant 3) est appelé composant flottant ;

En mode [text], le code est le suivant :

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      xmlns:app="http://schemas.android.com/apk/res-auto"
5.      android:id="@+id/main_content"
6.      android:layout_width="match_parent"
7.      android:layout_height="match_parent"
8.      android:fitsSystemWindows="true"
9.      tools:context="exemples.android.MainActivity">
10.
11.    <android.support.design.widget.AppBarLayout
12.        android:id="@+id/appbar"
13.        android:layout_width="match_parent"
14.        android:layout_height="wrap_content"

```

```

15.     android:paddingTop="@dimen/appbar_padding_top"
16.     android:theme="@style/AppTheme.AppBarOverlay">
17.
18.     <android.support.v7.widget.Toolbar
19.         android:id="@+id/toolbar"
20.         android:layout_width="match_parent"
21.         android:layout_height="?attr/actionBarSize"
22.         android:background="?attr/colorPrimary"
23.         app:popupTheme="@style/AppTheme.PopupOverlay"
24.         app:layout_scrollFlags="scroll/enterAlways">
25.
26.     </android.support.v7.widget.Toolbar>
27.
28.     <android.support.design.widget.TabLayout
29.         android:id="@+id/tabs"
30.         android:layout_width="match_parent"
31.         android:layout_height="wrap_content"/>
32.
33. </android.support.design.widget.AppBarLayout>
34.
35. <android.support.v4.view.ViewPager
36.     android:id="@+id/container"
37.     android:layout_width="match_parent"
38.     android:layout_height="match_parent"
39.     app:layout_behavior="@string/appbar_scrolling_view_behavior"/>
40.
41. <android.support.design.widget.FloatingActionButton
42.     android:id="@+id/fab"
43.     android:layout_width="wrap_content"
44.     android:layout_height="wrap_content"
45.     android:layout_gravity="end|bottom"
46.     android:layout_margin="@dimen/fab_margin"
47.     android:src="@android:drawable/ic_dialog_email"/>
48.
49. </android.support.design.widget.CoordinatorLayout>

```

On retrouve les éléments décrits précédemment :

- lignes 2-49 : la définition du composant [main_content] (ligne 5) qui est la totalité de la vue. On voit que c'est un *layout* (gestionnaire de disposition de composants) de type [CoordinatorLayout] (ligne 2) ;
- lignes 11-33 : le conteneur [appbar] (ligne 12). C'est un *layout* de type [AppBarLayout] (ligne 11) ;
- lignes 18-24 : le composant [toolbar] (ligne 19) de type [Toolbar] (ligne 18) ;
- lignes 28-31 : le conteneur [tabs] (ligne 29). C'est un *layout* de type [TabLayout] (ligne 28). Il va afficher les titres des onglets ;
- lignes 35-39 : le composant [container] (ligne 36). C'est ce conteneur qui affiche les différentes vues de l'activité ;
- lignes 41-47 : le composant [fab] (ligne 42) de type [FloatingActionButton] (ligne 41). C'est un bouton sur lequel on peut cliquer. Il se place par défaut en bas à droite de la vue totale ;

Nous n'essaierons pas de comprendre la signification de tous les attributs de ces composants. Nous allons les utiliser tels quels. C'est avec l'expérience et souvent en mode [design] que l'on découvre leur rôle. Dans ce mode, on découvre que les composants ont plusieurs dizaines d'attributs. En général, seuls certains sont initialisés, les autres gardant une valeur par défaut.

Précisons néanmoins quelques points. La plupart des valeurs configurant les différentes vues sont rassemblées dans le dossier [res / values] :



Ces valeurs sont référencées aux lignes 15-16, 23, 39, 46 du fichier [activity_main.xml]. Prenons un exemple :

- ligne 15 :

```
    android:paddingTop="@dimen/appbar_padding_top"
```

L'annotation [@dimen] fait référence au fichier [res / values / dimens.xml] :

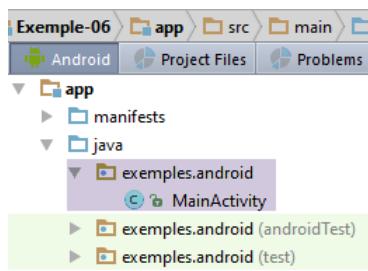
```
a) <resources>
b)   <!-- Default screen margins, per the Android Design guidelines. -->
c)   <dimen name="activity_horizontal_margin">16dp</dimen>
d)   <dimen name="activity_vertical_margin">16dp</dimen>
e)   <dimen name="fab_margin">16dp</dimen>
f)   <dimen name="appbar_padding_top">8dp</dimen>
g) </resources>
```

La ligne 15 du fichier [activity_main.xml] fait référence à la ligne (f) ci-dessus ;

De façon analogue, l'annotation :

- [@string] fait référence au fichier de ressources [res / values / strings.xml] ;
- [@color] fait référence au fichier de ressources [res / values / colors.xml] ;
- [@style] fait référence au fichier de ressources [res / values / styles.xml] ;

1.7.4 L'activité



Le code généré pour l'activité est à la hauteur de la vue décrite précédemment : il est complexe. Nous allons l'analyser en plusieurs étapes.

1.7.4.1 La gestion des fragments et des onglets

Le code de [MainActivity] relatif aux fragments et onglets est le suivant :

```
1. package exemples.android;
2.
3. import android.support.design.widget.TabLayout;
4. import android.support.design.widget.FloatingActionButton;
5. import android.support.design.widget.Snackbar;
6. import android.support.v7.app.AppCompatActivity;
7. import android.support.v7.widget.Toolbar;
8.
9. import android.support.v4.app.Fragment;
10. import android.support.v4.app.FragmentManager;
11. import android.support.v4.app.FragmentPagerAdapter;
12. import android.support.v4.view.ViewPager;
13. import android.os.Bundle;
14. import android.view.LayoutInflater;
15. import android.view.Menu;
16. import android.view.MenuItem;
17. import android.view.View;
18. import android.view.ViewGroup;
19.
20. import android.widget.TextView;
21.
22. public class MainActivity extends AppCompatActivity {
23.
24.     // le gestionnaire de fragments
25.     private SectionsPagerAdapter mSectionsPagerAdapter;
26.
27.     // le conteneur de fragments
28.     private ViewPager mViewPager;
29.
30.     @Override
31.     protected void onCreate(Bundle savedInstanceState) {
32.         // parent
33.         super.onCreate(savedInstanceState);
34.         // vue
35.         setContentView(R.layout.activity_main);
36.         // barre d'outils
```

```

37.     Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
38.     setSupportActionBar(toolbar);
39.     // le gestionnaire de fragments
40.     mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
41.
42.     // le conteneur de fragments est associé au gestionnaire de fragments
43.     // c'est à dire que le fragment n° i du conteneur de fragments est le fragment n° i délivré par le gestionnaire de fragments
44.     mViewPager = (ViewPager) findViewById(R.id.container);
45.     mViewPager.setAdapter(mSectionsPagerAdapter);
46.     // la barre d'onglets est également associée au conteneur de fragments
47.     // ce qui signifie que l'onglet n° i affiche le fragment n° i du conteneur
48.     TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
49.     tabLayout.setupWithViewPager(mViewPager);
50. }
51.
52.
53. // un fragment
54. public static class PlaceholderFragment extends Fragment {
55. ...
56. }
57.
58. // le gestionnaire de fragments
59. // c'est à lui qu'on demande les fragments à afficher dans la vue principale
60. // doit définir les méthodes [getItem] et [getCount] - les autres sont facultatives
61. public class SectionsPagerAdapter extends FragmentPagerAdapter {
62. ...
63. }
64. }
```

- ligne 28 : Android fournit un conteneur de vues de type [android.support.v4.view.ViewPager] (ligne 12). Il faut fournir à ce conteneur un gestionnaire de vues ou fragments. C'est le développeur qui le fournit ;
- ligne 25 : le gestionnaire de fragments utilisé dans cet exemple. Son implémentation est aux lignes 61-63 ;
- ligne 31 : la méthode exécutée à la création de l'activité ;
- ligne 35 : la vue [activity_main.xml] est associée à l'activité ;
- ligne 37 : on récupère la référence du composant [toolbar] de la vue via son identifiant ;
- ligne 38 : cette barre d'outils devient la barre d'action (une notion Android) de l'activité ;
- ligne 40 : le gestionnaire de fragments est instancié. Le paramètre du constructeur est la classe Android [android.support.v4.app.FragmentManager] (ligne 10) ;
- ligne 44 : on récupère dans la vue [activity_main.xml] la référence du conteneur de fragments via son identifiant ;
- ligne 45 : le gestionnaire de fragments est lié au conteneur de fragments. Cela signifie que lorsqu'on demandera au conteneur de fragments d'afficher le fragment n° i, celui sera demandé au gestionnaire de fragments ;
- ligne 48 : on récupère une référence sur la barre d'onglets via son identifiant ;
- ligne 49 : le gestionnaire d'onglets est associé au conteneur de fragments. Cela signifie que lorsqu'on cliquera sur l'onglet n° i, le conteneur affichera le fragment n° i. L'association faite entre le gestionnaire d'onglets et le conteneur de fragments nous évite toute gestion des onglets. Ainsi n'avons nous pas à définir de gestionnaire d'événement pour le clic sur un onglet. L'association avec le conteneur de fragments le fournit par défaut. Nous verrons un exemple où il y aura plus de fragments que d'onglets. Dans ce cas, on ne fait pas cette association.

Le gestionnaire de fragments [SectionsPagerAdapter] est le suivant :

```

1. // le gestionnaire de fragments
2. // c'est à lui qu'on demande les fragments à afficher dans la vue principale
3. // doit définir les méthodes [getItem] et [getCount] - les autres sont facultatives
4. public class SectionsPagerAdapter extends FragmentPagerAdapter {
5.
6.     public SectionsPagerAdapter(FragmentManager fm) {
7.         super(fm);
8.     }
9.
10.    // fragment n° position
11.    @Override
12.    public Fragment getItem(int position) {
13.        // on instancie un fragment [PlaceHolder] et on le rend
14.        return PlaceholderFragment.newInstance(position + 1);
15.    }
16.
17.    // rend le nombre de fragments générés
18.    @Override
19.    public int getCount() {
20.        return 3;
21.    }
22.
23.    // facultatif - donne un titre aux fragments générés
24.    @Override
25.    public CharSequence getPageTitle(int position) {
26.        switch (position) {
27.            case 0:
28.                return "SECTION 1";
```

```

29.         case 1:
30.             return "SECTION 2";
31.         case 2:
32.             return "SECTION 3";
33.     }
34. }
35. }
36. }
37. }

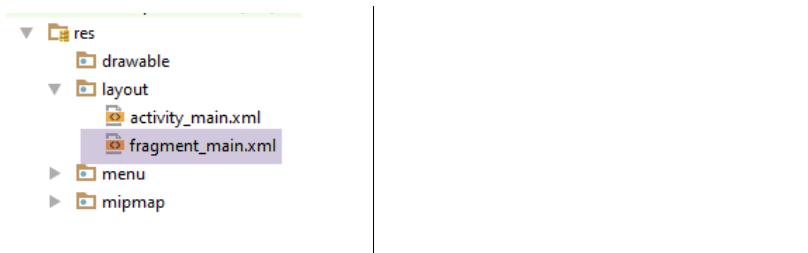
```

- les fragments affichés par une application dépendent de celle-ci. Le gestionnaire de fragments est défini par le développeur ;
- ligne 5 : le gestionnaire de fragments étend la classe Android [android.support.v4.app.FragmentPagerAdapter]. Le constructeur nous est imposé. Nous devons définir au moins les deux méthodes suivantes :
 - int getCount()** : rend le nombre de fragments à gérer ;
 - Fragment getItem(i)** : rend le fragment n° i ;

La méthode **CharSequence getPageTitle(i)** qui rend le titre du fragment n° i est facultative. Parce que le gestionnaire d'onglets a été associé au gestionnaire de fragments, le titre de l'onglet n° i sera le titre du fragment n° i. Ainsi, les titres des lignes 27-33 vont-ils être les titres des onglets ;

- lignes 18-21 : **getCount** rend le nombre de fragments gérés, ici trois ;
- lignes 11-15 : **getItem(i)** rend le fragment n° i. Ici tous les fragments seront identiques de type [PlaceholderFragment] ;
- lignes 24-35 : **getPageTitle(int i)** rend le titre du fragment n° i ;

1.7.4.2 Les fragments affichés



Les fragments de l'activité ont ici tous le même type et sont tous associés à la vue XML [fragment_main] suivante :

```

1. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.                 xmlns:tools="http://schemas.android.com/tools"
3.                 android:layout_width="match_parent"
4.                 android:layout_height="match_parent"
5.                 android:paddingLeft="@dimen/activity_horizontal_margin"
6.                 android:paddingRight="@dimen/activity_horizontal_margin"
7.                 android:paddingTop="@dimen/activity_vertical_margin"
8.                 android:paddingBottom="@dimen/activity_vertical_margin"
9.                 tools:context="exemples.android.MainActivity$PlaceholderFragment">
10.
11.    <TextView
12.        android:id="@+id/section_label"
13.        android:layout_width="wrap_content"
14.        android:layout_height="wrap_content"/>
15.
16. </RelativeLayout>

```

- lignes 1-16 : un *layout* de type [RelativeLayout] ;
- lignes 11-14 : l'unique composant de la vue (fragment) : un [TextView] identifié par [section_label] ;

Dans [MainActivity], les fragments gérés sont du type [PlaceholderFragment] suivant :

```

1. // un fragment
2. public static class PlaceholderFragment extends Fragment {
3.     // un texte affiché dans le fragment
4.     private static final String ARG_SECTION_NUMBER = "section_number";
5.
6.     public PlaceholderFragment() {
7.     }
8.
9.     // rend un fragment avec une information : le n° du fragment passé en paramètre
10.    public static PlaceholderFragment newInstance(int sectionNumber) {
11.        // fragment
12.        PlaceholderFragment fragment = new PlaceholderFragment();
13.        // info embarquée

```

```

14.     Bundle args = new Bundle();
15.     args.putInt(ARG_SECTION_NUMBER, sectionNumber);
16.     fragment.setArguments(args);
17.     // résultat
18.     return fragment;
19. }
20.
21. @Override
22. public View onCreateView(LayoutInflater inflater, ViewGroup container,
23.                         Bundle savedInstanceState) {
24.     // vue [fragment_main] est instanciée
25.     View rootView = inflater.inflate(R.layout.fragment_main, container, false);
26.     // le [TextView] est retrouvé
27.     TextView textView = (TextView) rootView.findViewById(R.id.section_label);
28.     // son contenu est modifié
29.     textView.setText(getString(R.string.section_format, getArguments().getInt(ARG_SECTION_NUMBER)));
30.     // on retourne la vue
31.     return rootView;
32. }
33. }
```

- ligne 2 : la classe [PlaceholderFragment] étend la classe Android [Fragment]. C'est en général toujours le cas ;
- ligne 2 : la classe [PlaceholderFragment] est statique. Sa méthode [newInstance] (ligne 10) permet d'obtenir des instances de type [PlaceholderFragment] ;
- lignes 10-19 : la méthode [newInstance] crée et retourne un objet de type [PlaceholderFragment] ;
- lignes 14-16 : le fragment est créé avec un argument ;

Un fragment doit définir la méthode [onCreateView] de la ligne 22. Cette méthode doit rendre la vue associée au fragment.

- ligne 25 : la vue [fragment_main.xml] est associée au fragment ;
- ligne 27 : cette vue contient un composant [TextView] dont on récupère la référence via son identifiant ;
- ligne 29 : on affiche un texte dans le [TextView] ;
 - [getString] est une méthode de la classe parent [AppCompatActivity] ;
 - le 1er argument est un n° de composant. [R.string.section_format] désigne le n° du composant identifié par [section_format] dans le fichier [res / values / strings.xml] (ligne 4 ci-dessous) :

```

a) <resources>
b)   <string name="app_name">Exemple-06</string>
c)   <string name="action_settings">Settings</string>
d)   <string name="section_format">Hello World from section: %1$d</string>
e) </resources>
```

- ligne (d) ci-dessus %1\$d indique que l'argument n° 1 (%1) doit être formaté comme un nombre entier (\$d) ;
- le second argument de [getString] est la valeur à donner à l'argument \$1 de la ligne (d) ci-dessus ;
- [getArguments] donne la référence du bundle des arguments du fragment. Il faut se rappeler ici que chaque argument a été créé avec le bundle suivant (lignes f-h) :

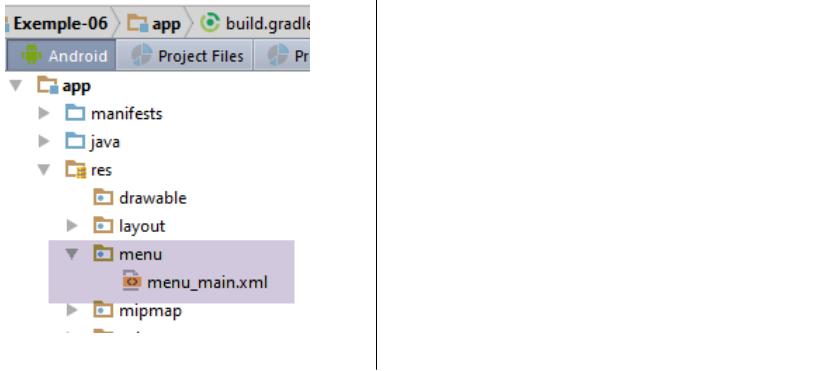
```

a)    // rend un fragment avec une information : le n° du fragment passé en paramètre
b)    public static PlaceholderFragment newInstance(int sectionNumber) {
c)        // fragment
d)        PlaceholderFragment fragment = new PlaceholderFragment();
e)        // info embarquée
f)        Bundle args = new Bundle();
g)        args.putInt(ARG_SECTION_NUMBER, sectionNumber);
h)        fragment.setArguments(args);
i)        // résultat
j)        return fragment;
k)    }
```

- getArguments().getInt(ARG_SECTION_NUMBER) va donc rendre la valeur [sectionNumber] des lignes (g) et (b) ci-dessus ;
- ligne 31 : on rend la vue ainsi créée ;

1.7.4.3 Gestion du menu

Dans l'application générée, il y a un menu :



Le contenu du fichier [menu_main.xml] est le suivant :

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.       xmlns:app="http://schemas.android.com/apk/res-auto"
3.       xmlns:tools="http://schemas.android.com/tools"
4.       tools:context="exemples.android.MainActivity">
5.     <item android:id="@+id/action_settings"
6.           android:title="@string/action_settings"
7.           android:orderInCategory="100"
8.           app:showAsAction="never"/>
9.   </menu>

```

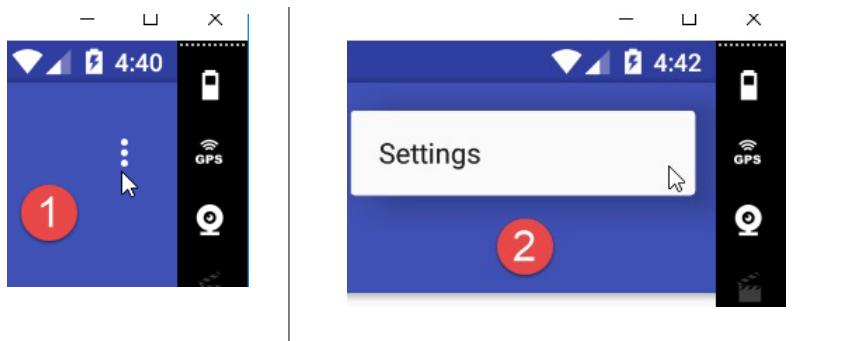
- lignes 1-9 : le menu ;
- lignes 5-8 : un élément du menu identifié par [action_settings] (ligne 5) ;
- ligne 6 : le label de l'option de menu. Elle est trouvée dans le fichier [res / values / strings.xml] (ligne (c) ci-dessous) :

```

a) <resources>
b)   <string name="app_name">Exemple-06</string>
c)   <string name="action_settings">Settings</string>
d)   <string name="section_format">Hello World from section: %1$d</string>
e) </resources>

```

Le code précédent correspond au visuel suivant (le menu est en haut à droite de la fenêtre d'exécution Android) :



Ce menu est géré de la façon suivante dans l'activité [MainActivity] :

```

1. @Override
2. public boolean onCreateOptionsMenu(Menu menu) {
3.     // Inflate the menu; this adds items to the action bar if it is present.
4.     getMenuInflater().inflate(R.menu.menu_main, menu);
5.     return true;
6. }
7.
8. @Override
9. public boolean onOptionsItemSelected(MenuItem item) {
10.    // Handle action bar item clicks here. The action bar will
11.    // automatically handle clicks on the Home/Up button, so long
12.    // as you specify a parent activity in AndroidManifest.xml.
13.    int id = item.getItemId();
14.
15.    //noinspection SimplifiableIfStatement
16.    if (id == R.id.action_settings) {
17.        return true;
18.    }

```

```

19.     return super.onOptionsItemSelected(item);
20. }

```

- lignes 1-6 : cette méthode est appelée lorsque le système est prêt à créer le menu de l'application. Le paramètre d'entrée [Menu menu] est un menu vide n'ayant pas encore d'options ;
- ligne 4 : le fichier [res / menu / menu_main.xml] est exploité. L'objet [Menu menu] passé en paramètre se voit attribuer les options de menu définis dans ce fichier ;
- ligne 5 : on indique que la création du menu a été faite ;
- lignes 8-21 : la méthode [onOptionsItemSelected] est exécutée dès qu'une option du menu est cliquée ;
- ligne 13 : la référence de l'option de menu cliquée ;
- lignes 16-18 : si l'option cliquée est l'option d'identifiant [action_settings], rien n'est fait et on indique que l'événement a été traité (ligne 17) ;
- ligne 20 : on passe l'événement à la classe parent ;

Pour mieux voir ce qui se passe avec ce menu, nous ajoutons des logs dans le code précédent :

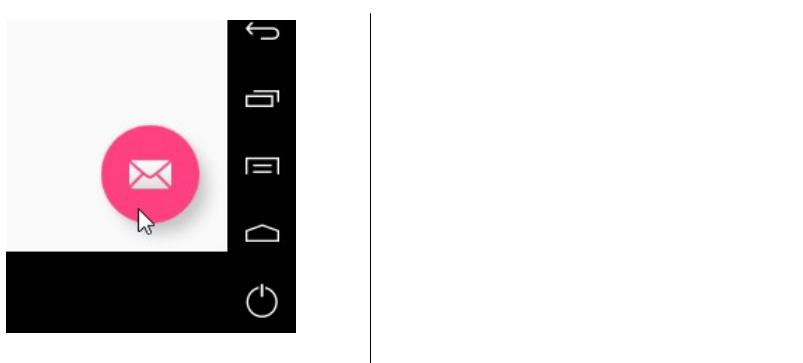
```

1.  @Override
2.  public boolean onCreateOptionsMenu(Menu menu) {
3.      Log.d("menu", "création menu en cours");
4.      // Inflate the menu; this adds items to the action bar if it is present.
5.      getMenuInflater().inflate(R.menu.menu_main, menu);
6.      return true;
7.  }
8.
9.  @Override
10. public boolean onOptionsItemSelected(MenuItem item) {
11.     Log.d("menu", "onOptionsItemSelected");
12.     // Handle action bar item clicks here. The action bar will
13.     // automatically handle clicks on the Home/Up button, so long
14.     // as you specify a parent activity in AndroidManifest.xml.
15.     int id = item.getItemId();
16.
17.     //noinspection SimplifiableIfStatement
18.     if (id == R.id.action_settings) {
19.         Log.d("menu", "action_settings selected");
20.         return true;
21.     }
22.     // parent
23.     return super.onOptionsItemSelected(item);
24. }

```

1.7.4.4 Le bouton flottant

La vue générée a un bouton flottant :



Ce composant est défini dans la vue principale [activity-main.xml] :

```

1.  <android.support.design.widget.FloatingActionButton
2.      android:id="@+id/fab"
3.      android:layout_width="wrap_content"
4.      android:layout_height="wrap_content"
5.      android:layout_gravity="end|bottom"
6.      android:layout_margin="@dimen/fab_margin"
7.      android:src="@android:drawable/ic_dialog_email"/>

```

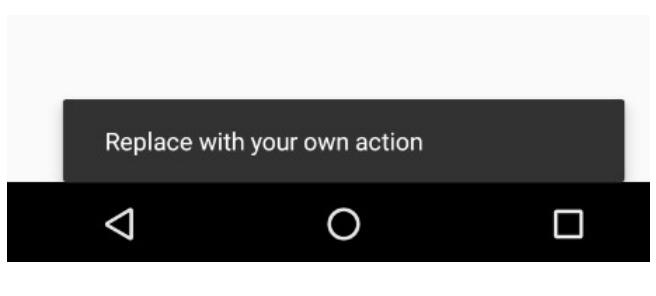
La ligne 7 référence une image fournie par le support Android, celle d'une enveloppe.

Ce composant est géré dans la classe [MainActivity] de la façon suivante :

```
1.      // bouton flottant
2.      FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
3.      fab.setOnClickListener(new View.OnClickListener() {
4.          @Override
5.          public void onClick(View view) {
6.              Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
7.                  .setAction("Action", null).show();
8.          }
9.     });
```

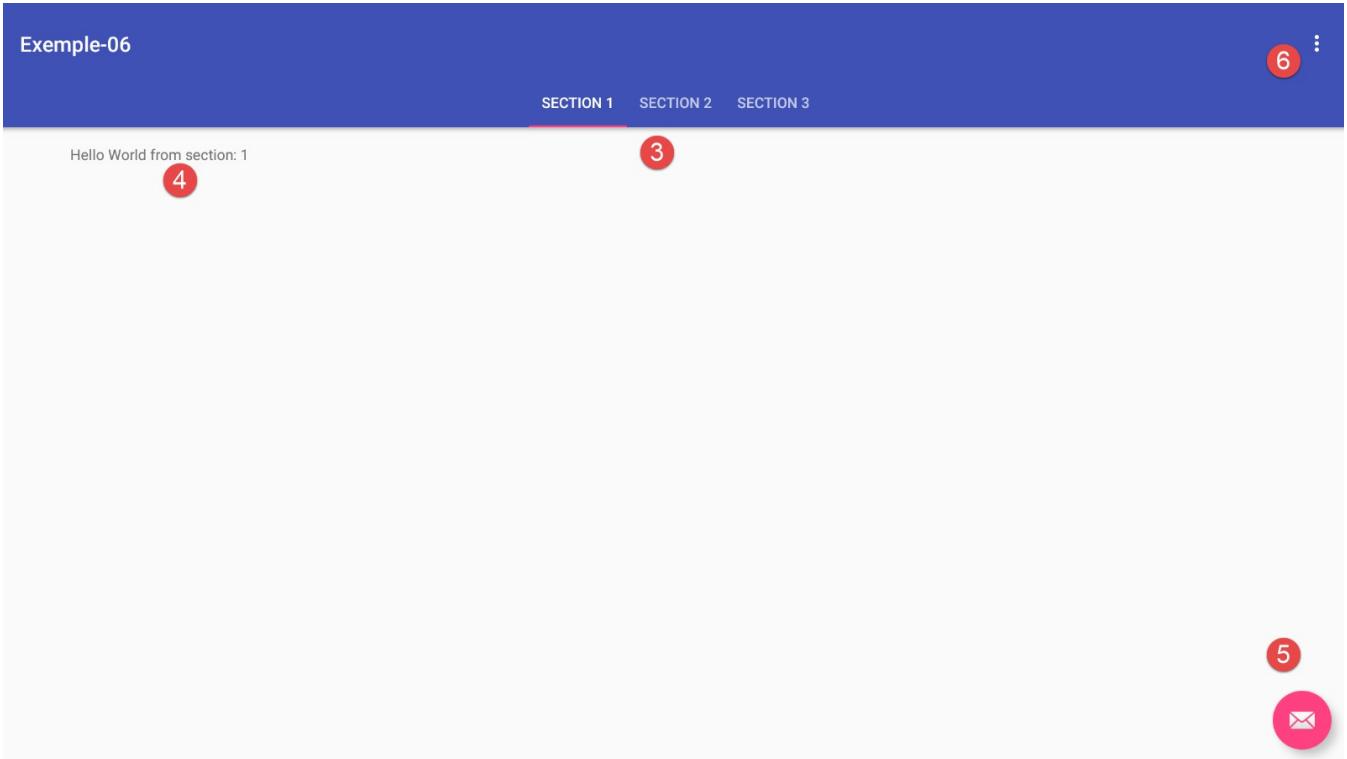
- ligne 2 : on récupère la référence du bouton flottant dans la vue associée à l'activité (activity_main) ;
- lignes 3-9 : on lui associe un gestionnaire pour gérer le clic sur lui ;
- ligne 6 : la classe [Snackbar] permet d'afficher des messages éphémères sur la vue avec sa méthode [Snackbar.make]. Le 1er argument est une vue à partir de laquelle [Snackbar] va chercher une vue parente dans laquelle afficher le message. Ici [view] est la vue de l'enveloppe cliquée (ligne 5). La vue parente qui sera trouvée sera la vue [activity_main]. Le second argument est le message à afficher. Le troisième argument est la durée d'affichage (SHORT ou LONG) ;
- ligne 7 : on peut cliquer sur le message affiché et déclencher ainsi une action. Ici aucune action n'est associée au clic sur le message. Finalement, la méthode [show] affiche le message ;

Le clic sur le bouton flottant donne le résultat visuel suivant :



1.7.5 Exécution du projet

Maintenant que nous avons expliqué les détails du code généré, nous pouvons mieux comprendre son exécution :



Lorsqu'on clique sur l'onglet n° i, le fragment n° i est affiché dans le conteneur de vues. Cela se voit au texte affiché en [4]. On peut remarquer aussi qu'on peut passer d'un onglet à l'autre en tirant la vue à droite ou à gauche avec la souris (swipe). Nous verrons qu'on peut contrôler ce comportement.

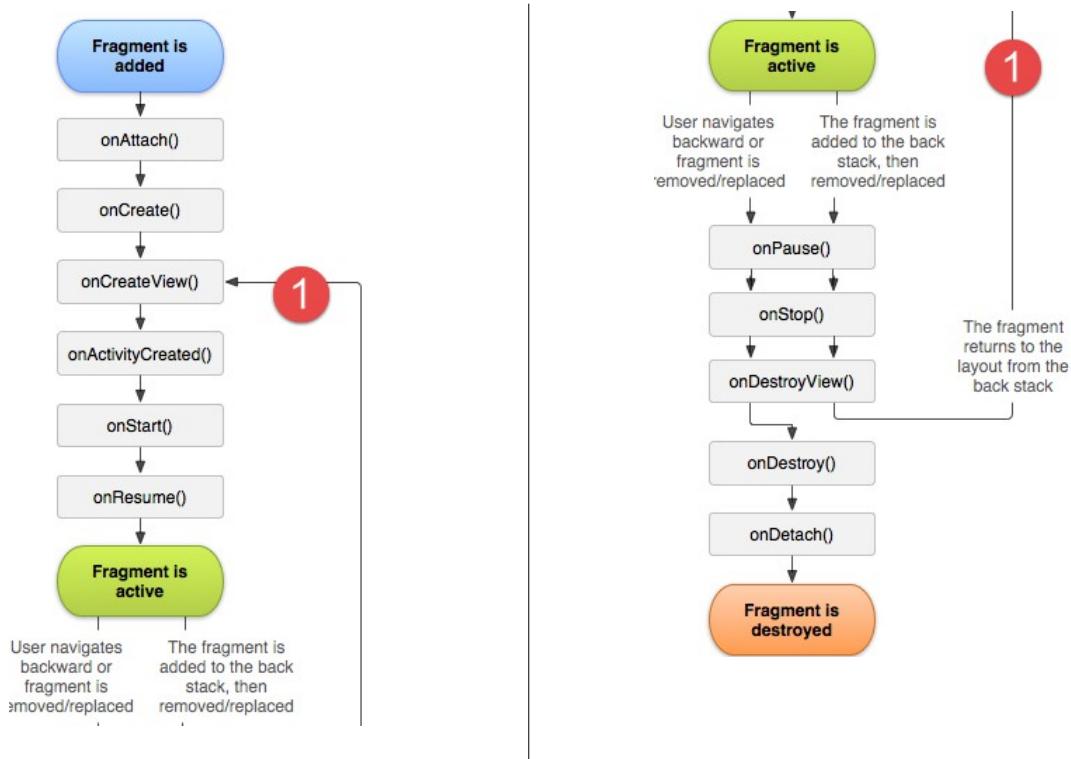
Lorsqu'on clique sur l'option de menu en [6], on a les logs suivants :

```

Android Monitor
Genymotion Custom Tablet - 6.0.0 - API 23 - 2560x1600 Android 6.0, API 23 | exemples.android (14751)
Logcat Monitors →
1 05-27 04:57:22.674 14751-14792/exemples.android W/OpenGLRenderer: Failed to set EGL_SWAP_BE
05-27 04:57:22.685 14751-14751/exemples.android W/ViewRootImpl: Dropping event due to no wi
05-27 04:57:22.686 14751-14751/exemples.android W/ViewRootImpl: Dropping event due to no wi
05-27 04:57:23.950 14751-14751/exemples.android D/menu: onOptionsItemSelected
05-27 04:57:23.951 14751-14751/exemples.android D/menu: action_settings selected
2 05-27 04:57:23.976 14751-14751/exemples.android W/ViewRootImpl: Cancelling event due to no i
05-27 04:57:23.976 14751-14751/exemples.android W/ViewRootImpl: Cancelling event due to no i
05-27 04:57:23.976 14751-14751/exemples.android W/ViewRootImpl: Cancelling event due to no i
05-27 04:57:23.976 14751-14751/exemples.android W/ViewRootImpl: Cancelling event due to no i
05-27 04:57:23.976 14751-14751/exemples.android W/ViewRootImpl: Cancelling event due to no i
05-27 04:57:23.976 14751-14751/exemples.android W/ViewRootImpl: Cancelling event due to no i
3 > Run T O L Android Monitor Terminal Messages

```

1.7.6 Cycle de vie des fragments



- en [1], on voit que la méthode [onCreateView] et les suivantes sont exécutées lors du 1er affichage du fragment et à chaque fois que l'activité doit le réafficher ;

Pour suivre le cycle de vie de l'activité et des fragments, nous ajoutons les logs suivants dans le code de [MainActivity] :

```

1. // constructeur
2. public MainActivity(){
3.     Log.d("MainActivity", "constructor");
4. }
5.
6. @Override
7. protected void onCreate(Bundle savedInstanceState) {
8.     Log.d("MainActivity", "onCreate");
9.     // parent
10.    super.onCreate(savedInstanceState);
11. ...
12. }
13.
14. // un fragment
15. public static class PlaceholderFragment extends Fragment {
16.     // un texte affiché dans le fragment
17.     private static final String ARG_SECTION_NUMBER = "section_number";
18.
19.     public PlaceholderFragment() {
20.         Log.d("PlaceholderFragment", "constructor");
21.     }
22.
23.     // rend un fragment avec une information : le n° du fragment passé en paramètre
24.     public static PlaceholderFragment newInstance(int sectionNumber) {
25.         Log.d("PlaceholderFragment", String.format("newInstance %s", sectionNumber));
26.         // fragment
27.         PlaceholderFragment fragment = new PlaceholderFragment();
28.         ...
29.     }
30.
31.     @Override
32.     public View onCreateView(LayoutInflater inflater, ViewGroup container,
33.                             Bundle savedInstanceState) {
34.         Log.d("PlaceholderFragment", String.format("newInstance %s", getArguments().getInt(ARG_SECTION_NUMBER)));
35.         ...
36.     }
37. }
38.
39. }
```

Nous exécutons de nouveau le projet. Les premiers logs sont les suivants :

```

1. 05-28 10:44:32.622 29371/exemples.android D/MainActivity: constructor
2. 05-28 10:44:32.626 29371/exemples.android D/MainActivity: onCreate
3. 05-28 10:44:32.759 29371/exemples.android D/PlaceholderFragment: newInstance 1
4. 05-28 10:44:32.759 29371/exemples.android D/PlaceholderFragment: constructor
5. 05-28 10:44:32.759 29371/exemples.android D/PlaceholderFragment: newInstance 2
6. 05-28 10:44:32.759 29371/exemples.android D/PlaceholderFragment: constructor
7. 05-28 10:44:32.759 29371/exemples.android D/PlaceholderFragment: onCreateView 2
8. 05-28 10:44:32.760 29371/exemples.android D/PlaceholderFragment: onCreateView 1
9. 05-28 10:44:33.349 29371/exemples.android D/menu: création menu en cours

```

- ligne 1 : création de l'activité ;
- ligne 2 : exécution de sa méthode [onCreate] ;
- lignes 3-4 : instanciation du fragment n° 1 ;
- lignes 5-6 : instanciation du fragment n° 2 ;
- ligne 7 : initialisation du fragment n° 2 ;
- ligne 8 : initialisation du fragment n° 1 ;
- ligne 9 : création du menu de l'activité ;

Il faut se rappeler ici le code qui préside à la création des fragments :

```

1. // le gestionnaire de fragments
2. // c'est à lui qu'on demande les fragments à afficher dans la vue principale
3. // doit définir les méthodes [getItem] et [getCount] - les autres sont facultatives
4. public class SectionsPagerAdapter extends FragmentPagerAdapter {
5.
6.     public SectionsPagerAdapter(FragmentManager fm) {
7.         super(fm);
8.     }
9.
10.    // fragment n° position
11.    @Override
12.    public Fragment getItem(int position) {
13.        // on instancie un fragment [PlaceHolder] et on le rend
14.        return PlaceholderFragment.newInstance(position + 1);
15.    }
16. ...

```

- lignes 11-15 : un fragment est instancié par [newInstance] à chaque fois que le conteneur de fragments en demande un ;

Les logs ci-dessus montrent que les deux premiers fragments ont été instanciés et initialisés.

Maintenant, cliquons sur l'onglet n° 2. Les nouveaux logs sont les suivants :

```

1. 05-28 10:47:15.566 29371/exemples.android D/PlaceholderFragment: newInstance 3
2. 05-28 10:47:15.566 29371/exemples.android D/PlaceholderFragment: constructor
3. 05-28 10:47:15.566 29371/exemples.android D/PlaceholderFragment: onCreateView 3

```

- lignes 1-3 : le fragment n° 3 est instancié et initialisé. On rappelle que c'est le fragment n° 2 qui est affiché ;

Maintenant, cliquons sur l'onglet n° 3. Là il n'y a aucun log. Probablement parce que le fragment n° 3 à afficher avait déjà été instancié. Maintenant, revenons à l'onglet n° 1. Les logs sont alors les suivants :

```
1. 05-28 10:48:26.630 29371/exemples.android D/PlaceholderFragment: onCreateView 1
```

Le fragment n° 1 n'est pas instancié de nouveau mais sa méthode [onCreateView] est de nouveau exécutée. Ce comportement se reproduit pour les deux autres fragments.

De ces logs, on retiendra que :

- l'activité a été instanciée puis initialisée une fois ;
- que chaque fragment a été instancié une fois ;
- que la méthode [onCreateView] de chaque fragment a été exécutée plusieurs fois ;

Ce qu'il faut savoir et ce que confirment les logs est que par défaut, lorsqu'un fragment n° i est affiché, les fragments i-1 et i+1 sont instanciés, s'ils ne le sont pas déjà. C'est ce qui explique par exemple qu'au démarrage, alors qu'il faut afficher le fragment n° 1, ce sont les fragments 1 et 2 qui ont été instanciés et initialisés. Ce que montrent également les logs est que la méthode [getItem(i)] n'est appelée qu'une fois, même si le fragment n° i est lui affiché plusieurs fois. Ainsi il semble que le conteneur de fragments [ViewPager] qui doit afficher le fragment n° i demande celui-ci une fois au gestionnaire de fragments [SectionsPagerAdapter]. Ensuite il ne le redemande plus et continue à utiliser celui qu'il a obtenu.

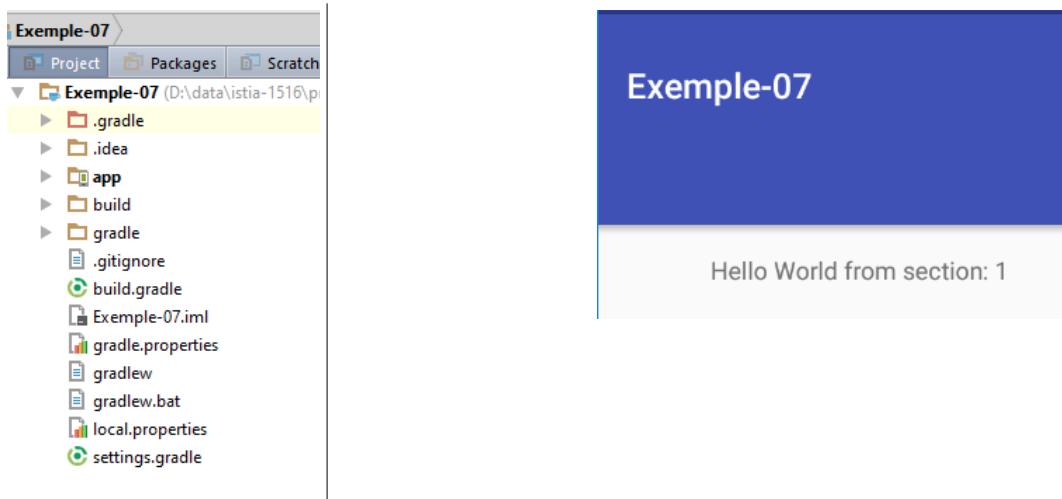
Enfin, les logs donnent des indications sur la méthode [onCreateView] des fragments :

- au démarrage, les fragments 1 et 2 ont été instanciés et leur méthode [onCreateView] exécutée ;
- lorsqu'on passe du fragment 1 au fragment 2, la méthode [onCreateView] du fragment 2 n'est pas réexécutée. On ne peut donc s'en servir pour mettre à jour le fragment 2. Or l'utilisateur peut, avec le fragment 1, avoir fait une opération dont le résultat devrait être affiché par le fragment 2. On voit que la méthode [onCreateView] ne pourra pas être utilisée pour mettre à jour le fragment 2. Il faudra trouver une autre solution ;

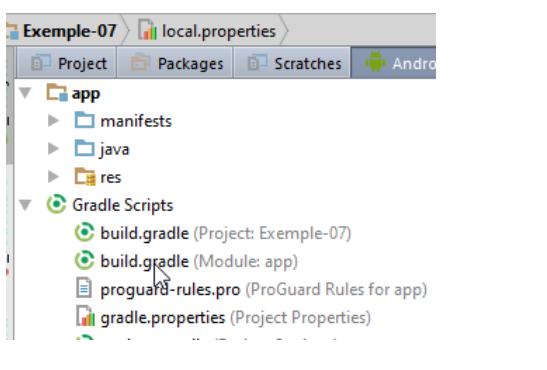
1.8 Exemple-07 : Exemple-06 réécrit avec la bibliothèque [AA]

1.8.1 Crédit du projet

Nous allons dupliquer le projet [Exemple-06] dans [Exemple-07] pour introduire dans ce dernier les annotations Android. Pour cela, suivez la procédure du paragraphe 1.4, page 36. Nous obtenons le résultat suivant :



1.8.2 Configuration Gradle



Nous faisons évoluer le fichier [build.gradle] de la façon suivante :

```
1. buildscript {
2.     repositories {
3.         mavenCentral()
4.     }
5.     dependencies {
6.         // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
7.         classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
8.     }
9. }
10.
11. apply plugin: 'com.android.application'
12. apply plugin: 'android-apt'
13.
14. android {
15.     compileSdkVersion 23
16.     buildToolsVersion "23.0.3"
17.     defaultConfig {
18.         applicationId "exemples.android"
19.         minSdkVersion 15
20.         targetSdkVersion 23
21.         versionCode 1
```

```

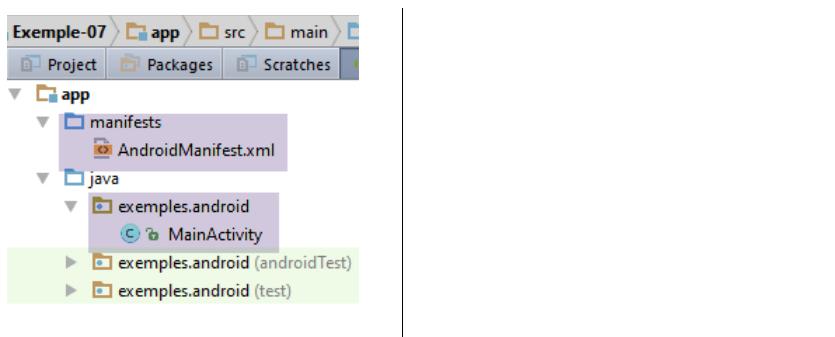
22.     versionName "1.0"
23.   }
24.   buildTypes {
25.     release {
26.       minifyEnabled false
27.       proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
28.     }
29.   }
30. }
31.
32. def AAVersion = '4.0.0'
33. dependencies {
34.   apt "org.androidannotations:androidannotations:$AAVersion"
35.   compile "org.androidannotations:androidannotations-api:$AAVersion"
36.   compile 'com.android.support:appcompat-v7:23.4.0'
37.   compile 'com.android.support:design:23.4.0'
38.   compile fileTree(dir: 'libs', include: ['*.jar'])
39.   testCompile 'junit:junit:4.12'
40. }

```

Nous avons ajouté la configuration nécessaire à l'utilisation de la bibliothèque [Android Annotations] (cf paragraphe 1.4, page 36).

1.8.3 Ajout des premières annotations AA

Nous allons créer des annotations AA dans [MainActivity] :



La classe [MainActivity] évolue de la façon suivante :

```

1. @EActivity(R.layout.activity_main)
2. public class MainActivity extends AppCompatActivity {
3.
4.     // le gestionnaire de fragments
5.     private SectionsPagerAdapter mSectionsPagerAdapter;
6.
7.     // le conteneur de fragments
8.     @ViewById(R.id.container)
9.     protected MyPagerAdapter mViewPagerAdapter;
10.    // le gestionnaire d'onglets
11.    @ViewById(R.id.tabs)
12.    protected TabLayout tabLayout;
13.    // le bouton flottant
14.    @ViewById(R.id.fab)
15.    protected FloatingActionButton fab;
16.
17.
18.    // constructeur
19.    public MainActivity() {
20.        Log.d("MainActivity", "constructor");
21.    }
22.
23.    @AfterViews
24.    protected void afterViews() {
25.        Log.d("MainActivity", "afterViews");
26.
27.        // barre d'outils
28.        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
29.        setSupportActionBar(toolbar);
30.
31.        // le gestionnaire de fragments
32.        mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
33.
34.        // le conteneur de fragments est associé au gestionnaire de fragments
35.        // c-à-d que le fragment n° i du conteneur de fragments est le fragment n° i délivré par le gestionnaire de fragments
36.        mViewPagerAdapter.setAdapter(mSectionsPagerAdapter);
37.

```

```

38.     // la barre d'onglets est également associée au conteneur de fragments
39.     // c-à-d que l'onglet n° i affiche le fragment n° i du conteneur
40.     tabLayout.setupWithViewPager(mViewPager);
41.
42.     // bouton flottant
43.     fab.setOnClickListener(new View.OnClickListener() {
44.         @Override
45.         public void onClick(View view) {
46.             Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
47.                 .setAction("Action", null).show();
48.         }
49.     });
50. }

```

- ligne 1 : l'annotation `[@EActivity]` fait de `[MainActivity]` une classe gérée par AA. Son paramètre `[R.layout.activity_main]` est l'identifiant de la vue `[activity_main.xml]` associée à l'activité ;
- lignes 11-12 : le composant identifié par `[R.id.tabs]` est injecté dans le champ `[tabLayout]`. C'est le gestionnaire d'onglets ;
- lignes 14-15 : le composant identifié par `[R.id.fab]` est injecté dans le champ `[fab]`. C'est le bouton flottant ;
- lignes 23-50 : le code qui était auparavant dans la méthode `[onCreate]` migre dans une méthode de nom quelconque mais annotée par `[@AfterViews]` (ligne 23). Dans la méthode ainsi annotée, on est assuré que tous les composants de l'interface visuelle annotés par `[@ViewById]` ont été initialisés ;
- on a mis par ailleurs des logs pour voir le cycle de vie de l'activité ;

On se rappelle que l'annotation `[@EActivity]` va générer une classe `[MainActivity_]` qui sera la véritable activité du projet. Il faut donc modifier le fichier `[AndroidManifest.xml]` de la façon suivante :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.           package="exemples.android">
4.
5.     <application
6.         android:allowBackup="true"
7.         android:icon="@mipmap/ic_launcher"
8.         android:label="@string/app_name"
9.         android:supportsRtl="true"
10.        android:theme="@style/AppTheme">
11.        <activity
12.            android:name=".MainActivity_"
13.            android:label="@string/app_name"
14.            android:theme="@style/AppTheme.NoActionBar">
15.                <intent-filter>
16.                    <action android:name="android.intent.action.MAIN"/>
17.
18.                    <category android:name="android.intent.category.LAUNCHER"/>
19.                </intent-filter>
20.            </activity>
21.        </application>
22.
23. </manifest>

```

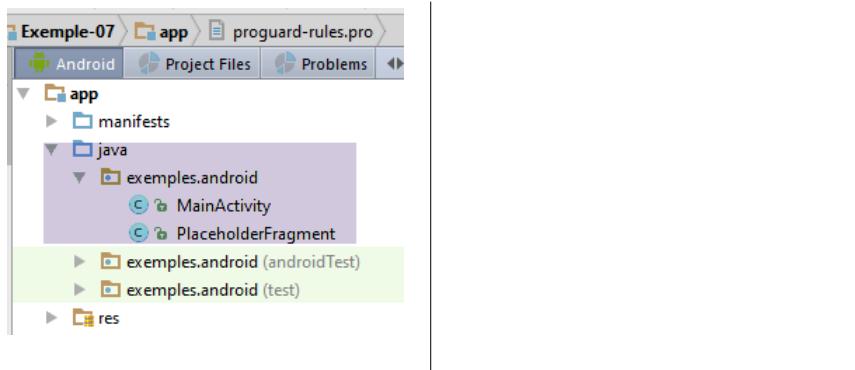
- ligne 12 : la nouvelle activité.

A ce point, exécutez de nouveau le projet et vérifiez que vous obtenez toujours l'interface avec onglets.

1.8.4 Réécriture des fragments

Nous allons revoir la gestion des fragments du projet. Pour l'instant la classe `[PlaceholderFragment]` est une classe interne statique de l'activité `[MainActivity]`. Nous allons revenir à un cas d'utilisation plus usuel, celui où les fragments sont définis dans des classes externes. Par ailleurs, nous introduisons les annotations AA pour les fragments.

Le projet `[Exemple-07]` évolue de la façon suivante :



Ci-dessus, on voit apparaître la classe [PlaceholderFragment] qui a été externalisée en-dehors de la classe [MainActivity]. Elle est réécrite de la façon suivante :

```

1. package exemples.android;
2.
3. import android.os.Bundle;
4. import android.support.v4.app.Fragment;
5. import android.util.Log;
6. import android.view.LayoutInflater;
7. import android.view.View;
8. import android.view.ViewGroup;
9. import android.widget.TextView;
10. import org.androidannotations.annotations.AfterViews;
11. import org.androidannotations.annotations.EFragment;
12. import org.androidannotations.annotations.ViewById;
13.
14. // un fragment est une vue affichée par un conteneur de fragments
15. @EFragment(R.layout.fragment_main)
16. public class PlaceholderFragment extends Fragment {
17.
18.     // composant de l'interface visuelle
19.     @ViewById(R.id.section_label)
20.     protected TextView textViewInfo;
21.
22.     // n° de fragment
23.     private static final String ARG_SECTION_NUMBER = "section_number";
24.
25.     // constructeur
26.     public PlaceholderFragment() {
27.         Log.d("PlaceholderFragment", "constructor");
28.     }
29.
30.     @AfterViews
31.     protected void afterViews() {
32.         Log.d("PlaceholderFragment", String.format("afterViews %s", getArguments().getInt(ARG_SECTION_NUMBER)));
33.     }
34.
35.
36.     @Override
37.     public View onCreateView(LayoutInflater inflater, ViewGroup container,
38.                             Bundle savedInstanceState) {
39.         Log.d("PlaceholderFragment", String.format("onCreateView %s", getArguments().getInt(ARG_SECTION_NUMBER)));
40.         return super.onCreateView(inflater, container, savedInstanceState);
41.     }
42.
43.     @Override
44.     public void onResume() {
45.         Log.d("PlaceholderFragment", String.format("onResume %s", getArguments().getInt(ARG_SECTION_NUMBER)));
46.         // parent
47.         super.onResume();
48.         // affichage
49.         if (textViewInfo != null) {
50.             Log.d("PlaceholderFragment", String.format("onResume setText %s", getArguments().getInt(ARG_SECTION_NUMBER)));
51.             textViewInfo.setText(getString(R.string.section_format, getArguments().getInt(ARG_SECTION_NUMBER)));
52.         }
53.     }
54. }
```

- ligne 15 : le fragment est annoté avec l'annotation [@EFragment] dont le paramètre est l'identifiant de la vue XML associée au fragment, ici la vue [fragment_main.xml] ;
- lignes 19-20 : injectent dans le champ [textViewInfo] la référence du composant de [fragment_main.xml] identifié par [R.id.section_label] qui est un type [TextView] (ligne 1) ci-dessous) :

a) <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

```

b)           xmlns:tools="http://schemas.android.com/tools"
c)           android:layout_width="match_parent"
d)           android:layout_height="match_parent"
e)           android:paddingLeft="@dimen/activity_horizontal_margin"
f)           android:paddingRight="@dimen/activity_horizontal_margin"
g)           android:paddingTop="@dimen/activity_vertical_margin"
h)           android:paddingBottom="@dimen/activity_vertical_margin"
i)           tools:context="exemples.android.MainActivity$PlaceholderFragment">
j)
k)       <TextView
l)           android:id="@+id/section_label"
m)           android:layout_width="wrap_content"
n)           android:layout_height="wrap_content"/>
o)
p)   </RelativeLayout>

```

- lignes 42-52 : la méthode [onResume] est exécutée avant l'affichage de la vue associée au fragment. On peut l'utiliser pour mettre à jour l'interface visuelle qui va être affichée ;
- ligne 47 : on doit appeler la méthode de même nom de la classe parent ;
- ligne 49 : il y a une difficulté à savoir si la méthode [onResume] peut être ou non exécutée avant l'initialisation du champ de la ligne 20. Les logs mis pour suivre le cycle de vie du fragment nous le diront. Pour l'instant et par précaution, on fait un test de nullité ;
- ligne 51 : on met à jour l'information du champ [textViewInfo] avec l'argument entier passé au fragment lors de sa création ;

La classe [MainActivity] perd sa classe interne [PlaceholderFragment] et voit son gestionnaire de fragments évoluer de la façon suivante :

```

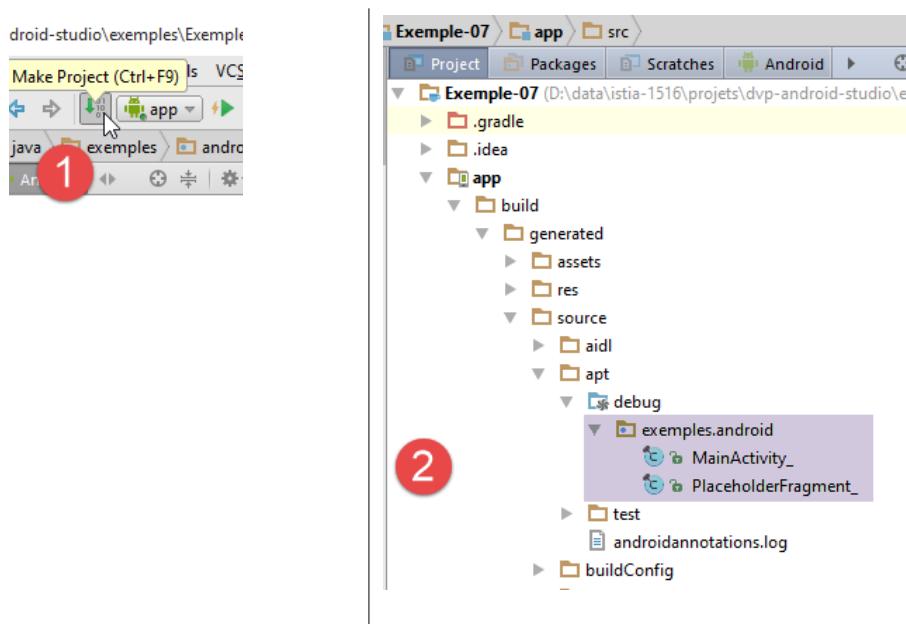
1.  public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.      // les fragments
4.      private Fragment[] fragments;
5.      // nombre de fragments
6.      private static final int FRAGMENTS_COUNT = 3;
7.      // n° de fragment
8.      private static final String ARG_SECTION_NUMBER = "section_number";
9.
10.     // constructeur
11.     public SectionsPagerAdapter(FragmentManager fm) {
12.         // parent
13.         super(fm);
14.         // initialisation du tableau des fragments
15.         fragments = new Fragment[FRAGMENTS_COUNT];
16.         for (int i = 0; i < fragments.length; i++) {
17.             // on crée un fragment
18.             fragments[i] = new PlaceholderFragment_();
19.             // on peut passer des arguments au fragment
20.             Bundle args = new Bundle();
21.             args.putInt(ARG_SECTION_NUMBER, i + 1);
22.             fragments[i].setArguments(args);
23.         }
24.     }
25.
26.     // fragment n° position
27.     @Override
28.     public Fragment getItem(int position) {
29.         Log.d("MainActivity", String.format("getItem[%s]", position));
30.         return fragments[position];
31.     }
32.
33.     // rend le nombre de fragments générés
34.     @Override
35.     public int getCount() {
36.         return fragments.length;
37.     }
38.
39.     // facultatif - donne un titre aux fragments générés
40.     @Override
41.     public CharSequence getPageTitle(int position) {
42.         return String.format("Onglet n° %s", (position + 1));
43.     }
44. }

```

- ligne 4 : les fragments sont mis dans un tableau ;
- lignes 16-23 : l'initialisation du tableau des fragments se fait dans le constructeur. Ils sont de type [PlaceholderFragment_] (ligne 18) et non [PlaceholderFragment]. La classe [PlaceholderFragment] a en effet été annotée par une annotation AA et va donner naissance à une classe [PlaceholderFragment_] dérivée de [PlaceholderFragment] et c'est cette classe que l'activité doit utiliser. Chaque fragment créé se voit passer un argument entier qui sera affiché par le fragment ;

- lignes 42-45 : on a changé les titres des fragments. Comme ceux-ci sont également les titres des onglets, on devrait voir un changement dans la barre des onglets ;

Compilons [Make] [1] ce projet :



- en [2], on voit que les classes générées par la bibliothèque AA le sont dans le dossier [app / build / generated / source / apt / debug] (il faut être dans la perspective [Project] pour voir [2]) ;

Exécutez le projet [Exemple-07] et vérifiez qu'il fonctionne toujours.

1.8.5 Examen des logs

Lorsqu'on lance l'application, les logs sont les suivants :

```

1. 05-28 13:54:54.801 8809/8809/exemples.android D/MainActivity: constructor
2. 05-28 13:54:54.901 8809/8809/exemples.android D/MainActivity: afterViews
3. 05-28 13:54:54.919 8809/8809/exemples.android D/PlaceholderFragment: constructor
4. 05-28 13:54:54.919 8809/8809/exemples.android D/PlaceholderFragment: constructor
5. 05-28 13:54:54.919 8809/8809/exemples.android D/PlaceholderFragment: constructor
6. 05-28 13:54:54.963 8809/8809/exemples.android D/MainActivity: getItem[0]
7. 05-28 13:54:54.963 8809/8809/exemples.android D/MainActivity: getItem[1]
8. 05-28 13:54:54.963 8809/8809/exemples.android D/PlaceholderFragment: onCreateView 2
9. 05-28 13:54:54.965 8809/8809/exemples.android D/PlaceholderFragment: afterViews 2
10. 05-28 13:54:54.966 8809/8809/exemples.android D/PlaceholderFragment: onCreateView 1
11. 05-28 13:54:54.968 8809/8809/exemples.android D/PlaceholderFragment: afterViews 1
12. 05-28 13:54:54.968 8809/8809/exemples.android D/PlaceholderFragment: onResume 1
13. 05-28 13:54:54.968 8809/8809/exemples.android D/PlaceholderFragment: onResume setText 1
14. 05-28 13:54:54.968 8809/8809/exemples.android D/PlaceholderFragment: onResume 2
15. 05-28 13:54:54.968 8809/8809/exemples.android D/PlaceholderFragment: onResume setText 2
16. 05-28 13:54:55.536 8809/8809/exemples.android D/menu: création menu en cours

```

- ligne 1 : construction de l'unique activité ;
- ligne 2 : méthode [afterViews] de l'activité : ses champs annotés par [@ViewById] sont initialisés ;
- lignes 3-5 : construction des trois fragments ;
- lignes 6-7 : le conteneur de fragments [ViewPager] demande les deux premiers fragments ;
- lignes 8-9 : méthodes du fragment 2 ;
- lignes 10-11 : méthodes du fragment 1 ;
- lignes 12-13 : méthode [onResume] du fragment 1 ;
- lignes 14-15 : méthode [onResume] du fragment 2 ;
- ligne 16 : création menu de l'activité ;

On notera qu'on a la réponse à une question posée précédemment : la méthode [onResume] du fragment 1 par exemple (ligne 12) s'exécute après la méthode [afterViews] du fragment (ligne 11). Donc lorsque la méthode [onResume] s'exécute, elle peut utiliser les champs annotés par [@ViewById]. Nous pourrons donc désormais écrire la méthode [onResume] de la façon suivante :

```

1.     @Override
2.     public void onResume() {
3.         Log.d("PlaceholderFragment", String.format("onResume %s", getArguments().getInt(ARG_SECTION_NUMBER)));
4.         // parent
5.         super.onResume();
6.         // affichage
7.         textViewInfo.setText(getString(R.string.section_format, getArguments().getInt(ARG_SECTION_NUMBER)));
8.     }

```

Maintenant passons de l'onglet 1 à l'onglet 2. Les nouveaux logs sont les suivants :

```

1. 05-28 14:01:42.786 8809-8809/exemples.android D/MainActivity: getItem[2]
2. 05-28 14:01:42.786 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 3
3. 05-28 14:01:42.789 8809-8809/exemples.android D/PlaceholderFragment: afterViews 3
4. 05-28 14:01:42.789 8809-8809/exemples.android D/PlaceholderFragment: onResume 3
5. 05-28 14:01:42.789 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 3

```

- ligne 1 : le conteneur de fragments [ViewPager] demande le fragment n° 3 ;
- lignes 2-3 : méthodes du fragment n° 3. On rappelle que ce fragment avait été instancié dès le démarrage de l'application ;
- lignes 4-5 : la méthode [onResume] du fragment n° 3 est exécutée. On rappelle que c'est le fragment n° 2 qui est affiché ;

Maintenant passons de l'onglet 2 à l'onglet 3. Il n'y a aucun log. Donc aucune des méthodes [onCreateView, afterViews, onResume] du fragment n° 3 n'est exécutée. Il affiche correctement le texte [Hello World from section:3] uniquement parce que ce texte avait déjà été créé à l'étape précédente lors de l'affichage du fragment n° 2. On se rappelle en effet qu'à cette étape, la méthode [onResume] du fragment n° 3 avait été exécutée. On se rend compte ici que pas plus que la méthode [onCreateView], la méthode [onResume] ne peut être utilisée pour mettre à jour le fragment 3. S'il avait fallu changer le texte affiché par le fragment, aucune de ces deux méthodes ne pouvait le faire.

Maintenant, revenons de l'onglet n° 3 à l'onglet n° 1. Les logs sont alors les suivants :

```

1. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 1
2. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: afterViews 1
3. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: onResume 1
4. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 1

```

On voit que toutes les méthodes du fragment 1 ont été exécutées. On voit que la méthode *getItem* n'a pas été appelée. On l'a dit, cette méthode n'est appelée qu'une fois pour chaque fragment ;

Maintenant, passons de l'onglet 1 à l'onglet adjacent 2. On a les logs suivants :

```

1. 05-28 14:12:59.526 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 3
2. 05-28 14:12:59.527 8809-8809/exemples.android D/PlaceholderFragment: afterViews 3
3. 05-28 14:12:59.527 8809-8809/exemples.android D/PlaceholderFragment: onResume 3
4. 05-28 14:12:59.527 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 3

```

Etonnant non ? Toutes les méthodes du fragment n° 3 sont réexécutées.

Pour comprendre ces phénomènes, il faut savoir que par défaut, lorsque le conteneur de fragments va afficher le fragment i, il initialise les fragments i-1, i, i+1. Relisons les logs à la lumière de cette information.

Tout d'abord les logs au démarrage de l'application :

```

1. 05-28 13:54:54.801 8809-8809/exemples.android D/MainActivity: constructor
2. 05-28 13:54:54.901 8809-8809/exemples.android D/MainActivity: afterViews
3. 05-28 13:54:54.919 8809-8809/exemples.android D/PlaceholderFragment: constructor
4. 05-28 13:54:54.919 8809-8809/exemples.android D/PlaceholderFragment: constructor
5. 05-28 13:54:54.919 8809-8809/exemples.android D/PlaceholderFragment: constructor
6. 05-28 13:54:54.963 8809-8809/exemples.android D/MainActivity: getItem[0]
7. 05-28 13:54:54.963 8809-8809/exemples.android D/MainActivity: getItem[1]
8. 05-28 13:54:54.963 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 2
9. 05-28 13:54:54.965 8809-8809/exemples.android D/PlaceholderFragment: afterViews 2
10. 05-28 13:54:54.966 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 1
11. 05-28 13:54:54.968 8809-8809/exemples.android D/PlaceholderFragment: afterViews 1
12. 05-28 13:54:54.968 8809-8809/exemples.android D/PlaceholderFragment: onResume 1
13. 05-28 13:54:54.968 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 1
14. 05-28 13:54:54.968 8809-8809/exemples.android D/PlaceholderFragment: onResume 2
15. 05-28 13:54:54.968 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 2
16. 05-28 13:54:55.536 8809-8809/exemples.android D/menu: création menu en cours

```

Parce que le conteneur de fragments va afficher le fragment 1, les fragments 1 et 2 sont initialisés (lignes 8-15).

On passe maintenant de l'onglet 1 à l'onglet 2 :

```
1. 05-28 14:01:42.786 8809-8809/exemples.android D/MainActivity: getItem[2]
2. 05-28 14:01:42.786 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 3
3. 05-28 14:01:42.789 8809-8809/exemples.android D/PlaceholderFragment: afterViews 3
4. 05-28 14:01:42.789 8809-8809/exemples.android D/PlaceholderFragment: onResume 3
5. 05-28 14:01:42.789 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 3
```

Parce que le conteneur de fragments va afficher le fragment 2, les fragments 1, 2 et 3 doivent être initialisés. Les fragments 1 et 2 le sont déjà depuis l'étape précédente. Le fragment 3 est initialisé au lignes 2-5.

On passe de l'onglet 2 à l'onglet 3. Il n'y a pas de logs. Parce que le conteneur de fragments va afficher le fragment 3, les fragments 2 et 3 doivent être initialisés. Or depuis l'étape précédente, ils le sont déjà. Ce qu'on ne voit pas ici, c'est que le fragment 1 qui n'est pas adjacent au fragment 3 perd son état qui n'est pas conservé en mémoire.

On passe de l'onglet 3 à l'onglet 1. Les logs sont les suivants :

```
1. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 1
2. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: afterViews 1
3. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: onResume 1
4. 05-28 14:11:18.353 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 1
```

Parce que le conteneur de fragments va afficher le fragment 1, le fragment 2 doit lui aussi être initialisé. Il l'est depuis l'étape précédente. Dans cette même étape, l'état du fragment 1 avait été perdu. Il est donc réinitialisé aux lignes 1-4. Ce qu'on ne voit pas ici, c'est que le fragment 3 qui n'est pas adjacent au fragment 1 perd son état qui n'est alors pas conservé en mémoire.

Lorsqu'on passe de l'onglet 1 à l'onglet adjacent 2, on a les logs suivants :

```
1. 05-28 14:12:59.526 8809-8809/exemples.android D/PlaceholderFragment: onCreateView 3
2. 05-28 14:12:59.527 8809-8809/exemples.android D/PlaceholderFragment: afterViews 3
3. 05-28 14:12:59.527 8809-8809/exemples.android D/PlaceholderFragment: onResume 3
4. 05-28 14:12:59.527 8809-8809/exemples.android D/PlaceholderFragment: onResume setText 3
```

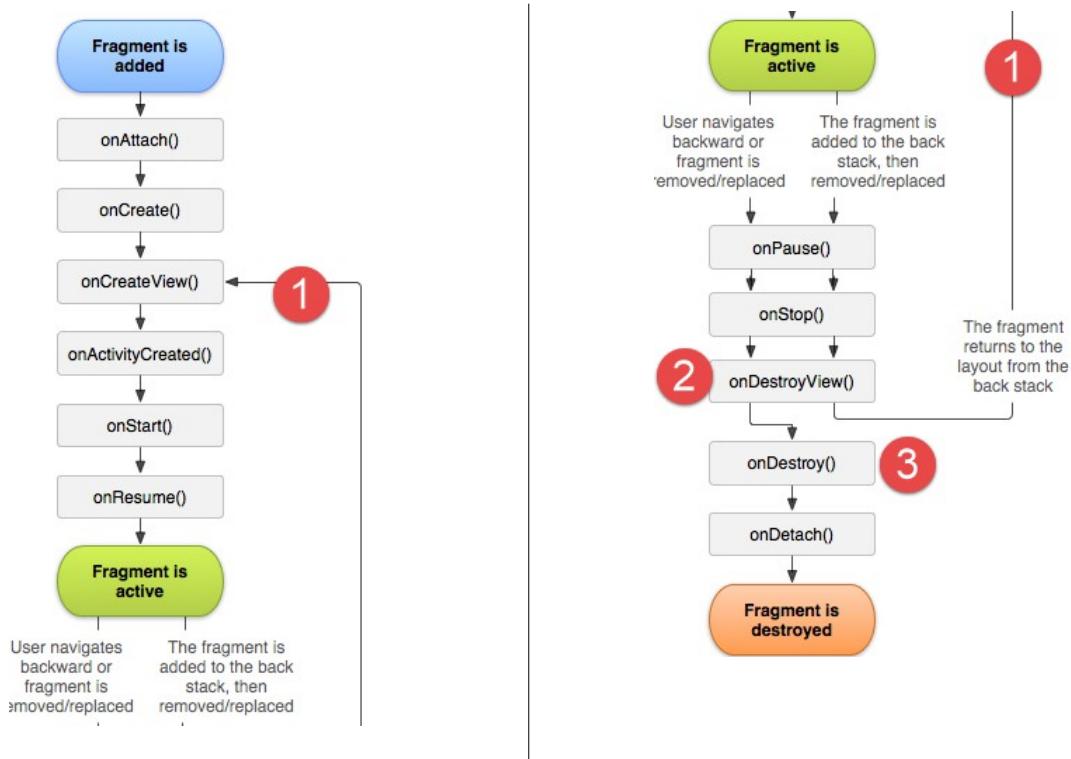
Parce que le conteneur de fragments va afficher le fragment 2, les fragments 1, 2 et 3 doivent être initialisés. Les fragments 1 et 2 le sont déjà depuis l'étape précédente. Le fragment 3 est initialisé au lignes 1-4.

Qu'avons-nous appris ?

- que la gestion par défaut des fragments est très particulière et qu'il faut la connaître si on ne veut pas perdre ses cheveux. On peut changer ce mode de gestion et nous le ferons un peu plus loin ;
- qu'avec cette gestion par défaut, aucune des méthodes [onCreateView, onResume] ne peut être utilisée pour mettre à jour le fragment qui va être affiché car on n'est pas sûr qu'elles vont être exécutées ;

1.8.6 onDestroyView

La méthode [onDestroyView] fait partie du cycle de vie des fragments (cf paragraphe 1.7.6, page 75) :



On voit que dans le cycle de vie d'un fragment :

- la méthode [onCreateView] peut être exécutée plusieurs fois ;
- avant de retourner à la méthode [onCreateView] ultérieurement, il y a forcément un passage par la méthode [onDestroyView] [2] ;

Nous allons insérer ces méthodes dans les fragments pour mieux suivre leur cycle de vie. Le code du fragment devient le suivant :

```

1. package exemples.android;
2.
3. import android.os.Bundle;
4. import android.support.v4.app.Fragment;
5. import android.util.Log;
6. import android.view.LayoutInflater;
7. import android.view.View;
8. import android.view.ViewGroup;
9. import android.widget.TextView;
10. import org.androidannotations.annotations.AfterViews;
11. import org.androidannotations.annotations.EFragment;
12. import org.androidannotations.annotations.ViewById;
13.
14. // un fragment est une vue affichée par un conteneur de fragments
15. @EFragment(R.layout.fragment_main)
16. public class PlaceholderFragment extends Fragment {
17.
18. ...
19.
20. @Override
21. public void onDestroyView() {
22.     // log
23.     Log.d("PlaceholderFragment", String.format("onDestroyView %s", getArguments().getInt(ARG_SECTION_NUMBER)));
24.     // parent
25.     super.onDestroyView();
26. }
27.
28. }
```

Exécutons l'application. Les premiers logs sont les suivants :

```

1. 06-03 02:45:42.163 2346-2346/exemples.android D/MainActivity: constructor
2. 06-03 02:45:42.331 2346-2346/exemples.android D/MainActivity: afterViews
3. 06-03 02:45:42.341 2346-2346/exemples.android D/PlaceholderFragment: constructor
4. 06-03 02:45:42.341 2346-2346/exemples.android D/PlaceholderFragment: constructor
5. 06-03 02:45:42.341 2346-2346/exemples.android D/PlaceholderFragment: constructor
6. 06-03 02:45:42.515 2346-2346/exemples.android D/MainActivity: getItem[0]
7. 06-03 02:45:42.516 2346-2346/exemples.android D/MainActivity: getItem[1]
```

```

8. 06-03 02:45:42.517 2346-2346/exemples.android D/PlaceholderFragment: onCreateView 2
9. 06-03 02:45:42.520 2346-2346/exemples.android D/PlaceholderFragment: afterViews 2
10. 06-03 02:45:42.523 2346-2346/exemples.android D/PlaceholderFragment: onCreateView 1
11. 06-03 02:45:42.524 2346-2346/exemples.android D/PlaceholderFragment: afterViews 1
12. 06-03 02:45:42.524 2346-2346/exemples.android D/PlaceholderFragment: onResume 1
13. 06-03 02:45:42.524 2346-2346/exemples.android D/PlaceholderFragment: onResume setText 1
14. 06-03 02:45:42.525 2346-2346/exemples.android D/PlaceholderFragment: onResume 2
15. 06-03 02:45:42.525 2346-2346/exemples.android D/PlaceholderFragment: onResume setText 2
16. 06-03 02:45:44.596 2346-2346/exemples.android D/menu: création menu en cours

```

- ligne 1 : construction de l'unique activité ;
- ligne 2 : méthode [afterViews] de l'activité : ses champs annotés par [@ViewById] sont initialisés ;
- lignes 3-5 : construction des trois fragments ;
- lignes 6-7 : le conteneur de fragments [ViewPager] demande les deux premiers fragments ;
- lignes 8-9 : la vue du fragment 2 est créée (pas forcément rendue visible) ;
- lignes 10-11 : la vue du fragment 1 est créée (pas forcément rendue visible) ;
- lignes 12-13 : méthode [onResume] du fragment 1 ;
- lignes 14-15 : méthode [onResume] du fragment 2 ;
- ligne 16 : création menu de l'activité ;

Passons de l'onglet 1 à l'onglet 3 :

```

1. 06-03 02:50:02.685 2346-2346/exemples.android D>MainActivity: getItem[2]
2. 06-03 02:50:02.685 2346-2346/exemples.android D/PlaceholderFragment: onCreateView 3
3. 06-03 02:50:02.686 2346-2346/exemples.android D/PlaceholderFragment: afterViews 3
4. 06-03 02:50:02.686 2346-2346/exemples.android D/PlaceholderFragment: onResume 3
5. 06-03 02:50:02.686 2346-2346/exemples.android D/PlaceholderFragment: onResume setText 3
6. 06-03 02:50:03.024 2346-2346/exemples.android D/PlaceholderFragment: onDestroyView 1

```

- ligne 1 : le conteneur de fragments demande le 3ième fragment ;
- lignes 2-3 : la vue du fragment 3 est créée (pas forcément affichée) ;
- lignes 4-5 : la méthode [onResume] du fragment 3 est exécutée ;
- ligne 6 : la méthode [onDestroyView] du fragment 1 est exécutée. Cela implique que lorsque l'utilisateur va revenir au fragment 1 ou à un fragment adjacent, le cycle de vie de ce fragment va être réexécuté ;

On revient de l'onglet 3 à l'onglet 1 :

```

1. 06-03 02:53:46.255 2346-2346/exemples.android D/PlaceholderFragment: onCreateView 1
2. 06-03 02:53:46.256 2346-2346/exemples.android D/PlaceholderFragment: afterViews 1
3. 06-03 02:53:46.256 2346-2346/exemples.android D/PlaceholderFragment: onResume 1
4. 06-03 02:53:46.256 2346-2346/exemples.android D/PlaceholderFragment: onResume setText 1
5. 06-03 02:53:46.604 2346-2346/exemples.android D/PlaceholderFragment: onDestroyView 3

```

- lignes 1-4 : le cycle de vie du fragment 1 est réexécuté parce qu'il avait subi un [onDestroyView] ;
- ligne 5 : c'est maintenant le fragment 3 qui voit sa méthode [onDestroyView] exécutée. Là encore, lorsque l'utilisateur va revenir au fragment 3 ou à un fragment adjacent, le cycle de vie de ce fragment va être réexécuté ;

1.8.7 setUserVisibleHint

La méthode [onCreateView] du cycle de vie instancie la vue associée au fragment mais ne la rend pas forcément visible. C'est ce que nous allons voir maintenant. La méthode [Fragment.setUserVisibleHint] est exécutée à chaque fois que la visibilité du fragment change. On ajoute cette méthode au code du fragment :

```

1. package exemples.android;
2.
3. ....
4.
5. // un fragment est une vue affichée par un conteneur de fragments
6. @EFragment(R.layout.fragment_main)
7. public class PlaceholderFragment extends Fragment {
8.
9.     // composant de l'interface visuelle
10.    @ViewById(R.id.section_label)
11.    protected TextView textViewInfo;
12.
13.    ...
14.
15.    @Override
16.    public void setUserVisibleHint(boolean isVisibleToUser) {
17.        // log
18.        Log.d("PlaceholderFragment", String.format("setUserVisibleHint %s isVisibleToUser=%s",
19.            getArguments().getInt(ARG_SECTION_NUMBER), isVisibleToUser));
20.    }

```

Au démarrage, les logs sont les suivants :

```
1. 06-03 03:06:13.263 20586-20586/exemples.android D/MainActivity: constructor
2. 06-03 03:06:13.291 20586-20586/exemples.android D/MainActivity: afterViews
3. 06-03 03:06:13.324 20586-20586/exemples.android D/PlaceholderFragment: constructor
4. 06-03 03:06:13.324 20586-20586/exemples.android D/PlaceholderFragment: constructor
5. 06-03 03:06:13.329 20586-20586/exemples.android D/PlaceholderFragment: constructor
6. 06-03 03:06:13.504 20586-20586/exemples.android D/MainActivity: getItem[0]
7. 06-03 03:06:13.504 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
8. 06-03 03:06:13.504 20586-20586/exemples.android D/MainActivity: getItem[1]
9. 06-03 03:06:13.504 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=false
10. 06-03 03:06:13.504 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=true
11. 06-03 03:06:13.511 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 1
12. 06-03 03:06:13.519 20586-20586/exemples.android D/PlaceholderFragment: afterViews 1
13. 06-03 03:06:13.519 20586-20586/exemples.android D/PlaceholderFragment: onResume 1
14. 06-03 03:06:13.519 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 1
15. 06-03 03:06:13.520 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 2
16. 06-03 03:06:13.527 20586-20586/exemples.android D/PlaceholderFragment: afterViews 2
17. 06-03 03:06:13.527 20586-20586/exemples.android D/PlaceholderFragment: onResume 2
18. 06-03 03:06:13.527 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 2
19. 06-03 03:06:15.075 20586-20586/exemples.android D/menu: création menu en cours
```

- les logs des lignes 7, 9-10 montrent que seul le fragment 1 devient visible. On voit également qu'il devient visible avant exécution de sa méthode [onCreateView] ;

Passons de l'onglet 1 à l'onglet 2 :

```
1. 06-03 03:10:15.215 20586-20586/exemples.android D/MainActivity: getItem[2]
2. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=false
3. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
4. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=true
5. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 3
6. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: afterViews 3
7. 06-03 03:10:15.216 20586-20586/exemples.android D/PlaceholderFragment: onResume 3
8. 06-03 03:10:15.216 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 3
```

- le fragment 1 est caché (ligne 3), le fragment 2 est affiché (ligne 4) ;

Passons de l'onglet 2 à l'onglet 3 :

```
1. 06-03 03:12:06.238 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=false
2. 06-03 03:12:06.238 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=true
3. 06-03 03:12:06.239 20586-20586/exemples.android D/PlaceholderFragment: onDestroyView 1
```

- le fragment 2 est caché (ligne 1), le fragment 3 est affiché (ligne 2) ;

Revenons à l'onglet 1 :

```
1. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
2. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=false
3. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=true
4. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 1
5. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: afterViews 1
6. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: onResume 1
7. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 1
8. 06-03 03:13:10.789 20586-20586/exemples.android D/PlaceholderFragment: onDestroyView 3
```

- le fragment 3 est caché (ligne 2), le fragment 1 est affiché (ligne 3) ;

Qu'avons-nous appris :

- la méthode [setUserVisibleHint] est exécutée une fois avec la propriété [isVisibleToUser] à *true*, pour le fragment qui va être affiché ;
- on ne peut pas situer quand va être exécutée cette méthode par rapport au cycle de vie du fragment. Ainsi, pour le fragment 1, la méthode [setUserVisibleHint, true] a été exécutée avant la méthode [onCreateView] du début du cycle de vie de ce fragment, alors que pour les fragments 2 et 3 c'est l'inverse qui s'est produit ;

1.8.8 setOffscreenPageLimit

Les logs précédents montrent que lorsque le conteneur de fragments [ViewPager] s'apprête à afficher le fragment n° i, il exécute, si ce n'est déjà fait, le cycle de vie des fragments adjacents i-1 et i+1. Ce fonctionnement peut être contrôlé par la méthode [ViewPager].setOffscreenPageLimit :

```
1. // offset des fragments
```

```
2.     [ViewPager].setOffscreenPageLimit(n);
```

Avec l'instruction ci-dessus,

1. lorsque le conteneur de fragments [ViewPager] s'apprête à afficher le fragment n° i, il exécute, si ce n'est déjà fait, le cycle de vie des fragments adjacents de l'intervalle [i-n, i+n] ;
2. si on affiche ensuite le fragment j :
 - le même phénomène se reproduit pour les fragments adjacents de l'intervalle [j-n, j+n] ;
 - les fragments initialisés lors de l'étape 1 et qui ne sont plus dans l'adjacence [j-n, j+n] du nouveau fragment, peuvent alors subir une opération [onDestroyView]. Néanmoins, j'ai pu observer sur d'autres applications, notamment celle du chapitre 3, page 341, que ce n'était pas systématiquement le cas ;

Nous modifions la méthode [MainActivity.afterViews] de la façon suivante :

```
1.     @AfterViews
2.     protected void afterViews() {
3.         Log.d("MainActivity", "afterViews");
4.
5.         // barre d'outils
6.         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
7.         setSupportActionBar(toolbar);
8.
9.         // le gestionnaire de fragments
10.        mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
11.
12.        // le conteneur de fragments est associé au gestionnaire de fragments
13.        // c-à-d que le fragment n° i du conteneur de fragments est le fragment n° i délivré par le gestionnaire de fragments
14.        mViewPager.setAdapter(mSectionsPagerAdapter);
15.
16.        // on inhibe le swipe entre fragments
17.        mViewPager.setSwipeEnabled(false);
18.
19.        // offset des fragments
20.        mViewPager.setOffscreenPageLimit(mSectionsPagerAdapter.getCount() - 1);
21.
22.        // la barre d'onglets est également associée au conteneur de fragments
23.        // c-à-d que l'onglet n° i affiche le fragment n° i du conteneur
24.        tabLayout.setupWithViewPager(mViewPager);
25.
26.        // bouton flottant
27.        fab.setOnClickListener(new View.OnClickListener() {
28.            @Override
29.            public void onClick(View view) {
30.                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
31.                    .setAction("Action", null).show();
32.            }
33.        });
34.    }
```

- ligne 20 : nous mettons le nombre de fragments adjacents à initialiser, au nombre total de fragments -1. Ainsi, au démarrage, lorsque le conteneur de fragments va afficher le fragment n° 1, il va en même temps initialiser les fragments 2, 3, ..., n avec n=1+mSectionsPagerAdapter.getCount() - 1=mSectionsPagerAdapter.getCount(). Ce sont donc tous les fragments qui vont être initialisés. Lorsque la fenêtre d'affichage va se déplacer sur un autre fragment, le conteneur de fragments :
 - va découvrir que tous les fragments adjacents du nouveau fragment sont déjà initialisés et il ne fera donc pas leur initialisation ;
 - comme l'adjacence du nouveau fragment couvre elle-aussi la totalité des fragments, aucun ne sera « désinitialisé » par le conteneur de fragments ;

Au total, on devrait voir tous les fragments instanciés et initialisés au démarrage de l'application et puis plus jamais ensuite. C'est ce que nous vérifions maintenant en examinant les logs.

Au démarrage, nous avons les logs suivants :

```
1. 06-03 03:30:55.411 10344-10344/exemples.android W/System: ClassLoader referenced unknown path: /data/app/exemples.android-1/lib/x86
2. 06-03 03:30:55.417 10344-10344/exemples.android D/MainActivity: constructor
3. 06-03 03:30:55.460 10344-10344/exemples.android D/MainActivity: afterViews
4. 06-03 03:30:55.474 10344-10344/exemples.android D/PlaceholderFragment: constructor
5. 06-03 03:30:55.474 10344-10344/exemples.android D/PlaceholderFragment: constructor
6. 06-03 03:30:55.474 10344-10344/exemples.android D/PlaceholderFragment: constructor
7. 06-03 03:30:55.559 10344-10344/exemples.android D/MainActivity: getItem[0]
8. 06-03 03:30:55.559 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
9. 06-03 03:30:55.560 10344-10344/exemples.android D/MainActivity: getItem[1]
10. 06-03 03:30:55.560 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=false
11. 06-03 03:30:55.560 10344-10344/exemples.android D/MainActivity: getItem[2]
```

```

12. 06-03 03:30:55.560 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=false
13. 06-03 03:30:55.560 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=true
14. 06-03 03:30:55.560 10344-10344/exemples.android D/PlaceholderFragment: onCreateView 1
15. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: afterViews 1
16. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: onResume 1
17. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: onResume setText 1
18. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: onCreateView 2
19. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: afterViews 2
20. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: onResume 2
21. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: onResume setText 2
22. 06-03 03:30:55.564 10344-10344/exemples.android D/PlaceholderFragment: onCreateView 3
23. 06-03 03:30:55.565 10344-10344/exemples.android D/PlaceholderFragment: afterViews 3
24. 06-03 03:30:55.565 10344-10344/exemples.android D/PlaceholderFragment: onResume 3
25. 06-03 03:30:55.565 10344-10344/exemples.android D/PlaceholderFragment: onResume setText 3
26. 06-03 03:30:56.798 10344-10344/exemples.android D/menu: création menu en cours

```

- lignes 4-6 : construction des trois fragments ;
- lignes 7, 9, 11 : le conteneur de fragments réclame les trois fragments. Dans la version précédente, il en réclamait deux ;
- lignes 14-25 : le cycle de vie des trois fragments s'exécute ;

Passons maintenant de l'onglet 1 à l'onglet 2 :

```

1. 06-03 03:34:03.388 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
2. 06-03 03:34:03.388 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=true

```

Passons de l'onglet 2 à l'onglet 3 :

```

1. 06-03 03:34:43.292 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=false
2. 06-03 03:34:43.292 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=true

```

Puis de l'onglet 3 à l'onglet 1 :

```

1. 06-03 03:35:32.666 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=false
2. 06-03 03:35:32.666 10344-10344/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=true

```

Les logs confirment la théorie. Tous les fragments ont été instanciés et initialisés au démarrage. Ensuite les méthodes de leur cycle de vie ne sont plus exécutées. On a là un fonctionnement très prévisible des fragments qui facilite énormément leur utilisation.

Ce que nous voulons trouver, c'est un moyen de mettre à jour un fragment qui va être affiché, ceci quelque soit l'adjacence de fragments choisie par le développeur. Les logs nous ont montré deux choses :

- la méthode [setUserVisibleHint, true] est toujours exécutée pour le fragment qui va être affiché et pas pour les autres ;
- cette événement peut se produire avant ou après le cycle de vie du fragment. Cela dépend de l'adjacence de fragments choisie par le développeur. C'est un problème, car si le cycle de vie n'a pas encore eu lieu, cela signifie que le fragment ne peut pas être mis à jour par la méthode [setUserVisibleHint, true] ;

Les logs au démarrage de l'application lorsque l'adjacence des fragments était 1 ont été les suivants :

```

1. 06-03 03:06:13.263 20586-20586/exemples.android D/MainActivity: constructor
2. 06-03 03:06:13.291 20586-20586/exemples.android D/MainActivity: afterViews
3. 06-03 03:06:13.324 20586-20586/exemples.android D/PlaceholderFragment: constructor
4. 06-03 03:06:13.324 20586-20586/exemples.android D/PlaceholderFragment: constructor
5. 06-03 03:06:13.329 20586-20586/exemples.android D/PlaceholderFragment: constructor
6. 06-03 03:06:13.504 20586-20586/exemples.android D/MainActivity: getItem[0]
7. 06-03 03:06:13.504 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
8. 06-03 03:06:13.504 20586-20586/exemples.android D/MainActivity: getItem[1]
9. 06-03 03:06:13.504 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=false
10. 06-03 03:06:13.504 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=true
11. 06-03 03:06:13.511 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 1
12. 06-03 03:06:13.519 20586-20586/exemples.android D/PlaceholderFragment: afterViews 1
13. 06-03 03:06:13.519 20586-20586/exemples.android D/PlaceholderFragment: onResume 1
14. 06-03 03:06:13.519 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 1
15. 06-03 03:06:13.520 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 2
16. 06-03 03:06:13.527 20586-20586/exemples.android D/PlaceholderFragment: afterViews 2
17. 06-03 03:06:13.527 20586-20586/exemples.android D/PlaceholderFragment: onResume 2
18. 06-03 03:06:13.527 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 2
19. 06-03 03:06:15.075 20586-20586/exemples.android D/menu: création menu en cours

```

- on voit que lorsque le fragment 1 devient visible, sa vue n'a pas encore été créée. On ne peut donc y toucher. Cela pourra alors se faire au cours du cycle de vie du fragment, par exemple dans les méthodes [onCreateView] (ligne 11) ou [onResume] (lignes 13-14). Comme nous utilisons les annotations AA, nous n'avons normalement pas à écrire la méthode [onCreateView]. C'est donc la méthode [onResume] qui semble la plus adaptée ici pour mettre à jour le fragment 1 ;

Lorsque nous sommes passés de l'onglet 1 à l'onglet 2, les logs ont été les suivants :

```

1. 06-03 03:10:15.215 20586-20586/exemples.android D/MainActivity: getItem[2]
2. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=false
3. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
4. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=true
5. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 3
6. 06-03 03:10:15.215 20586-20586/exemples.android D/PlaceholderFragment: afterViews 3
7. 06-03 03:10:15.216 20586-20586/exemples.android D/PlaceholderFragment: onResume 3
8. 06-03 03:10:15.216 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 3

```

Cette fois, on n'a que la méthode [setUserVisibleHint, true] de la ligne 4 pour mettre à jour le fragment 2 ;

Lorsque nous sommes passés de l'onglet 2 à l'onglet 3, les logs ont été les suivants :

```

1. 06-03 03:12:06.238 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 isVisibleToUser=false
2. 06-03 03:12:06.238 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=true
3. 06-03 03:12:06.239 20586-20586/exemples.android D/PlaceholderFragment: onDestoryView 1

```

Ici, on n'a que la méthode [setUserVisibleHint, true] de la ligne 2 pour mettre à jour le fragment 3 ;

Lorsque nous sommes passés de l'onglet 3 à l'onglet 1, les logs ont été les suivants :

```

1. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=false
2. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 isVisibleToUser=false
3. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 isVisibleToUser=true
4. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: onCreateView 1
5. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: afterViews 1
6. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: onResume 1
7. 06-03 03:13:10.427 20586-20586/exemples.android D/PlaceholderFragment: onResume setText 1
8. 06-03 03:13:10.789 20586-20586/exemples.android D/PlaceholderFragment: onDestoryView 3

```

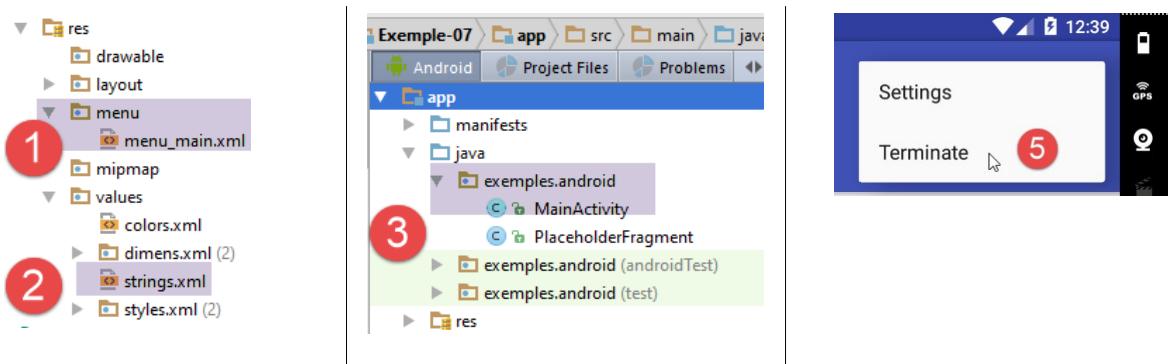
Ici, il faut utiliser la méthode [onResume] du fragment 1 (lignes 6-7) pour mettre à jour le fragment 1.

Donc sur cet exemple, on voit que pour mettre à jour un fragment qui va s'afficher, on dispose de deux méthodes : [setUserVisibleHint] et [onResume].

Nous allons mettre en oeuvre cette solution dans un nouveau projet où chaque fragment devra afficher le nombre de fois où il a été affiché, ce qu'on appellera une visite. Il faudra donc mettre à jour son affichage à chaque fois qu'il sera affiché. C'est bien le problème qu'on cherche à résoudre.

Auparavant, nous examinons la dernière étape de la vie d'une activité ou d'un fragment, celle où il est détruit. Le système peut prendre l'initiative de supprimer une activité si d'autres activités plus prioritaires réclament des ressources indisponibles. Pour libérer celles-ci, le système va prendre l'initiative de supprimer certaines activités. La méthode [onDestroy] de l'activité et des fragments va alors être appelée.

1.8.9 OnDestroy



Nous allons permettre à l'utilisateur de supprimer l'activité au moyen d'une option de menu [5]. Pour cela, nous ajoutons une nouvelle option de menu dans le fichier [menu_main.xml] [1] :

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.   xmlns:app="http://schemas.android.com/apk/res-auto"
3.   xmlns:tools="http://schemas.android.com/tools"
4.   tools:context="exemples.android.MainActivity">
5.   <item android:id="@+id/action_settings"
6.     android:title="@string/action_settings"

```

```

7.      android:orderInCategory="100"
8.      app:showAsAction="never"/>
9.      <item android:id="@+id/action_terminate"
10.         android:title="@string/action_terminate"
11.         android:orderInCategory="100"
12.         app:showAsAction="never"/>
13.  </menu>

```

On fait un simple copier / coller de la 1ère option de menu et on adapte le résultat (lignes 9 et 10). Le libellé de cette nouvelle option est ajouté dans le fichier [strings.xml] [2] :

```

1.  <resources>
2.    <string name="app_name">Exemple-07</string>
3.    <string name="action_settings">Settings</string>
4.    <string name="action_terminate">Terminate</string>
5.    <string name="section_format">Hello World from section: %1$d</string>
6.  </resources>

```

Enfin, dans la classe [MainActivity], on gère le clic sur l'option [Terminate] :

```

1.  @Override
2.  public boolean onOptionsItemSelected(MenuItem item) {
3.    Log.d("menu", "onOptionsItemSelected");
4.    // Handle action bar item clicks here. The action bar will
5.    // automatically handle clicks on the Home/Up button, so long
6.    // as you specify a parent activity in AndroidManifest.xml.
7.    int id = item.getItemId();
8.
9.    //noinspection SimplifiableIfStatement
10.   if (id == R.id.action_settings) {
11.     Log.d("menu", "action_settings selected");
12.     return true;
13.   }
14.   if (id == R.id.action_terminate) {
15.     Log.d("menu", "action_terminate selected");
16.     //on termine l'activité
17.     finish();
18.     return true;
19.   }
20.   // parent
21.   return super.onOptionsItemSelected(item);
22. }

```

- lignes 14-19 : on fait un copier / coller des lignes 10-13 et on adapte le code à la nouvelle option ;
- ligne 17 : l'activité est terminée par action logicielle ;

Maintenant exécutons cette nouvelle version, puis dès que la première vue est affichée, cliquons sur l'option de menu [Terminate]. Les logs sont alors les suivants :

```

1.  06-04 12:35:32.996 15994-15994/exemples.android D/menu: onOptionsItemSelected
2.  06-04 12:35:32.996 15994-15994/exemples.android D/menu: action_terminate selected
3.  06-04 12:35:33.561 15994-15994/exemples.android D>MainActivity: onDestroy
4.  06-04 12:35:33.561 15994-15994/exemples.android D/PlaceholderFragment: onDestroyView 1
5.  06-04 12:35:33.562 15994-15994/exemples.android D/PlaceholderFragment: onDestory 1
6.  06-04 12:35:33.562 15994-15994/exemples.android D/PlaceholderFragment: onDestoryView 2
7.  06-04 12:35:33.562 15994-15994/exemples.android D/PlaceholderFragment: onDestory 2
8.  06-04 12:35:33.562 15994-15994/exemples.android D/PlaceholderFragment: onDestoryView 3
9.  06-04 12:35:33.562 15994-15994/exemples.android D/PlaceholderFragment: onDestory 3

```

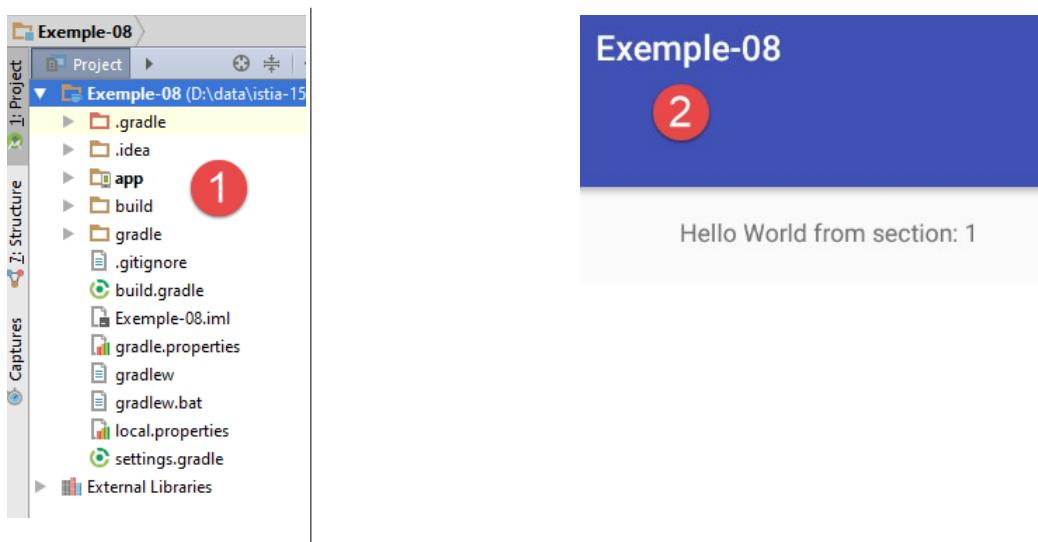
- lignes 1-2 : clic sur l'option [Terminate] ;
- ligne 4 : la méthode [onDestroy] de l'activité est appelée ;
- lignes 4-5 : la méthode [onDestroyView] du fragment 1 est appelée puis sa méthode[onDestory] ;
- lignes 6-9 : cette opération se répète pour les deux autres fragments ;

On se rappellera donc que la méthode [onDestroy] de l'activité et des fragments est appelée lorsque l'activité va être supprimée par le système, le développeur ou l'utilisateur. On peut utiliser cette méthode pour sauvegarder des informations, par exemple localement sur la tablette afin de les retrouver lorsque l'utilisateur relancera de nouveau l'application.

1.9 Exemple-08 : mise à jour d'un fragment avec une adjacence de fragments variable

1.9.1 Crédation du projet

On duplique le projet [Exemple-07] dans [Exemple-08]. Pour cela, on suivra la procédure décrite pour dupliquer [Exemple-02] dans [Exemple-03] au paragraphe 1.4, page 36 .



1.9.2 Réécriture du fragment [PlaceholderFragment]

Le nouveau code du fragment [PlaceholderFragment] est le suivant. Il fonctionne quelque soit l'adjacence donnée aux fragments (1, partielle, totale) :

```
1. package exemples.android;
2.
3. import android.support.v4.app.Fragment;
4. import android.util.Log;
5. import android.widget.TextView;
6. import org.androidannotations.annotations.AfterViews;
7. import org.androidannotations.annotations.EFragment;
8. import org.androidannotations.annotations.ViewById;
9.
10. // un fragment est une vue affichée par un conteneur de fragments
11. @EFragment(R.layout.fragment_main)
12. public class PlaceholderFragment extends Fragment {
13.
14.     // composant de l'interface visuelle
15.     @ViewById(R.id.section_label)
16.     protected TextView textViewInfo;
17.     // data
18.     private boolean afterViewsDone = false;
19.     private boolean initDone = false;
20.     private String text;
21.     private boolean isVisibleToUser = false;
22.     private boolean updateDone = false;
23.     private int numVisit = 0;
24.
25.     // n° de fragment
26.     private static final String ARG_SECTION_NUMBER = "section_number";
27.
28.     // constructeur
29.     public PlaceholderFragment() {
30.         Log.d("PlaceholderFragment", "constructor");
31.     }
32.
33.
34.     @AfterViews
35.     protected void afterViews() {
```

```

36.     // mémoire
37.     afterViewsDone = true;
38.     // log
39.     Log.d("PlaceholderFragment", String.format("afterViews %s %s", getArguments().getInt(ARG_SECTION_NUMBER), getInfos()));
40.     if (!initDone) {
41.         // texte initial
42.         text = getString(R.string.section_format, getArguments().getInt(ARG_SECTION_NUMBER));
43.         // init done
44.         initDone = true;
45.     }
46.     // affichage texte courant
47.     textViewInfo.setText(text);
48. }
49.
50.
51. @Override
52. public void setUserVisibleHint(boolean isVisibleToUser) {
53. ...
54. }
55.
56. @Override
57. public void onDestroyView() {
58. ...
59. }
60.
61. @Override
62. public void onResume() {
63. ...
64. }
65.
66. // update fragment
67. public void update() {
68.     // le travail à faire dépend du n° de la visite
69.     if (numVisit > 1) {
70.         // log
71.         Log.d("PlaceholderFragment", String.format("update %s : %s", getArguments().getInt(ARG_SECTION_NUMBER), getInfos()));
72.         // texte modifié
73.         textViewInfo.setText(String.format("%s update(%s)", text, (numVisit - 1)));
74.     }
75. }
76.
77. // infos locales pour logs
78. private String getInfos() {
79.     return String.format("numVisit=%s, afterViewsDone=%s, isVisibleToUser=%s, initDone=%s, updateDone=%s", numVisit,
80.     afterViewsDone, isVisibleToUser, initDone, updateDone);
81. }

```

- lignes 34-48 : la méthode `[@AfterViews]` est susceptible d'être exécutée plusieurs fois. On s'en servait pour initialiser le texte du fragment (ligne 42). On le fait toujours mais pour ne le faire qu'une fois, on gère un booléen `[initDone]` (ligne 44) pour indiquer que l'initialisation a été faite et qu'elle n'est pas à refaire ;
- lignes 56-59 : nous introduisons la méthode `[onDestroyView]` pour noter le fait que la prochaine fois que le fragment va être réaffiché, son cycle de vie va être réexécuté ;
- les logs ont montré que deux méthodes peuvent s'exécuter après la méthode `[@AfterViews]` : les méthodes `[setUserVisibleHint]` et `[onResume]`. La méthode `[onResume]` n'est exécutée que lorsque le cycle de vie du fragment est exécuté. La méthode `[setUserVisibleHint]` elle n'est pas toujours exécutée après la méthode `[@AfterViews]`. Les logs ont montré qu'au moins l'une des deux est exécutée après la méthode `[@AfterViews]`. Les logs n'ont jamais montré que les deux pouvaient être exécutées ensemble après la méthode `[@AfterViews]`. C'est soit l'une, soit l'autre. Par précaution, on positionnera un booléen `[updateDone]` lorsqu'une mise à jour a été faite ;

Les méthodes `[setUserVisibleHint]` et `[onResume]` sont les suivantes :

```

1.     // data
2.     private boolean afterViewsDone = false;
3.     private boolean initDone = false;
4.     private String text;
5.     private boolean isVisibleToUser = false;
6.     private boolean updateDone = false;
7.     private int numVisit = 0;
8.
9.     @Override
10.    public void setUserVisibleHint(boolean isVisibleToUser) {
11.        // parent
12.        super.setUserVisibleHint(isVisibleToUser);
13.        // mémoire
14.        this.isVisibleToUser = isVisibleToUser;
15.        // log
16.        Log.d("PlaceholderFragment", String.format("setUserVisibleHint %s : %s", getArguments().getInt(ARG_SECTION_NUMBER),
17.        getInfos()));
18.        // nombre de visites
19.        if (isVisibleToUser) {

```

```

19.     // incrément
20.     numVisit++;
21.     // update fragment
22.     if (afterViewsDone && !updateDone) {
23.         update();
24.         updateDone = true;
25.     }
26. } else {
27.     // le fragment va être caché
28.     updateDone = false;
29. }
30. }
31.
32. @Override
33. public void onResume() {
34.     // parent
35.     super.onResume();
36.     // log
37.     Log.d("PlaceholderFragment", String.format("onResume %s : %s", getArguments().getInt(ARG_SECTION_NUMBER),
getInfos()));
38.     // update
39.     if (isVisibleToUser && !updateDone) {
40.         update();
41.         updateDone = true;
42.     }
43. }

```

- ligne 14 : on mémorise l'état visible ou non du fragment ;
- lignes 22-25 : si le fragment est visible et que la méthode [`@AfterViews`] a été exécutée, la méthode [update] est exécutée et le booléen [updateDone] passé à `true` ;
- lignes 26-28 : si le fragment va être caché, on remet le booléen [updateDone] à `false`. Il nous faut en effet un événement pour réinitialiser à `false` le booléen [updateDone] mis à `true` dès que la méthode [update] est appelée afin que de nouvelles mises à jour puissent se faire. Nous utilisons le fait que le fragment n'est plus visible pour le faire. Lorsqu'il redeviendra visible, la mise à jour du fragment devra se faire de nouveau ;
- lignes 32-42 : les logs montrent que selon l'adjacence choisie pour les fragments, la méthode [onResume] peut s'exécuter alors que le fragment n'est pas visible. S'il n'est pas visible, on ne fait pas la mise à jour (ligne 39) et comme on l'a fait pour [setMenuVisibility], on gère le booléen [updateDone].

Enfin, la méthode [onDestroyView] est la suivante :

```

1.     @Override
2.     public void onDestroyView() {
3.         // parent
4.         super.onDestroyView();
5.         // mise à jour indicateur
6.         afterViewsDone = false;
7.         // log
8.         Log.d("PlaceholderFragment", String.format("onDestroyView %s : %s", getArguments().getInt(ARG_SECTION_NUMBER),
getInfos()));
9.     }

```

La méthode [onDestroyView] est exécutée lorsqu'un cycle de vie du fragment prend fin. Un autre cycle pourra reprendre ultérieurement.

- ligne 6 : la méthode [onDestroyView] supprime tout lien avec la vue attachée au fragment. Il sera recréé au prochain cycle de vie du fragment. Pour l'instant, il nous faut mettre le booléen [afterViews] à `false`, pour indiquer que le lien avec la vue n'existe plus ;

Nous allons exécuter l'application avec 5 fragments ayant une adjacence de 2. Les modifications sont faites dans [MainActivity] :

```

1.     // nombre de fragments
2.     private final int FRAGMENTS_COUNT = 5;
3.     // adjacence des fragments
4.     private final int OFF_SCREEN_PAGE_LIMIT=2;
5.
6.
7.     // le gestionnaire de fragments
8.     private SectionsPagerAdapter mSectionsPagerAdapter;
9.
10.    @AfterViews
11.    protected void afterViews() {
12.        Log.d("MainActivity", "afterViews");
13.
14.        ....
15.
16.        // offset des fragments
17.        mViewPager.setOffscreenPageLimit(OFF_SCREEN_PAGE_LIMIT);
18.
19.    ...

```

```
20. }
```

Les logs au démarrage sont les suivants :

```
1. 05-31 06:23:07.015 32551-32551/exemples.android D/MainActivity: constructor
2. 05-31 06:23:07.041 32551-32551/exemples.android D/MainActivity: afterViews
3. 05-31 06:23:07.050 32551-32551/exemples.android D/PlaceholderFragment: constructor
4. 05-31 06:23:07.053 32551-32551/exemples.android D/PlaceholderFragment: constructor
5. 05-31 06:23:07.053 32551-32551/exemples.android D/PlaceholderFragment: constructor
6. 05-31 06:23:07.053 32551-32551/exemples.android D/PlaceholderFragment: constructor
7. 05-31 06:23:07.053 32551-32551/exemples.android D/PlaceholderFragment: constructor
8. 05-31 06:23:07.278 32551-32551/exemples.android D/MainActivity: getItem[0]
9. 05-31 06:23:07.278 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
10. 05-31 06:23:07.278 32551-32551/exemples.android D/MainActivity: getItem[1]
11. 05-31 06:23:07.278 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
12. 05-31 06:23:07.278 32551-32551/exemples.android D/MainActivity: getItem[2]
13. 05-31 06:23:07.278 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
14. 05-31 06:23:07.278 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=true, initDone=false, updateDone=false
15. 05-31 06:23:07.280 32551-32551/exemples.android D/PlaceholderFragment: afterViews 2 numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=false, updateDone=false
16. 05-31 06:23:07.291 32551-32551/exemples.android D/PlaceholderFragment: afterViews 3 numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=false, updateDone=false
17. 05-31 06:23:07.294 32551-32551/exemples.android D/PlaceholderFragment: afterViews 1 numVisit=1, afterViewsDone=true,
   isVisibleToUser=true, initDone=false, updateDone=false
18. 05-31 06:23:07.295 32551-32551/exemples.android D/PlaceholderFragment: onResume 1 : numVisit=1, afterViewsDone=true,
   isVisibleToUser=true, initDone=true, updateDone=false
19. 05-31 06:23:07.295 32551-32551/exemples.android D/PlaceholderFragment: onResume 2 : numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=true, updateDone=false
20. 05-31 06:23:07.295 32551-32551/exemples.android D/PlaceholderFragment: onResume 3 : numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=true, updateDone=false
21. 05-31 06:23:07.798 32551-32551/exemples.android D/menu: création menu en cours
```

- lignes 8, 10, 12 : le conteneur de fragments réclame tous les fragments adjacents au fragment 1 ;
- lignes 9, 11, 13 : la méthode [setUserVisibleHint] de ces fragments est exécutée avec [visibleToUser] à *false* ;
- ligne 14 : la méthode [setUserVisibleHint] du fragment 1 est exécutée avec [visibleToUser] à *true* ;
- lignes 15-17 : la méthode [afterViews] des 3 segments adjacents est appelée. On voit donc ici un cas où cette méthode est appelée après qu'un fragment soit devenu visible (le fragment 1 ligne 14) ;
- lignes 18-20 : la méthode [onResume] des 3 segments adjacents est appelée ;

On passe de l'onglet 1 à l'onglet 2 :

```
1. 05-31 06:52:36.132 32551-32551/exemples.android D/MainActivity: getItem[3]
2. 05-31 06:52:36.132 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
3. 05-31 06:52:36.132 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=1,
   afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
4. 05-31 06:52:36.132 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=0,
   afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
5. 05-31 06:52:36.134 32551-32551/exemples.android D/PlaceholderFragment: afterViews 4 numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=false, updateDone=false
6. 05-31 06:52:36.134 32551-32551/exemples.android D/PlaceholderFragment: onResume 4 : numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=true, updateDone=false
```

- parce que l'adjacence de fragments est décalée d'une position vers la droite, le fragment 4 est réclamé par le conteneur de fragments ;
- ligne 2 : la méthode [setUserVisibleHint] du fragment 4 est appelée avec [visibleToUser] à *false* ;
- ligne 3 : la méthode [setUserVisibleHint] du fragment 1 est appelée avec [visibleToUser] à *false*. En effet, le fragment 1 est désormais caché ;
- ligne 4 : la méthode [setUserVisibleHint] du fragment 2 est appelée avec [visibleToUser] à *true*. Le fragment 2 est désormais visible ;
- lignes 5-6 : le cycle de vie du fragment 4 se poursuit ;

On passe de l'onglet 2 à l'onglet 3 :

```
1. 05-31 06:58:16.228 32551-32551/exemples.android D/MainActivity: getItem[4]
2. 05-31 06:58:16.228 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 5 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
3. 05-31 06:58:16.228 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=1,
   afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
4. 05-31 06:58:16.228 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=0,
   afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
5. 05-31 06:58:16.229 32551-32551/exemples.android D/PlaceholderFragment: afterViews 5 numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=false, updateDone=false
```

```
6. 05-31 06:58:16.229 32551-32551/exemples.android D/PlaceholderFragment: onResume 5 : numVisit=0, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=false
```

- parce que l'adjacence de fragments est décalée d'une position vers la droite, le fragment 5 est réclamé par le conteneur de fragments ;
- ligne 2 : la méthode [setUserVisibleHint] du fragment 5 est appelée avec [visibleToUser] à *false* ;
- ligne 3 : la méthode [setUserVisibleHint] du fragment 2 est appelée avec [visibleToUser] à *false*. En effet, le fragment 2 est désormais caché ;
- ligne 4 : la méthode [setUserVisibleHint] du fragment 3 est appelée avec [visibleToUser] à *true*. Le fragment 3 est désormais visible ;
- lignes 5-6 : le cycle de vie du fragment 5 se poursuit ;

On passe de l'onglet 3 à l'onglet 4 :

```
1. 05-31 07:00:17.762 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:00:17.762 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=0, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
3. 05-31 07:00:17.762 32551-32551/exemples.android D/PlaceholderFragment: onDestroyView 1 : numVisit=1, afterViewsDone=false, isVisibleToUser=false, initDone=true, updateDone=false
```

- ligne 1 : le fragment 3 est désormais caché ;
- ligne 2 : le fragment 4 est désormais visible. On notera qu'il n'y a pas exécution du cycle de vie du fragment 4. Celui-ci a déjà été fait deux étapes précédemment ;
- ligne 3 : le fragment 1 sort de l'adjacence du fragment 4 affiché. Sa méthode [onDestroyView] est exécutée. La prochaine fois qu'il sera affiché, son cycle de vue [onCreateView, afterViews, onResume] sera réexécuté ;

On passe de l'onglet 4 à l'onglet 5 :

```
1. 05-31 07:04:19.004 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:04:19.004 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 5 : numVisit=0, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
3. 05-31 07:04:19.004 32551-32551/exemples.android D/PlaceholderFragment: onDestroyView 2 : numVisit=1, afterViewsDone=false, isVisibleToUser=false, initDone=true, updateDone=false
```

- ligne 1 : le fragment 4 est désormais caché ;
- ligne 2 : le fragment 5 est désormais visible. On notera qu'il n'y a pas exécution du cycle de vie du fragment 5. Celui-ci a déjà été fait 2 étapes précédemment ;
- ligne 3 : le fragment 2 sort de l'adjacence du fragment 5 affiché. Sa méthode [onDestroyView] est exécutée ;

On passe de l'onglet 5 à l'onglet 1 :

```
1. 05-31 07:06:17.246 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=1, afterViewsDone=false, isVisibleToUser=false, initDone=true, updateDone=false
2. 05-31 07:06:17.246 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=1, afterViewsDone=false, isVisibleToUser=false, initDone=true, updateDone=false
3. 05-31 07:06:17.246 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 5 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
4. 05-31 07:06:17.246 32551-32551/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=1, afterViewsDone=false, isVisibleToUser=true, initDone=true, updateDone=false
5. 05-31 07:06:17.246 32551-32551/exemples.android D/PlaceholderFragment: afterViews 1 numVisit=2, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
6. 05-31 07:06:17.246 32551-32551/exemples.android D/PlaceholderFragment: onResume 1 : numVisit=2, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
7. 05-31 07:06:17.247 32551-32551/exemples.android D/PlaceholderFragment: update 1 : numVisit=2, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
8. 05-31 07:06:17.247 32551-32551/exemples.android D/PlaceholderFragment: afterViews 2 numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=false
9. 05-31 07:06:17.247 32551-32551/exemples.android D/PlaceholderFragment: onResume 2 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=false
10. 05-31 07:06:17.819 32551-32551/exemples.android D/PlaceholderFragment: onDestroyView 4 : numVisit=1, afterViewsDone=false, isVisibleToUser=false, initDone=true, updateDone=false
11. 05-31 07:06:17.819 32551-32551/exemples.android D/PlaceholderFragment: onDestroyView 5 : numVisit=1, afterViewsDone=false, isVisibleToUser=false, initDone=true, updateDone=false
```

- lignes 1, 4, 5, 6 : le cycle de vie du fragment 1 est réexécuté. En effet, il avait perdu la connexion avec sa vue ;
- lignes 2, 5, 8, 9 : pour la même raison le cycle de vie du fragment 2 est réexécuté ;
- lignes 10-11 : les fragments 4 et 5 sortent de l'adjacence du fragment affiché ;
- ligne 7 : le fragment 1 est mis à jour ;

Exemple-07

Hello World from section: 1 update(1)

Les logs n'ont jamais montré que les méthodes [setUserVisibleHint] et [onResume] tentaient toutes les deux de mettre à jour le fragment. C'est soit l'une, soit l'autre. Le lecteur est invité à faire d'autres tests et de suivre les logs pour bien comprendre la notion d'adjacence et de cycle de vie des fragments.

Maintenant, mettons une adjacence totale et faisons les mêmes tests.

Dans [MainActivity] :

```
1. // nombre de fragments
2. private final int FRAGMENTS_COUNT = 5;
3. // adjacence des fragments
4. private final int OFF_SCREEN_PAGE_LIMIT = FRAGMENTS_COUNT - 1;
```

Les logs au démarrage sont les suivants :

```
1. 05-31 07:34:44.717 28908-28908/exemples.android D/MainActivity: constructor
2. 05-31 07:34:44.844 28908-28908/exemples.android D/MainActivity: afterViews
3. 05-31 07:34:44.887 28908-28908/exemples.android D/PlaceholderFragment: constructor
4. 05-31 07:34:44.887 28908-28908/exemples.android D/PlaceholderFragment: constructor
5. 05-31 07:34:44.887 28908-28908/exemples.android D/PlaceholderFragment: constructor
6. 05-31 07:34:44.887 28908-28908/exemples.android D/PlaceholderFragment: constructor
7. 05-31 07:34:44.887 28908-28908/exemples.android D/PlaceholderFragment: constructor
8. 05-31 07:34:45.201 28908-28908/exemples.android D/MainActivity: getItem[0]
9. 05-31 07:34:45.201 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
10. 05-31 07:34:45.201 28908-28908/exemples.android D/MainActivity: getItem[1]
11. 05-31 07:34:45.204 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
12. 05-31 07:34:45.204 28908-28908/exemples.android D/MainActivity: getItem[2]
13. 05-31 07:34:45.204 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
14. 05-31 07:34:45.204 28908-28908/exemples.android D/MainActivity: getItem[3]
15. 05-31 07:34:45.204 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
16. 05-31 07:34:45.205 28908-28908/exemples.android D/MainActivity: getItem[4]
17. 05-31 07:34:45.205 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 5 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
18. 05-31 07:34:45.205 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
afterViewsDone=false, isVisibleToUser=true, initDone=false, updateDone=false
19. 05-31 07:34:45.207 28908-28908/exemples.android D/PlaceholderFragment: afterViews 2 numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=false, updateDone=false
20. 05-31 07:34:45.208 28908-28908/exemples.android D/PlaceholderFragment: afterViews 3 numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=false, updateDone=false
21. 05-31 07:34:45.208 28908-28908/exemples.android D/PlaceholderFragment: afterViews 4 numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=false, updateDone=false
22. 05-31 07:34:45.209 28908-28908/exemples.android D/PlaceholderFragment: afterViews 5 numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=false, updateDone=false
23. 05-31 07:34:45.210 28908-28908/exemples.android D/PlaceholderFragment: afterViews 1 numVisit=1, afterViewsDone=true,
isVisibleToUser=true, initDone=false, updateDone=false
24. 05-31 07:34:45.210 28908-28908/exemples.android D/PlaceholderFragment: onResume 1 : numVisit=1, afterViewsDone=true,
isVisibleToUser=true, initDone=true, updateDone=false
25. 05-31 07:34:45.210 28908-28908/exemples.android D/PlaceholderFragment: onResume 2 : numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=true, updateDone=false
26. 05-31 07:34:45.210 28908-28908/exemples.android D/PlaceholderFragment: onResume 3 : numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=true, updateDone=false
27. 05-31 07:34:45.210 28908-28908/exemples.android D/PlaceholderFragment: onResume 4 : numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=true, updateDone=false
28. 05-31 07:34:45.210 28908-28908/exemples.android D/PlaceholderFragment: onResume 5 : numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=true, updateDone=false
29. 05-31 07:34:46.548 28908-28908/exemples.android D/menu: création menu en cours
```

- les logs montrent que le cycle de vie des 5 fragment est exécuté ;
- le fragment 1 est affiché ligne 18 ;

On passe de l'onglet 1 à l'onglet 2 :

```
1. 05-31 07:38:27.780 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:38:27.780 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=0, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
```

- ligne 1 : le fragment 1 est caché ;
- ligne 2 : le fragment 2 est affiché ;

On passe de l'onglet 2 à l'onglet 3 :

```
1. 05-31 07:39:33.059 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:39:33.059 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=0, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
```

- ligne 1 : le fragment 2 est caché ;
- ligne 2 : le fragment 3 est affiché ;

On passe de l'onglet 3 à l'onglet 4 :

```
1. 05-31 07:40:30.362 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:40:30.362 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=0, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
```

- ligne 1 : le fragment 3 est caché ;
- ligne 2 : le fragment 4 est affiché ;

On passe de l'onglet 4 à l'onglet 5 :

```
1. 05-31 07:41:23.479 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:41:23.479 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 5 : numVisit=0, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
```

- ligne 1 : le fragment 4 est caché ;
- ligne 2 : le fragment 5 est affiché ;

On passe de l'onglet 5 à l'onglet 1 :

```
1. 05-31 07:42:22.549 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 5 : numVisit=1, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:42:22.549 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=1, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
3. 05-31 07:42:22.549 28908-28908/exemples.android D/PlaceholderFragment: update 1 : numVisit=2, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
```

- ligne 1 : le fragment 5 est caché ;
- ligne 2 : le fragment 1 est affiché ;
- ligne 3 : le fragment 1 est mis à jour ;

On passe de l'onglet 1 à l'onglet 4 :

```
1. 05-31 07:44:13.129 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=2, afterViewsDone=true, isVisibleToUser=false, initDone=true, updateDone=true
2. 05-31 07:44:13.129 28908-28908/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=1, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
3. 05-31 07:44:13.129 28908-28908/exemples.android D/PlaceholderFragment: update 4 : numVisit=2, afterViewsDone=true, isVisibleToUser=true, initDone=true, updateDone=false
```

- ligne 1 : le fragment 1 est caché ;
- ligne 2 : le fragment 4 est affiché ;
- ligne 3 : le fragment 4 est mis à jour ;

On remarque, qu'avec l'adjacence totale, le comportement des fragments est beaucoup plus prévisible.

Maintenant, mettons une adjacence nulle et voyons ce qui se passe. La classe [MainActivity] évolue comme suit :

```
1. // nombre de fragments
```

```

2.     private final int FRAGMENTS_COUNT = 5;
3.     // adjacence des fragments
4.     private final int OFF_SCREEN_PAGE_LIMIT = 0;

```

Les logs au démarrage sont les suivants :

```

1. 06-01 03:11:52.068 5679-5679/exemples.android D/MainActivity: constructor
2. 06-01 03:11:52.353 5679-5679/exemples.android D/MainActivity: afterViews
3. 06-01 03:11:52.433 5679-5679/exemples.android D/PlaceholderFragment: constructor
4. 06-01 03:11:52.433 5679-5679/exemples.android D/PlaceholderFragment: constructor
5. 06-01 03:11:52.434 5679-5679/exemples.android D/PlaceholderFragment: constructor
6. 06-01 03:11:52.434 5679-5679/exemples.android D/PlaceholderFragment: constructor
7. 06-01 03:11:52.434 5679-5679/exemples.android D/PlaceholderFragment: constructor
8. 06-01 03:11:52.566 5679-5679/exemples.android D/MainActivity: getItem[0]
9. 06-01 03:11:52.566 5679-5679/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
10. 06-01 03:11:52.566 5679-5679/exemples.android D/MainActivity: getItem[1]
11. 06-01 03:11:52.566 5679-5679/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false
12. 06-01 03:11:52.566 5679-5679/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
   afterViewsDone=false, isVisibleToUser=true, initDone=false, updateDone=false
13. 06-01 03:11:52.571 5679-5679/exemples.android D/PlaceholderFragment: afterViews 2 numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=false, updateDone=false
14. 06-01 03:11:52.574 5679-5679/exemples.android D/PlaceholderFragment: afterViews 1 numVisit=1, afterViewsDone=true,
   isVisibleToUser=true, initDone=false, updateDone=false
15. 06-01 03:11:52.574 5679-5679/exemples.android D/PlaceholderFragment: onResume 1 : numVisit=1, afterViewsDone=true,
   isVisibleToUser=true, initDone=true, updateDone=false
16. 06-01 03:11:52.574 5679-5679/exemples.android D/PlaceholderFragment: onResume 2 : numVisit=0, afterViewsDone=true,
   isVisibleToUser=false, initDone=true, updateDone=false
17. 06-01 03:11:54.597 5679-5679/exemples.android D/menu: création menu en cours

```

- lignes 8 et 10, on voit que le conteneur de fragments a réclamé 2 fragments, les n°s 1 et 2. Tout se passe donc comme s'il y avait une adjacence de 1. L'adjacence de 0 a donc été ignorée.

1.9.3 Communication inter-fragments

Dans l'architecture précédente, nous avons une activité et n fragments. L'utilisateur interagit avec les différents fragments. Ces interactions modifient l'état de l'application. On appelle ici état de l'application, l'ensemble des informations qu'elle mémorise tout au long de sa vie. Le problème suivant se pose alors :

- lorsque l'utilisateur interagit avec le fragment i, l'application passe d'un état E1 à un état E2 ;
- une action de l'utilisateur sur le fragment i fait afficher le fragment j ;
- comment mettre à jour le fragment j avec l'état actuel E2 de l'application ;

Des exemples précédents, nous savons comment mettre à jour le fragment j. Mais où trouver l'état E2 de l'application pour le mettre à jour ?

Il y a différentes solutions à ce problème. Nous en avons vu une : le fragment i peut transmettre l'état E2 de l'application au fragment j via des arguments. Nous avons rencontré cette méthode dans la classe [MainActivity] lors de la création des fragments :

```

1.     for (int i = 0; i < fragments.length; i++) {
2.         // on crée un fragment
3.         fragments[i] = new PlaceholderFragment_();
4.         // on peut passer des arguments au fragment
5.         Bundle args = new Bundle();
6.         args.putInt(ARG_SECTION_NUMBER, i + 1);
7.         fragments[i].setArguments(args);
8.     }

```

Cette solution n'est pas immédiatement utilisable ici. En effet, lorsque l'utilisateur clique sur l'onget j qui va faire apparaître le fragment j, notre code n'est pas appelé. C'est uniquement du code système qui s'exécute. Nous verrons dans un prochain projet comment intercepter le clic sur un onglet, mais pour l'instant nous allons adopter une autre voie.

Nous avons parlé d'état de l'application : l'ensemble des données gérées par l'application au fil du temps. Ici l'application est constituée d'une activité et de n fragments tous instanciés une unique fois au démarrage de l'application et dont la durée de vie est celle de l'application. Donc chacun de ces éléments ou plusieurs ensemble peuvent être candidats pour stocker l'état de l'application. Chaque fragment a accès, par la méthode [Fragment.getActivity()], à l'activité qui l'a créé . Tous les fragments ayant accès à l'activité, il semble naturel de stocker l'état de l'application dans celle-ci.

Cependant le résultat de la méthode [Fragment.getActivity()] dépend du moment où elle est appelée dans le cycle de vie. Nous illustrons ce point en ajoutant quelques logs dans la classe [PlaceholderFragment] :

```

1.     // update fragment
2.     public void update() {

```

```

3.     Log.d("PlaceholderFragment", String.format("update %s : %s", getArguments().getInt(ARG_SECTION_NUMBER), getInfos()));
4.     // le travail à faire dépend du n° de la visite
5.     if (numVisit > 1) {
6.         // log
7.         Log.d("PlaceholderFragment", String.format("update %s : %s", getArguments().getInt(ARG_SECTION_NUMBER),
8.             getInfos()));
8.         // texte modifié
9.         textViewInfo.setText(String.format("%s update(%s)", text, (numVisit - 1)));
10.    }
11. }
12.
13. // infos locales pour logs
14. private String getInfos() {
15.     return String.format("numVisit=%s, afterViewsDone=%s, isVisibleToUser=%s, initDone=%s, updateDone=%s,
16.     getActivity()==null:%s",
17.         numVisit, afterViewsDone, isVisibleToUser, initDone, updateDone, getActivity() == null);

```

- lignes 14-16 : la méthode [getInfos] affiche partie de l'état de l'application ;

Nous lançons l'application avec une adjacence de fragments de 2. Les logs au démarrage de l'application :

```

1. 06-01 03:26:13.769 10931-10931/exemples.android D/MainActivity: constructor
2. 06-01 03:26:13.856 10931-10931/exemples.android D/MainActivity: afterViews
3. 06-01 03:26:13.864 10931-10931/exemples.android D/PlaceholderFragment: constructor
4. 06-01 03:26:13.864 10931-10931/exemples.android D/PlaceholderFragment: constructor
5. 06-01 03:26:13.864 10931-10931/exemples.android D/PlaceholderFragment: constructor
6. 06-01 03:26:13.864 10931-10931/exemples.android D/PlaceholderFragment: constructor
7. 06-01 03:26:13.864 10931-10931/exemples.android D/PlaceholderFragment: constructor
8. 06-01 03:26:14.535 10931-10931/exemples.android D>MainActivity: getItem[0]
9. 06-01 03:26:14.538 10931-10931/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true
10. 06-01 03:26:14.538 10931-10931/exemples.android D>MainActivity: getItem[1]
11. 06-01 03:26:14.538 10931-10931/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true
12. 06-01 03:26:14.538 10931-10931/exemples.android D>MainActivity: getItem[2]
13. 06-01 03:26:14.538 10931-10931/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true
14. 06-01 03:26:14.538 10931-10931/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
afterViewsDone=false, isVisibleToUser=true, initDone=false, updateDone=false, getActivity()==null:false
15. 06-01 03:26:14.541 10931-10931/exemples.android D/PlaceholderFragment: afterViews 2 numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:false
16. 06-01 03:26:14.545 10931-10931/exemples.android D/PlaceholderFragment: afterViews 3 numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:false
17. 06-01 03:26:14.547 10931-10931/exemples.android D/PlaceholderFragment: afterViews 1 numVisit=1, afterViewsDone=true,
isVisibleToUser=true, initDone=false, updateDone=false, getActivity()==null:false
18. 06-01 03:26:14.547 10931-10931/exemples.android D/PlaceholderFragment: onResume 1 : numVisit=1, afterViewsDone=true,
isVisibleToUser=true, initDone=true, updateDone=false, getActivity()==null:false
19. 06-01 03:26:14.547 10931-10931/exemples.android D/PlaceholderFragment: update 1 : numVisit=1, afterViewsDone=true,
isVisibleToUser=true, initDone=true, updateDone=false, getActivity()==null:false
20. 06-01 03:26:14.547 10931-10931/exemples.android D/PlaceholderFragment: onResume 2 : numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=true, updateDone=false, getActivity()==null:false
21. 06-01 03:26:14.547 10931-10931/exemples.android D/PlaceholderFragment: onResume 3 : numVisit=0, afterViewsDone=true,
isVisibleToUser=false, initDone=true, updateDone=false, getActivity()==null:false
22. 06-01 03:26:15.967 10931-10931/exemples.android D/menu: création menu en cours

```

- lignes 9, 10, 13, 14 : on voit que dans les méthodes [setUserVisibleHint], on a [getActivity()==null] si le fragment n'est pas encore visible (isVisibleToUser==false) ;
- ligne 19 : on voit que lorsque le flux d'exécution arrive à la méthode [update] du fragment 1, la méthode [getActivity] rend bien l'activité ;

Lorsqu'on met l'adjacence de fragments à 4 (adjacence totale), les logs sont les suivants :

```

1. 06-01 03:35:23.553 2814-2814/exemples.android D/MainActivity: constructor
2. 06-01 03:35:23.751 2814-2819/exemples.android I/art: Ignoring second debugger -- accepting and dropping
3. 06-01 03:35:23.900 2814-2814/exemples.android D/MainActivity: afterViews
4. 06-01 03:35:23.991 2814-2814/exemples.android D/PlaceholderFragment: constructor
5. 06-01 03:35:23.991 2814-2814/exemples.android D/PlaceholderFragment: constructor
6. 06-01 03:35:23.991 2814-2814/exemples.android D/PlaceholderFragment: constructor
7. 06-01 03:35:23.991 2814-2814/exemples.android D/PlaceholderFragment: constructor
8. 06-01 03:35:24.002 2814-2814/exemples.android D/PlaceholderFragment: constructor
9. 06-01 03:35:24.207 2814-2814/exemples.android D>MainActivity: getItem[0]
10. 06-01 03:35:24.207 2814-2814/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true
11. 06-01 03:35:24.207 2814-2814/exemples.android D>MainActivity: getItem[1]
12. 06-01 03:35:24.207 2814-2814/exemples.android D/PlaceholderFragment: setUserVisibleHint 2 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true
13. 06-01 03:35:24.207 2814-2814/exemples.android D>MainActivity: getItem[2]
14. 06-01 03:35:24.207 2814-2814/exemples.android D/PlaceholderFragment: setUserVisibleHint 3 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true
15. 06-01 03:35:24.207 2814-2814/exemples.android D>MainActivity: getItem[3]
16. 06-01 03:35:24.207 2814-2814/exemples.android D/PlaceholderFragment: setUserVisibleHint 4 : numVisit=0,
afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true

```

```
17. 06-01 03:35:24.207 2814-2814/exemples.android D/MainActivity: getItem[4]
18. 06-01 03:35:24.207 2814-2814/exemples.android D/PlaceholderFragment: setUserVisibleHint 5 : numVisit=0,
    afterViewsDone=false, isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:true
19. 06-01 03:35:24.207 2814-2814/exemples.android D/PlaceholderFragment: setUserVisibleHint 1 : numVisit=0,
    afterViewsDone=false, isVisibleToUser=true, initDone=false, updateDone=false, getActivity()==null:false
20. 06-01 03:35:24.210 2814-2814/exemples.android D/PlaceholderFragment: afterViews 2 numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:false
21. 06-01 03:35:24.211 2814-2814/exemples.android D/PlaceholderFragment: afterViews 3 numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:false
22. 06-01 03:35:24.214 2814-2814/exemples.android D/PlaceholderFragment: afterViews 4 numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:false
23. 06-01 03:35:24.215 2814-2814/exemples.android D/PlaceholderFragment: afterViews 5 numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=false, updateDone=false, getActivity()==null:false
24. 06-01 03:35:24.215 2814-2814/exemples.android D/PlaceholderFragment: afterViews 1 numVisit=1, afterViewsDone=true,
    isVisibleToUser=true, initDone=false, updateDone=false, getActivity()==null:false
25. 06-01 03:35:24.215 2814-2814/exemples.android D/PlaceholderFragment: onResume 1 : numVisit=1, afterViewsDone=true,
    isVisibleToUser=true, initDone=true, updateDone=false, getActivity()==null:false
26. 06-01 03:35:24.215 2814-2814/exemples.android D/PlaceholderFragment: update 1 : numVisit=1, afterViewsDone=true,
    isVisibleToUser=true, initDone=true, updateDone=false, getActivity()==null:false
27. 06-01 03:35:24.216 2814-2814/exemples.android D/PlaceholderFragment: onResume 2 : numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=true, updateDone=false, getActivity()==null:false
28. 06-01 03:35:24.216 2814-2814/exemples.android D/PlaceholderFragment: onResume 3 : numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=true, updateDone=false, getActivity()==null:false
29. 06-01 03:35:24.216 2814-2814/exemples.android D/PlaceholderFragment: onResume 4 : numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=true, updateDone=false, getActivity()==null:false
30. 06-01 03:35:24.216 2814-2814/exemples.android D/PlaceholderFragment: onResume 5 : numVisit=0, afterViewsDone=true,
    isVisibleToUser=false, initDone=true, updateDone=false, getActivity()==null:false
31. 06-01 03:35:26.602 2814-2814/exemples.android D/menu: création menu en cours
```

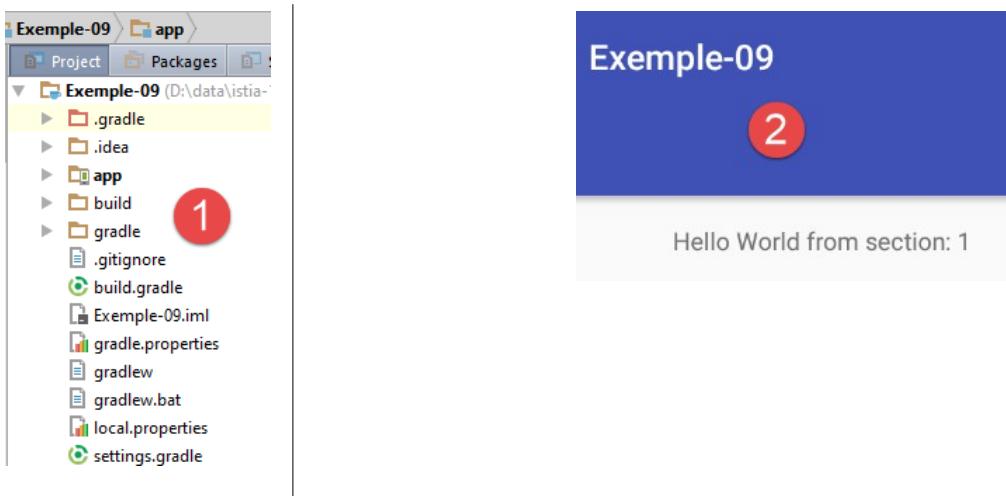
On a les mêmes résultats. On en déduit que dès que le fragment est visible la méthode [getActivity] rend l'activité du fragment. On remarque également que lorsque l'exécution atteint la méthode [update] du fragment qui va s'afficher, la méthode [getActivity] rend bien une valeur.

Pour illustrer la communication inter-fragments nous construisons un nouveau projet.

1.10 Exemple-09 : communication inter-fragments, swipe et scrolling

1.10.1 Cration du projet

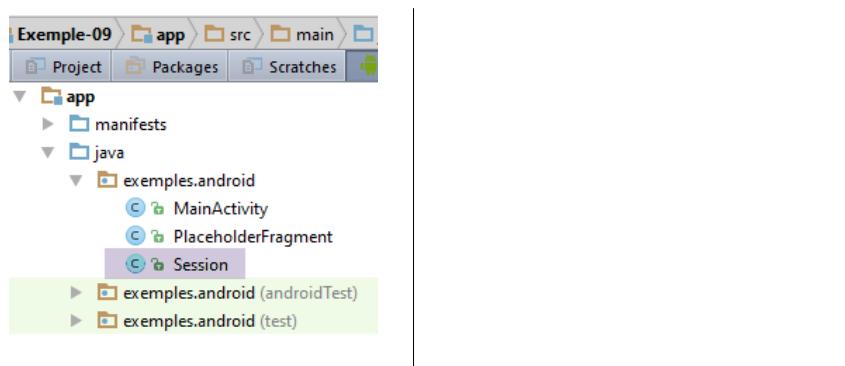
On duplique le projet [Exemple-07] dans [Exemple-08]. Pour cela, on suivra la procure dcrite pour dupliquer [Exemple-02] dans [Exemple-03] au paragraphe 1.4, page 36 .



1.10.2 La session

Dans ce nouveau projet, nous voulons que les fragments affichent le nombre total de fragments affichs par l'utilisateur. Il faut ici entretenir un compteur qui soit accessible  tous les fragments. Nous appellerons **session** l'objet encapsulant les donnes partages par les fragments. Cette terminologie vient du dveloppement web, o on met dans une session les donnes  partager par diffentes vues demandes par un mme utilisateur. Le fait d'encapsuler les informations partages par les diffents fragments dans un mme objet rend les choses plus lisibles.

La classe [Session] sera la suivante :



```
1. package exemples.android;
2.
3. import org.androidannotations.annotations.EBean;
4.
5. @EBean(scope = EBean.Scope.Singleton)
6. public class Session {
7.     // nombre de fragments visits
8.     private int numVisit;
9.
10.    // getters et setters
11.
12.    public int getNumVisit() {
13.        return numVisit;
14.    }
```

```

15.
16.     public void setNumVisit(int numVisit) {
17.         this.numVisit = numVisit;
18.     }
19. }
```

- ligne 8 : la session encapsulera le nombre de fragments visités ;
- ligne 5 : l'annotation [EBean] est une annotation AA. L'attribut [scope] désigne la portée (ou durée de vie) de la classe ainsi annotée. Ici l'attribut [scope = EBean.Scope.Singleton] fait que la classe [Session] est un singleton : elle va être instanciée une fois et une fois seulement au démarrage de l'application. La référence d'une classe annotée [EBean] peut ensuite être injectée dans une autre classe. C'est la notion d'injection de dépendances ;

1.10.3 L'activité [MainActivity]

L'activité [MainActivity] évolue de la façon suivante :

```

1.  @EActivity(R.layout.activity_main)
2.  public class MainActivity extends AppCompatActivity {
3.
4.      ...
5.
6.      // injection session
7.      @Bean(Session.class)
8.      protected Session session;
9.
10.     // nombre de fragments
11.     private final int FRAGMENTS_COUNT = 5;
12.     // adjacence des fragments
13.     private final int OFF_SCREEN_PAGE_LIMIT = 2;
14.
15.     @AfterInject
16.     protected void afterInject(){
17.         Log.d("MainActivity", "afterInject");
18.
19.         // initialisation session
20.         session.setNumVisit(0);
21.     }
22.
23. }
```

- lignes 7-8 : injection de la référence au singleton de la session grâce à l'annotation [@Bean]. Le paramètre de l'annotation est la classe du bean à injecter. Le champ ainsi annoté ne peut pas avoir la portée [private] ;
- ligne 15 : l'annotation [@AfterInject] sert à désigner une méthode à appeler lorsque toutes les injections de la classe auront été faites. Ainsi lorsqu'on entre dans la méthode [afterInject] de la ligne 16, la référence de la ligne 8 a été initialisée ;
- ligne 20 : on met le compteur de visites à zéro ;

1.10.4 Le fragment [PlaceholderFragment]

Le fragment [PlaceholderFragment] évolue de la façon suivante :

```

1.  @EFragment(R.layout.fragment_main)
2.  public class PlaceholderFragment extends Fragment {
3.
4.      ....
5.
6.      // session
7.      protected Session session;
8.
9.      @Override
10.     public void setUserVisibleHint(boolean isVisibleToUser) {
11.         // parent
12.         super.setUserVisibleHint(isVisibleToUser);
13.         // mémoire
14.         this.isVisibleToUser = isVisibleToUser;
15.         // log
16.         Log.d("PlaceholderFragment", String.format("setUserVisibleHint %s : %s", getArguments().getInt(ARG_SECTION_NUMBER),
17.             getInfos()));
18.         // nombre de visites
19.         if (isVisibleToUser) {
20.             // update fragment
21.             if (afterViewsDone && !updateDone) {
22.                 update();
23.                 updateDone = true;
24.             }
25.         } else {
26.             // le fragment va être caché
27.             updateDone = false;
28.         }
29.     }
30.
```

```

29.
30.    // update fragment
31.    public void update() {
32.        // log
33.        Log.d("PlaceholderFragment", String.format("update %s : %s", getArguments().getInt(ARG_SECTION_NUMBER), getInfos()));
34.        // session
35.        if (session == null) {
36.            session = ((MainActivity) getActivity()).getSession();
37.        }
38.        // incrément n° de visite
39.        numVisit = session.getNumVisit();
40.        numVisit++;
41.        session.setNumVisit(numVisit);
42.        // texte modifié
43.        textViewInfo.setText(String.format("%s, visite %s", text, numVisit));
44.    }

```

- ligne 7 : la session ;
- lignes 35-37 : nous savons que lorsqu'on arrive dans la méthode [update], la méthode [getActivity] rend bien l'activité. On en profite pour récupérer la session et la mémoriser localement (ligne 36) ;
- lignes 39-41 : pour incrémenter le n° de la visite, on va chercher celui-ci dans la session. On aurait pu mettre ce code dans la méthode [setUserVisibleHint] à partir de la ligne 19 car on sait qu'alors la méthode [getActivity] rend l'activité. On décide ici de ne pas faire jouer de rôle particulier à cette méthode et de faire migrer le code spécifique à un fragment dans la méthode [update] de celui-ci qui est faite pour cela ;
- ligne 43 : affiche le n° de la visite ;

Lorsqu'on exécute cette application avec 5 fragments, une adjacence de 2 fragments, les premiers logs sont les suivants :

```

1. 05-31 08:38:47.305 20114-20114/exemples.android D/MainActivity: constructor
2. 05-31 08:38:47.307 20114-20114/exemples.android D/MainActivity: afterInject
3. 05-31 08:38:47.351 20114-20114/exemples.android D/MainActivity: afterViews
4. 05-31 08:38:47.354 20114-20114/exemples.android D/PlaceholderFragment: constructor
5. 05-31 08:38:47.354 20114-20114/exemples.android D/PlaceholderFragment: constructor
6. 05-31 08:38:47.354 20114-20114/exemples.android D/PlaceholderFragment: constructor
7. 05-31 08:38:47.354 20114-20114/exemples.android D/PlaceholderFragment: constructor
8. 05-31 08:38:47.354 20114-20114/exemples.android D/PlaceholderFragment: constructor
9. ...

```

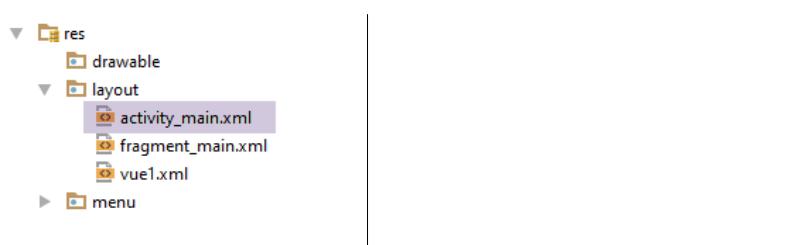
- lignes 2-3 : on voit que la méthode [afterInject] de l'activité est exécutée avant sa méthode [afterViews] ;

Le lecteur est invité à tester cette nouvelle application.

1.10.5 Désactiver le Swipe ou Balayage

Dans l'application précédente, lorsqu'on balaie l'émulateur Android avec la souris vers la gauche ou la droite, la vue courante laisse alors place à la vue de droite ou de gauche selon les cas. Ce comportement par défaut n'est pas toujours souhaitable. Nous allons apprendre à désactiver le balayage des vues (swipe).

Revenons sur la vue XML principale [activity_main] :



Dans le code XML de la vue on trouve celui du conteneur de fragments :

```

1. <android.support.v4.view.ViewPager
2.     android:id="@+id/container"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     app:layout_behavior="@string/appbar_scrolling_view_behavior"/>

```

La ligne 1 désigne la classe qui gère les pages de l'activité. On retrouve cette classe dans l'activité [MainActivity] :

```

1. import android.support.v4.view.ViewPager;
2. ...

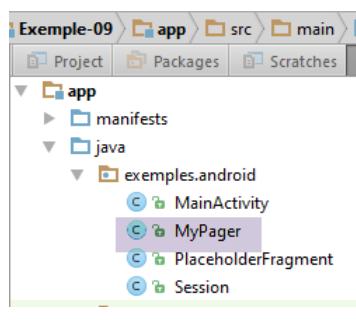
```

```

3.     @EActivity(R.layout.activity_main)
4.     public class MainActivity extends AppCompatActivity {
5.
6.     // le gestionnaire de fragments
7.     private SectionsPagerAdapter mSectionsPagerAdapter;
8.
9.     // le conteneur de fragments
10.    @ViewById(R.id.container)
11.    protected ViewPager mViewPager;
12.
13.    ...

```

Ligne 12, le conteneur de fragments est de type [android.support.v4.view.ViewPager] (ligne 1). Pour désactiver le balayage, on est amené à dériver cette classe de la façon suivante :



```

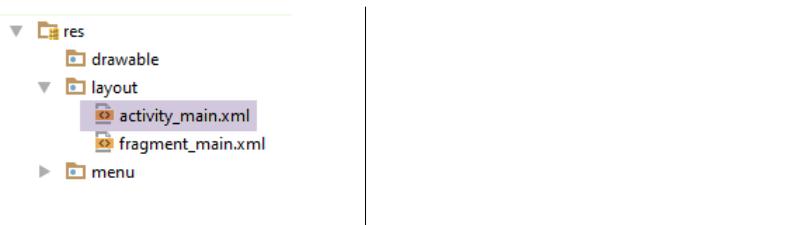
1. package exemples.android;
2.
3. import android.content.Context;
4. import android.support.v4.view.ViewPager;
5. import android.util.AttributeSet;
6. import android.view.MotionEvent;
7.
8. public class MyPagerAdapter extends ViewPager {
9.
10.    // contrôle le swipe
11.    private boolean isSwipeEnabled;
12.
13.    // constructeurs
14.    public MyPagerAdapter(Context context) {
15.        super(context);
16.    }
17.
18.    public MyPagerAdapter(Context context, AttributeSet attrs) {
19.        super(context, attrs);
20.    }
21.
22.    // méthodes à redéfinir pour gérer le swipe
23.    @Override
24.    public boolean onInterceptTouchEvent(MotionEvent event) {
25.        // swipe autorisé ?
26.        if (isSwipeEnabled) {
27.            return super.onInterceptTouchEvent(event);
28.        } else {
29.            return false;
30.        }
31.    }
32.
33.    @Override
34.    public boolean onTouchEvent(MotionEvent event) {
35.        // swipe autorisé ?
36.        if (isSwipeEnabled) {
37.            return super.onTouchEvent(event);
38.        } else {
39.            return false;
40.        }
41.    }
42.
43.    // setter
44.    public void setSwipeEnabled(boolean isSwipeEnabled) {
45.        this.isSwipeEnabled = isSwipeEnabled;
46.    }
47.
48.    }

```

- ligne 8 : la classe [MyPagerAdapter] étend la classe Android [ViewPager] (ligne 4) ;

- sur un balayage de la main, les gestionnaires d'événements des lignes 24 et 34 peuvent être appelés. Elles rendent toutes deux un booléen. Il leur suffit de rendre le booléen [false] pour inhiber le balayage ;
- ligne 11 : le booléen qui sert à indiquer si on accepte ou non le balayage de la main.

Ceci fait, il faut utiliser désormais notre nouveau gestionnaire de pages. Cela se fait dans la vue XML [activity_main.xml] et dans l'activité principale [MainActivity]. Dans [activity_main.xml] on écrit :



```

1. <exemples.android.MyPager
2.     android:id="@+id/container"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     app:layout_behavior="@string/appbar_scrolling_view_behavior"/>

```

Ligne 1, on utilise la nouvelle classe. Dans [MainActivity], le code évolue comme suit :

```

1. package exemples.android;
2.
3. ...
4. @EActivity(R.layout.activity_main)
5. public class MainActivity extends AppCompatActivity {
6.
7.     // le gestionnaire de fragments
8.     private SectionsPagerAdapter mSectionsPagerAdapter;
9.
10.    // le conteneur de fragments
11.    @ViewById(R.id.container)
12.    protected MyPager mViewPager;
13.
14.    @AfterViews
15.    protected void afterViews() {
16.        Log.d("MainActivity", "afterViews");
17.    ...
18.        // le conteneur de fragments est associé au gestionnaire de fragments
19.        // c-à-d que le fragment n° i du conteneur de fragments est le fragment n° i délivré par le gestionnaire de fragments
20.        mViewPager.setAdapter(mSectionsPagerAdapter);
21.
22.        // on inhibe le swipe entre fragments
23.        mViewPager.setSwipeEnabled(false);
24.        // la barre d'onglets est également associée au conteneur de fragments
25.    ...

```

- ligne 12 : le gestionnaire de pages a désormais le type [MyPager] ;
- ligne 23 : on inhibe ou non le balayage de la main.

Testez cette nouvelle version. Inhibez ou non le balayage et constatez la différence de comportement des vues lorsque vous les tirez à droite ou à gauche avec la souris. Dans toutes les applications à venir, le balayage sera inhibé. Nous ne le rappellerons pas.

1.10.6 Désactiver le scrolling entre fragments

Continuons avec une amélioration du gestionnaire d'onglets. Lorsqu'on passe de l'onglet 1 à l'onglet 4, on voit défiler les deux onglets intermédiaires 2 et 3. Cela s'appelle en jargon Android le *smoothScrolling*. Ce comportement peut devenir gênant s'il y a beaucoup d'onglets. Il peut être inhibé en ajoutant le code suivant dans le gestionnaire de fragments [MyPagerAdapter] :

```

1. // contrôle le swipe
2. private boolean isSwipeEnabled;
3. // contrôle le scrolling
4. private boolean isScrollingEnabled;
5.
6. ...
7. // scrolling
8. @Override
9. public void setCurrentItem(int position){
10.     super.setCurrentItem(position, isScrollingEnabled);
11. }
12.
13. // setters

```

```

14. ...
15.
16.     public void setScrollingEnabled(boolean scrollingEnabled) {
17.         isScrollingEnabled = scrollingEnabled;
18.     }

```

Parce que le gestionnaire d'onglets a été associé au gestionnaire de fragments [MyPagerAdapter], lorsqu'on clique sur l'onglet n° i, le fragment n° i est affiché par le conteneurs de fragments par la méthode [setCurrentItem] ci-dessus (ligne 9). [position] est le n° du fragment à afficher ;

- ligne 10 : on fait appel à la méthode [setCurrentItem] de la classe parent. Le second argument à [false] demande à ce qu'il y ait une transition immédiate entre l'ancien et le nouveau fragment (pas de scrolling), à [true] qu'il y ait une transition par scrolling. Ici, le second argument est la valeur du champ de la ligne 4, champ que le développeur peut fixer avec la méthode des lignes 16-18 ;

Si on veut désactiver le scrolling, la classe [MainActivity] sera la suivante :

```

1. ...
2.     // offset des fragments
3.     mViewPager.setOffscreenPageLimit(OFF_SCREEN_PAGE_LIMIT);
4.
5.     // on inhibe le swipe entre fragments
6.     mViewPager.setSwipeEnabled(false);
7.
8.     // pas de scrolling
9.     mViewPager.setScrollingEnabled(false);
10. ...

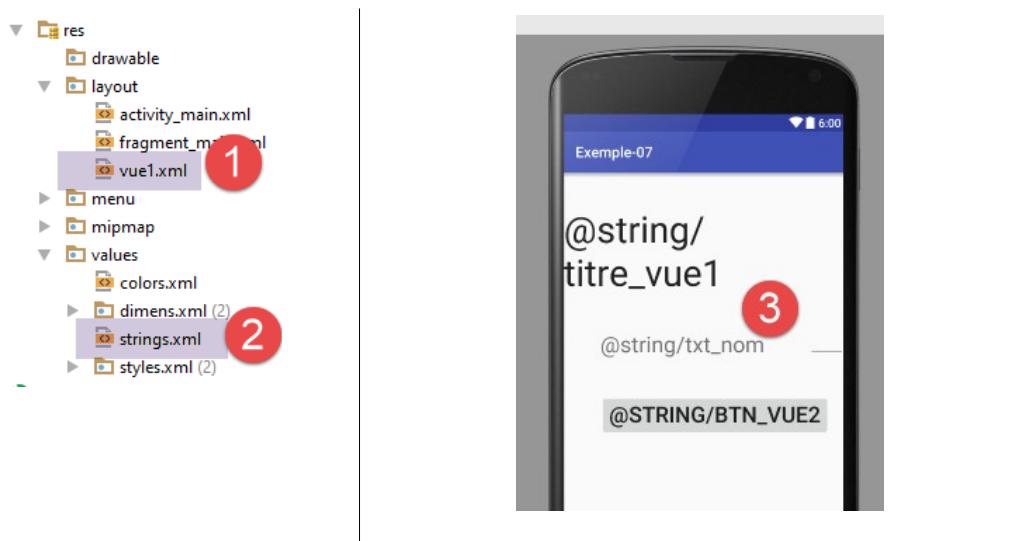
```

Exécutez de nouveau le projet et vérifiez qu'il n'y a plus de scrolling entre les onglets 1 et 4 par exemple. Dans la suite, nous inhiberons toujours le scrolling. Nous ne le rappellerons pas.

1.10.7 Un nouveau fragment

Dans notre exemple, tous les fragments sont du même type [PlaceHolderFragment]. Nous allons maintenant apprendre à créer un nouveau fragment et à l'afficher.

Tout d'abord copions la vue [vue1.xml] du projet [Exemple-04] dans le projet [Exemple-09] [1] :



- en [1], la vue [vue1.xml] ;
- en [3], la vue présente des erreurs provenant de textes absents dans le fichier [res/values/strings.xml] ;

En [2], on rajoute les textes manquants en les prenant dans le fichier [res/values/strings.xml] du projet [Exemple-04]

```

1. <resources>
2.     <string name="app_name">Exemple-07</string>
3.     <string name="action_settings">Settings</string>
4.     <string name="section_format">Hello World from section: %1$d</string>
5.     <!-- vue 1 -->
6.     <string name="titre_vue1">Vue n° 1</string>

```

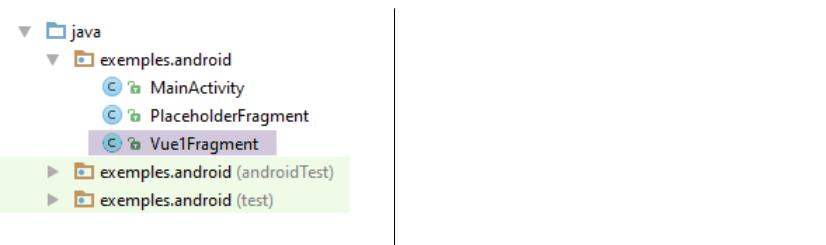
```

7.      <string name="txt_nom">Quel est votre nom ?</string>
8.      <string name="btn_valider">Valider</string>
9.      <string name="btn_vue2">Vue n° 2</string>
10.     </resources>

```

- ci-dessus, on a ajouté les lignes 6-9 ;

Maintenant, nous créons la classe [Vue1Fragment] qui va être le fragment chargé d'afficher la vue [vue1.xml] :



La classe [Vue1Fragment] sera la suivante :

```

1. package exemples.android;
2.
3. import android.support.v4.app.Fragment;
4. import android.widget.EditText;
5. import android.widget.Toast;
6. import org.androidannotations.annotations.Click;
7. import org.androidannotations.annotations.EFragment;
8. import org.androidannotations.annotations.ViewById;
9.
10. @EFragment(R.layout.vue1)
11. public class Vue1Fragment extends Fragment {
12.
13.     // les éléments de l'interface visuelle
14.     @ViewById(R.id.editTextNom)
15.     protected EditText editTextNom;
16.
17.     // gestionnaire d'évt
18.     @Click(R.id.buttonValider)
19.     protected void doValidator() {
20.         // on affiche le nom saisi
21.         Toast.makeText(getActivity(), String.format("Bonjour %s", editTextNom.getText().toString()), Toast.LENGTH_LONG).show();
22.     }
23. }

```

- ligne 10 : l'annotation [<@EFragment>] fait que le fragment utilisé par l'activité sera en réalité la classe [Vue1Fragment]. Il faut s'en souvenir. Le fragment est associé à la vue [vue1.xml] ;
- lignes 14-15 : le composant identifié par [R.id.editTextNom] est injecté dans le champ [editTextNom] de la ligne 15 ;
- lignes 18-20 : la méthode [doValidator] gère l'événement 'click' sur le bouton identifié par [R.id.buttonValider] ;
- ligne 21 : le premier paramètre de [Toast.makeText] est de type [Activity]. La méthode [Fragment.getActivity()] permet d'avoir l'activité dans laquelle se trouve le fragment. Il s'agit de [MainActivity] puisque dans cette architecture, nous n'avons qu'une activité qui affiche différentes vues ou fragments ;

Dans la classe [MainActivity], le gestionnaire de fragments évolue de la façon suivante :

```

1. public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.     // les fragments
4.     private Fragment[] fragments;
5.     // n° de fragment
6.     private static final String ARG_SECTION_NUMBER = "section_number";
7.
8.     // constructeur
9.     public SectionsPagerAdapter(FragmentManager fm) {
10.         // parent
11.         super(fm);
12.         // initialisation du tableau des fragments
13.         fragments = new Fragment[FRAGMENTS_COUNT];
14.         for (int i = 0; i < fragments.length - 1; i++) {
15.             // on crée un fragment
16.             fragments[i] = new PlaceholderFragment_();
17.             // on peut passer des arguments au fragment
18.             Bundle args = new Bundle();
19.             args.putInt(ARG_SECTION_NUMBER, i + 1);
20.             fragments[i].setArguments(args);

```

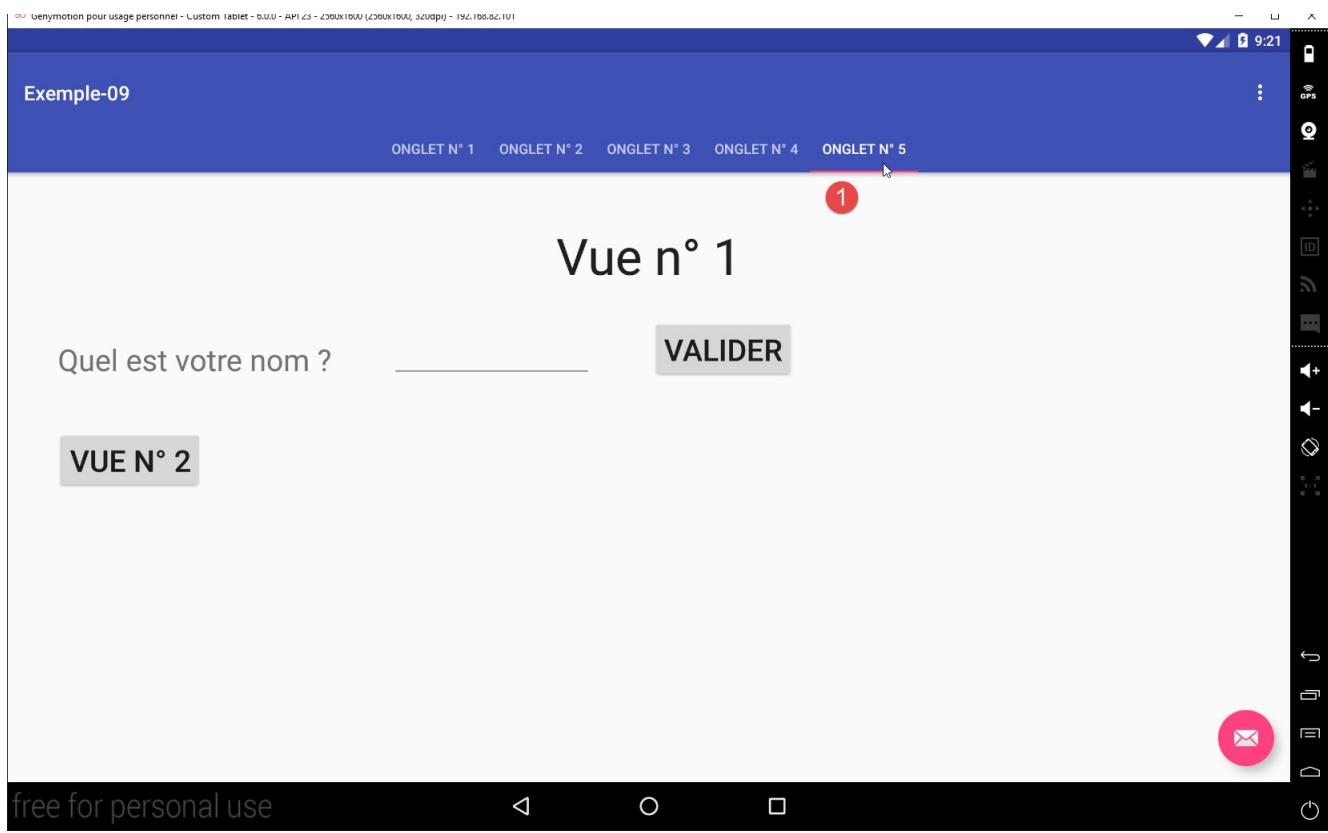
```

21.      }
22.      // un fragment de +
23.      fragments[fragments.length - 1] = new Vue1Fragment_();
24.    }
25.
26.    ...
27.  }

```

- ligne 13 : il y a [FRAGMENTS_COUNT] fragments : [FRAGMENTS_COUNT-1] fragments de type [PlaceholderFragment] (lignes 14-21) et un fragment de type [Vue1Fragment_], ligne 23 (attention à l'underscore) ;

Compilez puis exécutez le projet [Exemple-09]. L'onglet n° 5 doit être différent :



1.10.8 Faire dériver tous les fragments d'une même classe abstraite

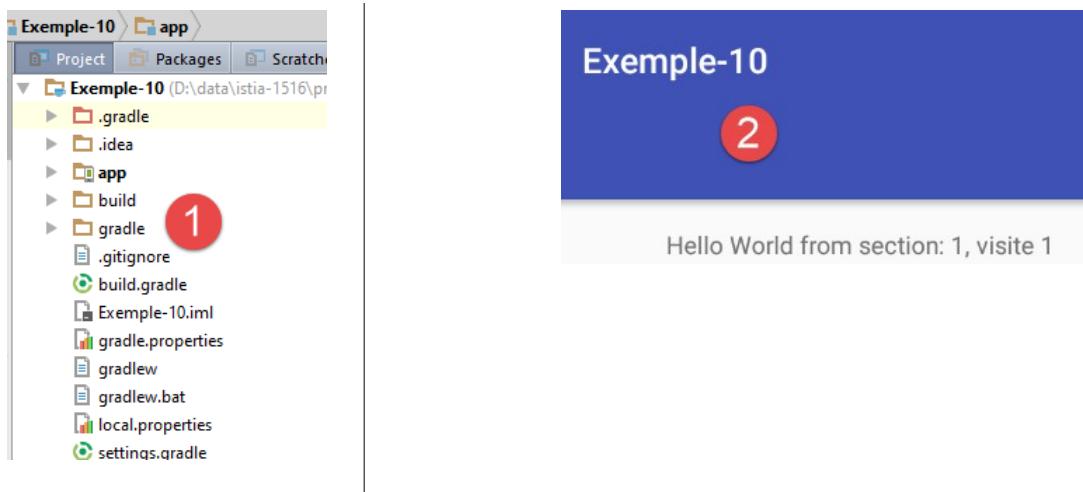
Le nouveau fragment [Vue1Fragment] a besoin lui aussi de se mettre à jour lorsqu'il est affiché. Pour ce faire, on va devoir créer un code similaire à celui créé pour le fragment [PlaceholderFragment]. Pour éviter de se répéter, on va factoriser ce qui peut l'être dans une classe abstraite dont tous les fragments de l'application hériteront.

Pour cela nous créons un nouveau projet.

1.11 Exemple-10 : faire dériver tous les fragments d'une classe abstraite

1.11.1 Crédation du projet

On duplique le projet [Exemple-09] dans [Exemple-10] :

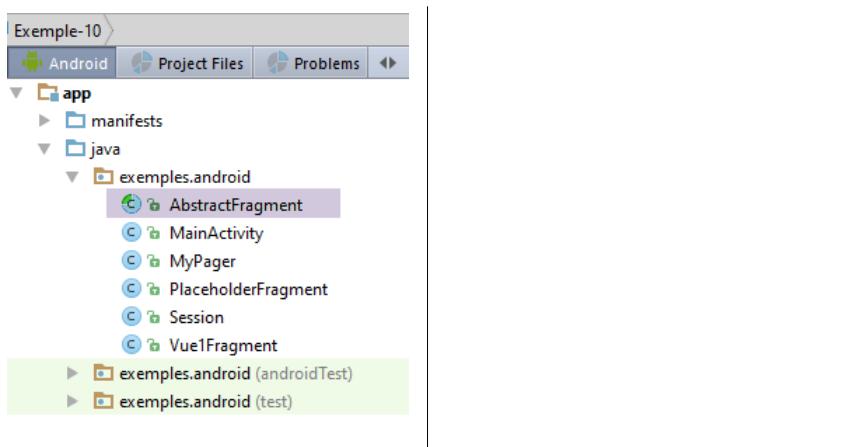


1.11.2 Gestion du mode debug

Nous ajoutons au projet la possibilité d'afficher ou non les logs du mode debug. Pour cela, nous ajoutons une constante statique à la classe [MainActivity] :

```
1. // mode debug
2. public static final boolean IS_DEBUG_ENABLED = false;
```

1.11.3 La classe abstraite parente de tous les fragments



La classe [AbstractFragment] est la suivante :

```
1. package examples.android;
2.
3. import android.app.Activity;
4. import android.support.v4.app.Fragment;
5. import android.util.Log;
6.
7. public abstract class AbstractFragment extends Fragment {
8.
9.     // données privées
10.    private boolean isVisibleToUser = false;
```

```

11.  private boolean updateDone = false;
12.  private String className;
13.
14.  // données accessibles aux classes filles
15.  protected boolean afterViewsDone = false;
16.  protected boolean isDebugEnabled = true;
17.
18.  // activité
19.  protected MainActivity activity;
20.
21.  // session
22.  protected Session session;
23.
24.  // constructeur
25.  public AbstractFragment() {
26.      // init
27.      isDebugEnabled = MainActivity.IS_DEBUG_ENABLED;
28.      className = getClass().getSimpleName();
29.      // log
30.      if (isDebugEnabled) {
31.          Log.d("AbstractFragment", String.format("constructor %s", className));
32.      }
33.  }
34.
35.  @Override
36.  public void setUserVisibleHint(boolean isVisibleToUser) {
37.      // parent
38.      super.setUserVisibleHint(isVisibleToUser);
39.      ...
40.  }
41.
42.  @Override
43.  public void onDestroyView() {
44.      // parent
45.      super.onDestroyView();
46.      ...
47.  }
48.
49.  @Override
50.  public void onResume() {
51.      // parent
52.      super.onResume();
53.      ...
54.  }
55.
56.  // infos locales
57.  protected String getParentInfos() {
58.      return String.format("className=%s, isVisibleToUser=%s, updateDone=%s, afterViewsDone=%s", className, isVisibleToUser,
59.      updateDone, afterViewsDone);
60.  }
61.
62.  // update fragment
63.  protected void update() {
64.      ...
65.      // on demande à la classe fille de se mettre à jour
66.      updateFragment();
67.  }
68.  protected abstract void updateFragment();
69. }

```

- ligne 7 : la classe [AbstractFragment] étend la classe Android [Fragment] ;
- tout fragment doit pouvoir se mettre à jour. C'est pourquoi la classe parent [AbstractFragment] impose la présence à ses classes filles d'une méthode [updateFragment] (ligne 68) qu'elle appelle (ligne 65) ;
- ligne 19 : la classe stockera une référence sur l'activité de l'application ;
- ligne 22 : la classe stockera une référence sur la session où sont rassemblées les données partagées par les fragments et l'activité ;
- lignes 25-33 : le constructeur de la classe abstraite ;
- ligne 27 : création d'une copie de la constante [MainActivity.IS_DEBUG_ENABLED] dans le champ de la ligne 16 ;
- ligne 28 : on mémorise le nom de la classe instanciée, donc le nom d'une classe fille ;
- lignes 15-22 : ces champs ont l'attribut [protected] pour que les classes filles y aient accès. On remarquera que les classes filles ignorent l'existence des booléens [isVisibleToUser] et [updateDone] (lignes 10-11) ;
- ligne 57 : la méthode [getParentInfos] a l'attribut [protected] pour que les classes filles puissent l'appeler ;

Les méthodes [setUserVisibleHint, onDestroyView, onResume] restent analogues à ce qu'elles étaient dans la classe [PlaceholderFragment] du projet précédent :

```

1.  @Override
2.  public void setUserVisibleHint(boolean isVisibleToUser) {
3.      // parent

```

```

4.     super.setUserVisibleHint(isVisibleToUser);
5.     // mémoire
6.     this.isVisibleToUser = isVisibleToUser;
7.     // log
8.     if (isDebugEnabled) {
9.         Log.d("AbstractFragment", String.format("setUserVisibleHint : %s", getParentInfos()));
10.    }
11.    // cas où le fragment va devenir visible
12.    if (isVisibleToUser) {
13.        // update fragment
14.        if (afterViewsDone && !updateDone) {
15.            update();
16.            updateDone = true;
17.        }
18.    } else {
19.        // on quitte le fragment
20.        updateDone = false;
21.    }
22. }
23.
24. @Override
25. public void onDestroyView() {
26.     // parent
27.     super.onDestroyView();
28.     // mise à jour indicateur
29.     afterViewsDone = false;
30.     // log
31.     if (isDebugEnabled) {
32.         Log.d("AbstractFragment", String.format("onDestroyView : %s", getParentInfos()));
33.     }
34. }
35.
36. @Override
37. public void onResume() {
38.     // parent
39.     super.onResume();
40.     // log
41.     if (isDebugEnabled) {
42.         Log.d("AbstractFragment", String.format("onResume : %s", getParentInfos()));
43.     }
44.     if (isVisibleToUser) {
45.         // update
46.         if (!updateDone) {
47.             update();
48.             updateDone = true;
49.         }
50.     }
51. }

```

La méthode [update] est la suivante :

```

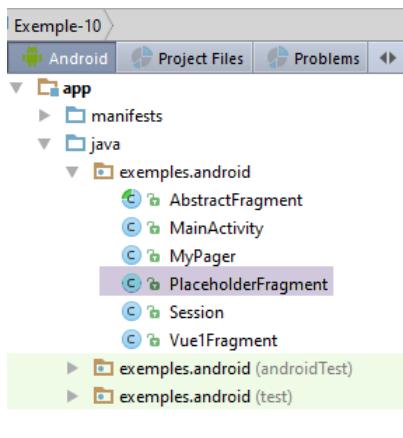
1.     // update fragment
2.     protected void update() {
3.         // on récupère l'activité et la session
4.         if (activity == null) {
5.             Activity activity = getActivity();
6.             if (activity != null) {
7.                 this.activity = (MainActivity) activity;
8.                 this.session = this.activity.getSession();
9.             }
10.        }
11.        // on demande à la classe fille de se mettre à jour
12.        updateFragment();
13.    }

```

D'après le code ci-dessus, lorsque la méthode [update] d'un fragment s'exécute, celui-ci est visible. Cela est important parce que cela signifie que la méthode [Fragment.getActivity] rend alors une référence sur l'activité de l'application (cf paragraphe 1.10.8, page 111), ce qui donne ensuite accès à la session.

- lignes 4-10 : on initialise l'activité et la session si ce n'a pas déjà été fait ;
- ligne 12 : on appelle la méthode [updateFragment] de la classe fille. Lorsque celle-ci s'exécutera, les champs [activity] et [session] auxquels elle a accès autont été initialisés ;

1.11.4 La classe [PlaceholderFragment]



La classe [PlaceholderFragment] évolue de la façon suivante :

```
1. package exemples.android;
2.
3. import android.support.v4.app.Fragment;
4. import android.util.Log;
5. import android.widget.TextView;
6. import org.androidannotations.annotations.*;
7.
8. // un fragment est une vue affichée par un conteneur de fragments
9. @EFragment(R.layout.fragment_main)
10. public class PlaceholderFragment extends AbstractFragment {
11.
12.     // composant de l'interface visuelle
13.     @ViewById(R.id.section_label)
14.     protected TextView textViewInfo;
15.
16.     // data
17.     private boolean initDone;
18.
19.     // data
20.     private String text;
21.     private int numVisit;
22.
23.     // n° de fragment
24.     private static final String ARG_SECTION_NUMBER = "section_number";
25.
26.     // constructeur
27.     public PlaceholderFragment() {
28.         super();
29.         // log
30.         if (isDebugEnabled) {
31.             Log.d("PlaceholderFragment", "constructor");
32.         }
33.     }
34.
35.
36.     @AfterViews
37.     protected void afterViews() {
38.         // mémoire
39.         afterViewsDone = true;
40.         ...
41.     }
42.
43.     // update fragment
44.     public void updateFragment() {
45.         ...
46.     }
47.
48. }
```

- ligne 10 : la classe [PlaceholderFragment] étend la classe [AbstractFragment]. Avec cette architecture, l'écriture d'un fragment consiste :
 - à écrire la méthode [@AfterViews] qui sert à initialiser le fragment lors de son premier cycle de vie ou à le réinitialiser s'il y a eu un [onDestroyView] auparavant. La ligne 39 est obligatoire pour gérer correctement le cycle de vie du fragment ;

- à écrire la méthode [updateFragment] qui va mettre à jour le fragment juste avant son affichage. Cette méthode peut utiliser la session de sa classe parent ;
- à écrire les gestionnaires d'événements du fragment. C'est ce que nous allons faire dans de futurs projets ;

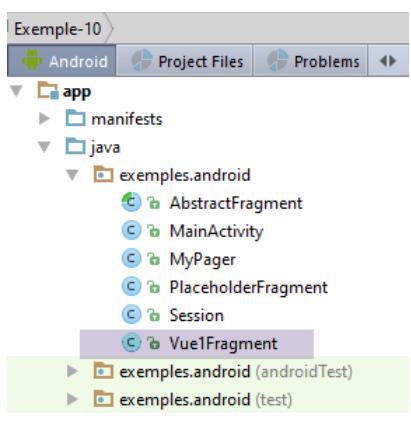
Les méthodes [@AfterViews] et [updateFragment] restent analogues à ce qu'elles étaient dans le projet précédent :

```

1.  @AfterViews
2.  protected void afterViews() {
3.      // mémoire
4.      afterViewsDone = true;
5.      // log
6.      if (isDebugEnabled) {
7.          Log.d("PlaceholderFragment", String.format("afterViews %s - %s - %s", getArguments().getInt(ARG_SECTION_NUMBER),
8.              getParentInfos(), getLocalInfos()));
9.      }
10.     if (!initDone) {
11.         // texte initial
12.         text = getString(R.string.section_format, getArguments().getInt(ARG_SECTION_NUMBER));
13.         initDone = true;
14.     }
15.     // affichage texte courant
16.     textViewInfo.setText(text);
17. }
18.
19. // update fragment
20. public void updateFragment() {
21.     // log
22.     if (isDebugEnabled) {
23.         Log.d("PlaceholderFragment", String.format("update %s - %s - %s", getArguments().getInt(ARG_SECTION_NUMBER),
24.             getParentInfos(), getLocalInfos()));
25.     }
26.     // incrément n° de visite
27.     numVisit = session.getNumVisit();
28.     numVisit++;
29.     session.setNumVisit(numVisit);
30.     // texte modifié
31.     textViewInfo.setText(String.format("%s, visite %s", text, numVisit));
32. }
33. // infos locales pour logs
34. protected String getLocalInfos() {
35.     return String.format("numVisit=%s, initDone=%s, getActivity()==null:%s",
36.         numVisit, initDone, getActivity() == null);
37. }
```

- ligne 7 et 23 : dans les logs, on affiche les informations de la classe parent avec la méthode héritée [getParentInfos] ;

1.11.5 La classe [Vue1Fragment]



La classe [Vue1Fragment] présente la même structure que la classe [PlaceholderFragment] :

```

1. package exemples.android;
2.
3. import android.util.Log;
4. import android.widget.EditText;
5. import android.widget.Toast;
6. import org.androidannotations.annotations.*;
7.
```

```

8. @EFragment(R.layout.vue1)
9. public class Vue1Fragment extends AbstractFragment {
10.
11.    // les éléments de l'interface visuelle
12.    @ViewById(R.id.editTextNom)
13.    protected EditText editTextNom;
14.
15.    // data
16.    private int numVisit;
17.
18.    @AfterViews
19.    protected void afterViews() {
20.        // mémoire
21.        afterViewsDone = true;
22.        // log
23.        if (isDebugEnabled) {
24.            Log.d("Vue1Fragment", String.format("afterViews %s - %s", getParentInfos(), getLocalInfos()));
25.        }
26.    }
27.
28.    // gestionnaire d'évt
29.    @Click(R.id.buttonValider)
30.    protected void doValider() {
31.        // on affiche le nom saisi
32.        Toast.makeText(getActivity(), String.format("Bonjour %s", editTextNom.getText().toString()), Toast.LENGTH_LONG).show();
33.    }
34.
35.    // infos locales pour logs
36.    protected String getLocalInfos() {
37.        return String.format("numVisit=%s", numVisit);
38.    }
39.
40.    // mise à jour fragment
41.    @Override
42.    protected void updateFragment() {
43.        // incrément n° de visite
44.        numVisit = session.getNumVisit();
45.        numVisit++;
46.        session.setNumVisit(numVisit);
47.        // on affiche le n° de la visite
48.        Toast.makeText(getActivity(), String.format("Visite n° %s", numVisit), Toast.LENGTH_SHORT).show();
49.    }
50. }

```

- ligne 9 : la classe [Vue1Fragment] étend la classe [AbstractFragment] ;
- lignes 18-26 : la méthode [@AfterViews] n'a rien d'intéressant à faire. Il faut quand même l'écrire pour mettre le booléen [afterViewsDone] à *true*, car cette information est utilisée par la classe parent ;
- lignes 42-49 : la méthode [updateFragment] consiste à afficher un message court montrant le n° de la visite (ligne 48) et à incrémenter ce n° dans la session (lignes 44-46) ;

Le lecteur est invité à tester ce nouveau projet.

Nous reprendrons dans tous les futurs projets cette architecture :

- une activité et n fragments ;
- tous les fragments étendent la classe [AbstractFragment] ;
- les données à partager entre fragments et entre fragments et activité sont placées dans la classe [Session] ;

1.11.6 Assocation onglets / fragments

Dans la classe [MainActivity] qui gère les onglets il est écrit :

```

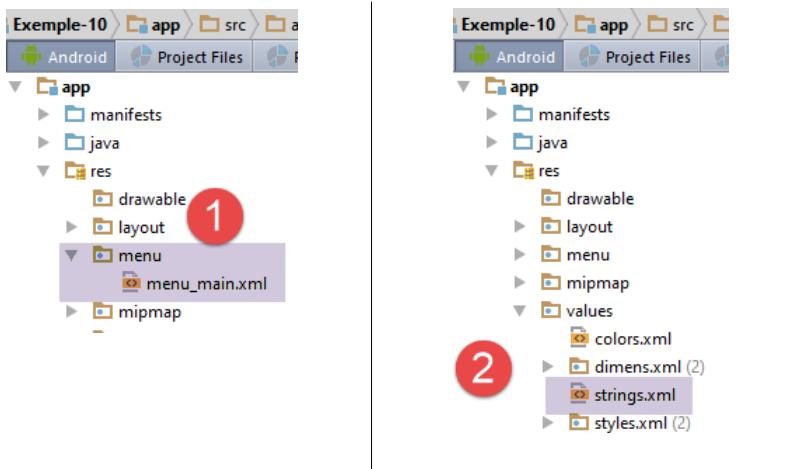
1. // la barre d'onglets est également associée au conteneur de fragments
2. // c-à-d que l'onglet n° i affiche le fragment n° i du conteneur
3. tabLayout.setupWithViewPager(mViewPager);

```

La ligne 3 associe le gestionnaire d'onglets au conteneur de fragment. On a vu une conséquence de cette association : lorsque l'utilisateur clique sur l'onglet n° i, le conteneur de fragments fait afficher le fragment n° i. On n'a pas vu l'inverse : lorsqu'on demande au conteneur de fragments d'afficher le fragment n° i, alors l'onglet n° i est automatiquement sélectionné.

Pour illustrer ce comportement, nous allons ajouter les options [Fragment 1, Fragment 2, ...] au menu actuel. Lorsque l'utilisateur cliquera l'option [Fragment i], on demandera au conteneur de fragments d'afficher le fragment n° i. On verra alors si l'onglet n° i a été sélectionné ou non.

Cette étape commence par le changement du menu de l'application :



Le contenu du fichier [res / menu / menu_main.xml] évolue de la façon suivante :

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.   xmlns:app="http://schemas.android.com/apk/res-auto"
3.   xmlns:tools="http://schemas.android.com/tools"
4.   tools:context="exemples.android.MainActivity">
5.     <item android:id="@+id/action_settings"
6.       android:title="@string/action_settings"
7.       android:orderInCategory="100"
8.       app:showAsAction="never"/>
9.     <item android:id="@+id/fragment1"
10.       android:title="@string/fragment1"
11.       android:orderInCategory="100"
12.       app:showAsAction="never"/>
13.     <item android:id="@+id/fragment2"
14.       android:title="@string/fragment2"
15.       android:orderInCategory="100"
16.       app:showAsAction="never"/>
17.     <item android:id="@+id/fragment3"
18.       android:title="@string/fragment3"
19.       android:orderInCategory="100"
20.       app:showAsActions="never"/>
21.     <item android:id="@+id/fragment4"
22.       android:title="@string/fragment4"
23.       android:orderInCategory="100"
24.       app:showAsActions="never"/>
25.     <item android:id="@+id/fragment5"
26.       android:title="@string/fragment5"
27.       android:orderInCategory="100"
28.       app:showAsAction="never"/>
29.   </menu>

```

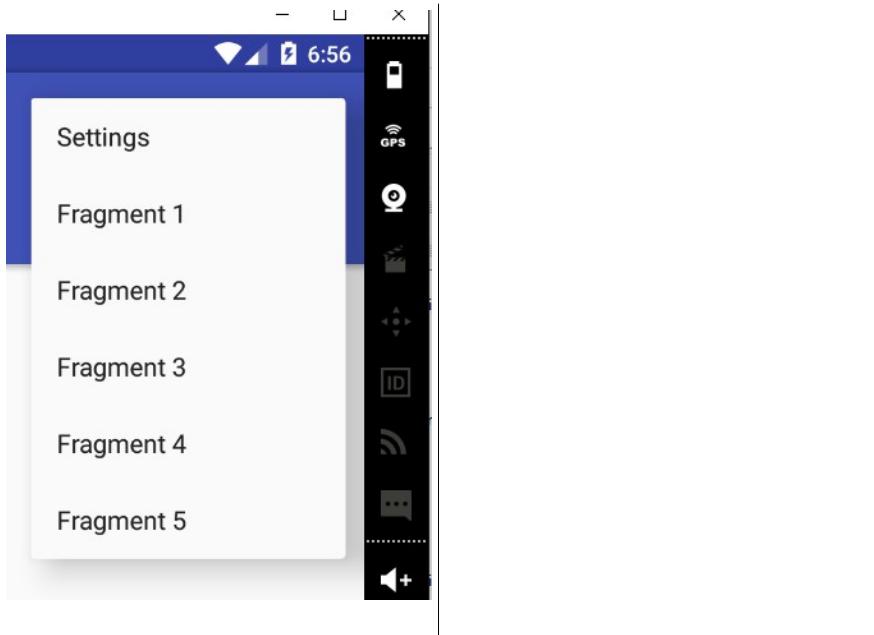
- lignes 9-28 : les cinq nouvelles options du menu ;
- les libellés des options (lignes 10, 14, 18, 22, 26) sont définis dans le fichier [res / values / strings.xml] [2] :

```

1. <resources>
2.   <string name="app_name">Exemple-10</string>
3.   <string name="action_settings">Settings</string>
4.   <string name="section_format">Hello World from section: %1$d</string>
5.   <!-- vue 1 -->
6.   <string name="titre_vue1">Vue n° 1</string>
7.   <string name="txt_nom">Quel est votre nom ?</string>
8.   <string name="btn_valider">Valider</string>
9.   <string name="btn_vue2">Vue n° 2</string>
10.  <!-- menu -->
11.  <string name="fragment1">Fragment 1</string>
12.  <string name="fragment2">Fragment 2</string>
13.  <string name="fragment3">Fragment 3</string>
14.  <string name="fragment4">Fragment 4</string>
15.  <string name="fragment5">Fragment 5</string>
16. </resources>

```

Le résultat visuel est le suivant :



La gestion du clic sur ces options de menu se fait dans la classe [MainActivity] :

```

1.  @Override
2.  public boolean onOptionsItemSelected(MenuItem item) {
3.      // log
4.      if (IS_DEBUG_ENABLED) {
5.          Log.d("menu", "onOptionsItemSelected");
6.      }
7.      // traitement des options de menu
8.      int id = item.getItemId();
9.      switch (id) {
10.          case R.id.action_settings: {
11.              if (IS_DEBUG_ENABLED) {
12.                  Log.d("menu", "action_settings selected");
13.              }
14.              break;
15.          }
16.          case R.id.fragment1: {
17.              showFragment(0);
18.              break;
19.          }
20.          case R.id.fragment2: {
21.              showFragment(1);
22.              break;
23.          }
24.          case R.id.fragment3: {
25.              showFragment(2);
26.              break;
27.          }
28.          case R.id.fragment4: {
29.              showFragment(3);
30.              break;
31.          }
32.          case R.id.fragment5: {
33.              showFragment(4);
34.              break;
35.          }
36.      }
37.      // item traité
38.      return true;
39.  }
40.
41.  private void showFragment(int i) {
42.      if (i < FRAGMENTS_COUNT && mViewPager.getCurrentItem() != i) {
43.          // on change le fragment affiché
44.          mViewPager.setCurrentItem(i);
45.      }
46.  }

```

- ligne 2 : la méthode [onOptionsItemSelected] est appelée lorsque se produit un clic sur l'une des options de menu ;
- ligne 8 : on récupère l'identifiant de l'option cliquée ;
- lignes 9-36 : les différents cas sont traités par un switch ;

- lignes 16-36 : le clic sur l'option [Fragment i] renvoie à la méthode [showFragment(i-1)] des lignes 41-45 ;
- ligne 43 : on demande au conteneur de fragments d'afficher le fragment demandé ;
- ligne 42 : on vérifie auparavant qu'on peut le faire (condition 1) et que c'est nécessaire (condition 2) ;

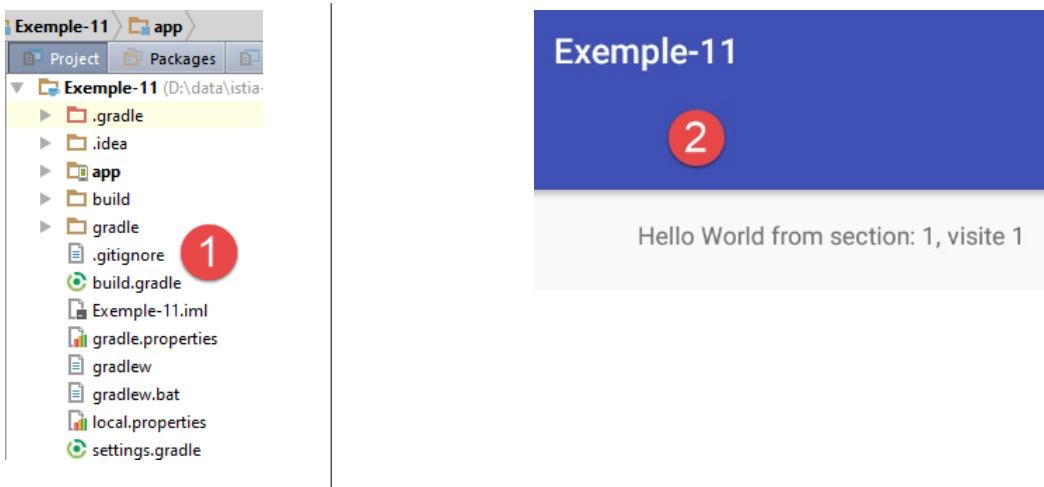
Le lecteur est invité à tester cette nouvelle version. On constate que lorsqu'on demande l'affichage du fragment n° i, celui-ci est bien affiché et l'onglet n° i est lui-même sélectionné.

Maintenant que nous avons vu comment fonctionnait l'association onglets / fragments, nous allons nous intéresser à un autre cas : celui où la gestion des onglets est dissociée de celle des fragments. C'est le cas par exemple lorsqu'il y a moins d'onglets que de fragments. Pour illustrer ce nouveau cas d'utilisation, nous construisons un nouveau projet.

1.12 Exemple-11 : onglets dissociés des fragments

1.12.1 Crédit du projet

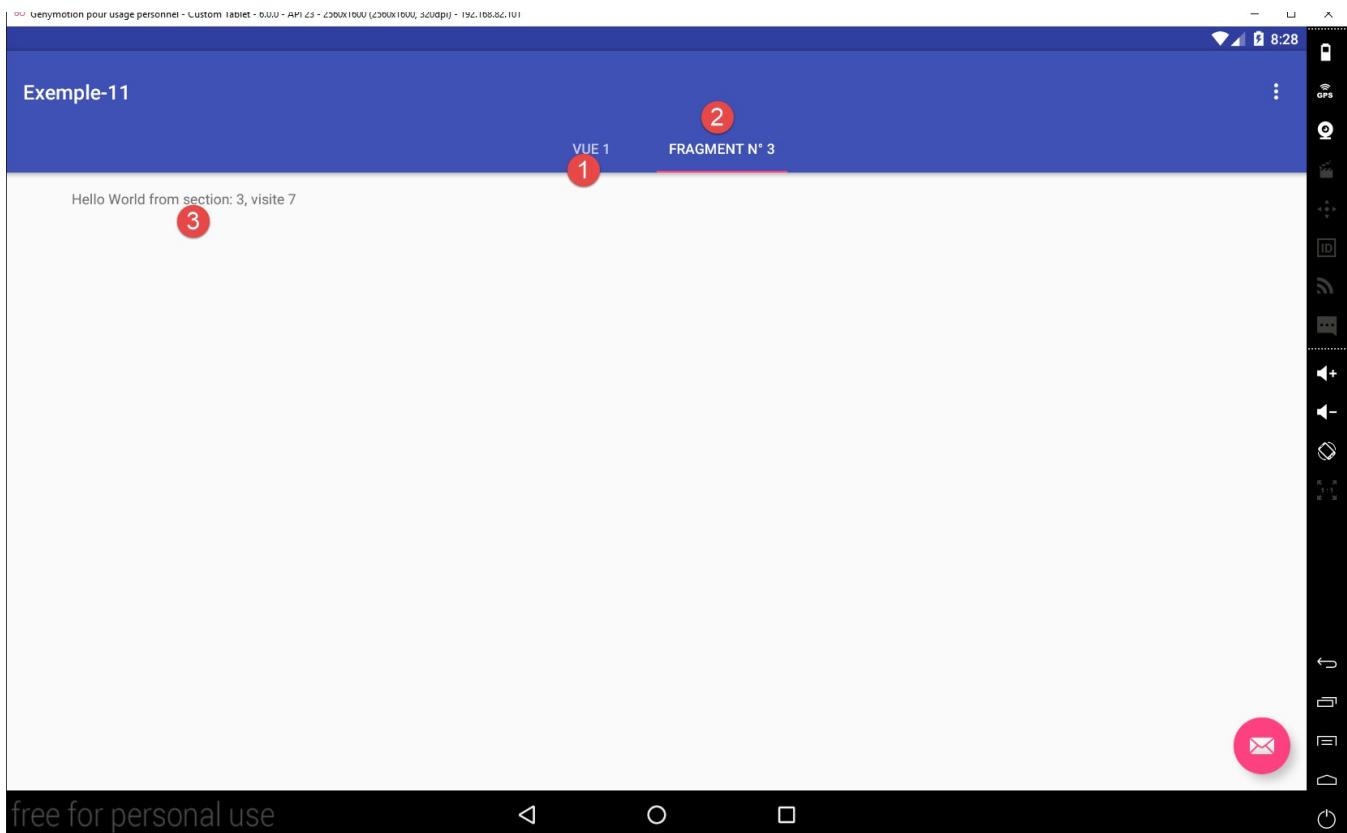
On duplique le projet [Exemple-10] dans [Exemple-11] :



1.12.2 Objectifs

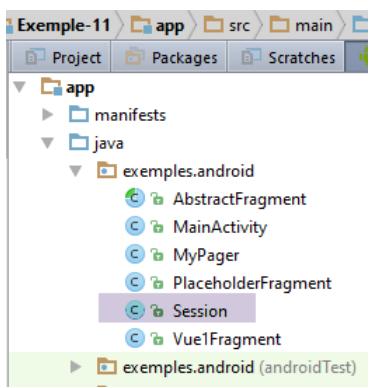
La nouvelle application aura deux onglets :

- le 1er onglet affichera toujours le fragment [Vue1] ;
- le second onglet affichera un fragment choisi dans le menu ;



- en [1], le fragment [Vue1] ;
- en [2], le fragment de type [PlaceholderFragment] choisi par l'utilisateur ;
- en [3], on continue à compter les visites ;

1.12.3 La session



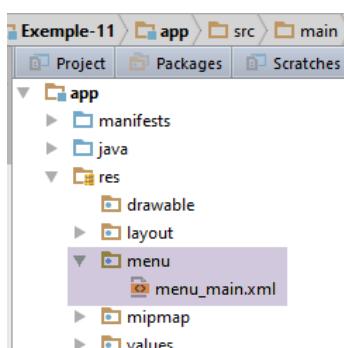
La nouvelle session sera la suivante :

```

1. package exemples.android;
2.
3. import org.androidannotations.annotations.EBean;
4.
5. @EBean(scope = EBean.Scope.Singleton)
6. public class Session {
7.     // nombre de fragments visités
8.     private int numVisit;
9.     // n° fragment de type [PlaceholderFragment] affiché dans second onglet
10.    private int numFragment;
11.
12.    // getters et setters
13. ...
14. }
```

- ligne 10 : nous allons gérer nous-mêmes le clic sur les onglets. Lorsqu'on clique sur un onglet, il faut restituer le fragment qu'il affichait la dernière fois qu'il était sélectionné. Le champ [numFragment] mémorisera le n° de celui-ci pour l'onglet n° 2, un nombre dans [0, Fragments_COUNT-2]. Lorsque l'onglet n° 2 sera cliqué, on ira chercher dans la session, le n° du fragment à afficher ;

1.12.4 Le menu



Le menu [res / menu / menu_main.xml] évolue de la façon suivante :

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.       xmlns:app="http://schemas.android.com/apk/res-auto"
3.       xmlns:tools="http://schemas.android.com/tools"
4.       tools:context="exemples.android.MainActivity">
5.     <item android:id="@+id/action_settings"
6.           android:title="@string/action_settings"
```

```

7.      android:orderInCategory="100"
8.      app:showAsAction="never"/>
9.      <item android:id="@+id/fragment1"
10.         android:title="@string/fragment1"
11.         android:orderInCategory="100"
12.         app:showAsAction="never"/>
13.      <item android:id="@+id/fragment2"
14.         android:title="@string/fragment2"
15.         android:orderInCategory="100"
16.         app:showAsAction="never"/>
17.      <item android:id="@+id/fragment3"
18.         android:title="@string/fragment3"
19.         android:orderInCategory="100"
20.         app:showAsAction="never"/>
21.      <item android:id="@+id/fragment4"
22.         android:title="@string/fragment4"
23.         android:orderInCategory="100"
24.         app:showAsAction="never"/>
25.  </menu>

```

L'onglet n° 2 affichera l'un des quatre fragments des lignes 9-24. Le 5ième fragment est le fragment [Vue1Fragment] qui sera lui toujours affiché dans l'onglet n° 1.

1.12.5 La classe [MainActivity]

La classe [MainActivity] doit désormais gérer les onglets et la navigation entre eux, ce qu'elle ne faisait pas jusqu'à maintenant. Son code évolue de la façon suivante :

```

1.  // le gestionnaire d'onglets
2.  @ViewById(R.id.tabs)
3.  protected TabLayout tabLayout;
4. ...
5. @AfterViews
6. protected void afterViews() {
7.     // log
8.     if (IS_DEBUG_ENABLED) {
9.         Log.d("MainActivity", "afterViews");
10.    }
11. ...
12.
13.     // pas de scrolling
14.     mViewPager.setScrollingEnabled(false);
15.
16.     // affichage Vue1
17.     mViewPager.setCurrentItem(FRAGMENTS_COUNT - 1);
18.
19.     // au départ on n'a qu'un seul onglet
20.     TabLayout.Tab tab = tabLayout.newTab();
21.     tab.setText("Vue 1");
22.     tabLayout.addTab(tab);
23.
24.     // gestionnaire d'évt
25.     tabLayout.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
26.         @Override
27.         public void onTabSelected(TabLayout.Tab tab) {
28.             // un onglet a été sélectionné - on change le fragment affiché par le conteneur de fragments
29.             ...
30.         }
31.
32.         @Override
33.         public void onTabUnselected(TabLayout.Tab tab) {
34.
35.         }
36.
37.         @Override
38.         public void onTabReselected(TabLayout.Tab tab) {
39.
40.         }
41.     });
42.
43. ...
44.
45. }

```

- ligne 17 : le 1er fragment affiché par le conteneur de fragments sera le fragment [Vue1Fragment]. Par construction, ce sera le dernier fragment du conteneur ;
- lignes 20-22 : parce qu'on n'a pas fait d'association entre onglets et conteneur de fragments, nous devons gérer les onglets nous-mêmes. Au départ, la barre d'onglets [tabLayout] de la ligne 3 n'a aucun onglet ;
- ligne 20 : on crée le 1er onglet ;
- ligne 21 : on lui donne un titre. Dans les exemples précédents, le titre des onglets était le titre des fragments. C'est désormais fini. Du coup, on retire la méthode [getPageTitle] du gestionnaire de fragments. On n'en a plus besoin :

```

1.      // facultatif - donne un titre aux fragments générés
2.      @Override
3.      public CharSequence getPageTitle(int position) {
4.          return String.format("Onglet n° %s", (position + 1));
5.      }

```

- ligne 22 : l'onglet créé est ajouté à la barre d'onglets. Notre barre d'onglets a désormais un onglet. Qu'affiche cet onglet ? Il faut comprendre que les onglets et les fragments sont deux notions indépendantes. Le fragment affiché est toujours celui choisi par le conteneur de fragments. Si on change d'onglet et qu'on ne demande pas au conteneur de changer le fragment affiché, rien ne se passe : c'est toujours le même fragment qui est affiché mais l'onglet sélectionné a lui changé. Donc ici, le fragment affiché est celui choisi ligne 17 : le fargment [Vue1Fragment] ;
- lignes 26-30 : la méthode à écrire pour gérer le changement d'onglet par l'utilisateur ;

La méthode [onTabSelected] des lignes 26-30 est déclenchée dès qu'il y a changement d'onglet (si l'utilisateur clique sur un onglet déjà sélectionné, il ne se passe rien). Son code est le suivant :

```

1.      @Override
2.      public void onTabSelected(TabLayout.Tab tab) {
3.          if (IS_DEBUG_ENABLED) {
4.              Log.d("onglets", "onTabSelected");
5.          }
6.          // un onglet a été sélectionné - on change le fragment affiché par le conteneur de fragments
7.          // position de l'onglet
8.          int position = tab.getPosition();
9.          // n° du fragment à afficher
10.         int numFragment;
11.         switch (position) {
12.             case 0:
13.                 // n° fragment [Vue1Fragment]
14.                 numFragment = FRAGMENTS_COUNT - 1;
15.                 break;
16.             default:
17.                 // n° fragment [PlaceholderFragment]
18.                 numFragment = session.getNumFragment();
19.             }
20.             // affichage fragment
21.             mViewPager.setCurrentItem(numFragment);
22.     }

```

- ligne 8 : on récupère la position de l'onglet qui a été cliqué. On va récupérer ici un nombre 0 ou 1 ;
- lignes 12-15 : si c'est le premier onglet qui a été cliqué, on se prépare à afficher le fragment [Vue1Fragment] ;
- lignes 16-18 : dans les autres cas (onglet n° 2 cliqué), on se prépare à réafficher le fragment qui était affiché la dernière fois que l'onglet n° 2 était sélectionné. Le n° de celui-ci avait alors été mis dans la session de l'application ;
- ligne 21 : on demande au conteneur de fragments d'afficher le fragment désiré ;

Voyons maintenant la gestion des options de menu (toujours dans [MainActivity]) :

```

1.      @Override
2.      public boolean onOptionsItemSelected(MenuItem item) {
3.          // log
4.          if (IS_DEBUG_ENABLED) {
5.              Log.d("menu", "onOptionsItemSelected");
6.          }
7.          // traitement des options de menu
8.          int id = item.getItemId();
9.          switch (id) {
10.              case R.id.action_settings: {
11.                  if (IS_DEBUG_ENABLED) {
12.                      Log.d("menu", "action_settings selected");
13.                  }
14.                  break;
15.              }
16.              case R.id.fragment1: {
17.                  showFragment(0);
18.                  break;
19.              }
20.              case R.id.fragment2: {
21.                  showFragment(1);
22.                  break;
23.              }
24.              case R.id.fragment3: {
25.                  showFragment(2);
26.                  break;
27.              }
28.              case R.id.fragment4: {
29.                  showFragment(3);
30.                  break;
31.              }

```

```

32.     }
33.     // item traité
34.     return true;
35. }
```

- lignes 16-31 : gestion des 4 options du menu. Chaque gestionnaire appelle la méthode [showFragment] avec le n° du fragment à afficher ;

La méthode [showFragment] est la suivante :

```

1.    // l'onglet n° 2
2.    private TabLayout.Tab tab2 = null;
3.
4.    private void showFragment(int i) {
5.        if (i < FRAGMENTS_COUNT && mViewPager.getCurrentItem() != i) {
6.            // si le 2ième onglet n'existe pas encore, on le crée
7.            if (tab2 == null) {
8.                tab2 = tabLayout.newTab();
9.                tabLayout.addTab(tab2);
10.            }
11.            // on fixe le titre du second onglet
12.            tab2.setText(String.format("Fragment n° %s", (i + 1)));
13.            // on change le fragment affiché
14.            mViewPager.setCurrentItem(i);
15.            // le n° du fragment affiché est mis en session
16.            session.setNumFragment(i);
17.            // on sélectionne l'onglet 2 - ne fait rien si celui-ci est déjà sélectionné
18.            tab2.select();
19.        }
20.    }
```

- on se rappelle qu'au départ de l'application, on n'a qu'un onglet ;
- ligne 2 : une référence sur l'onglet n° 2, *null* au départ ;
- ligne 5 : les conditions d'affichage n'ont pas changé par rapport à la version précédente ;
- lignes 7-10 : si l'onglet n° 2 n'existe pas encore, il est créé (ligne 8) et ajouté à la barre d'onglets (ligne 9) ;
- ligne 12 : on donne au second onglet le n° du fragment qui va être affiché avec une numérotation commençant à 1 ;
- ligne 14 : le fragment désiré est affiché ;
- ligne 16 : son n° est mis dans la session ;
- ligne 18 : l'onglet n° 2 est sélectionné. S'il était déjà sélectionné, rien ne se passera : la méthode [onTabSelected] ne sera pas exécutée. S'il n'était pas déjà sélectionné, la méthode [onTabSelected] va se déclencher. Cette méthode demande alors au conteneur de fragments d'afficher le fragment déjà affiché ligne 14. Un simple test dans la méthode [onTabSelected] évite ce cas :

```

1.        // affichage fragment seulement si c'est nécessaire
2.        if (numFragment != mViewPager.getCurrentItem()) {
3.            mViewPager.setCurrentItem(numFragment);
4.        }
```

Le lecteur est invité à tester cette nouvelle version.

1.12.6 Améliorations

Nous avons désormais une bonne compréhension des fragments, de leur cycle de vie, de la notion d'adjacence de fragments et de leur relation avec la barre d'onglets. Nous avons par ailleurs une architecture robuste qui vient de passer le test de l'exemple 11 :

- une activité et n fragments ;
- tous les fragments étendent la classe [AbstractFragment] ;
- les données à partager entre fragments et entre fragments et activité sont placées dans la classe [Session] ;

Nous allons dans un nouveau projet préciser les relations entre activité et fragments par l'ajout d'une interface.

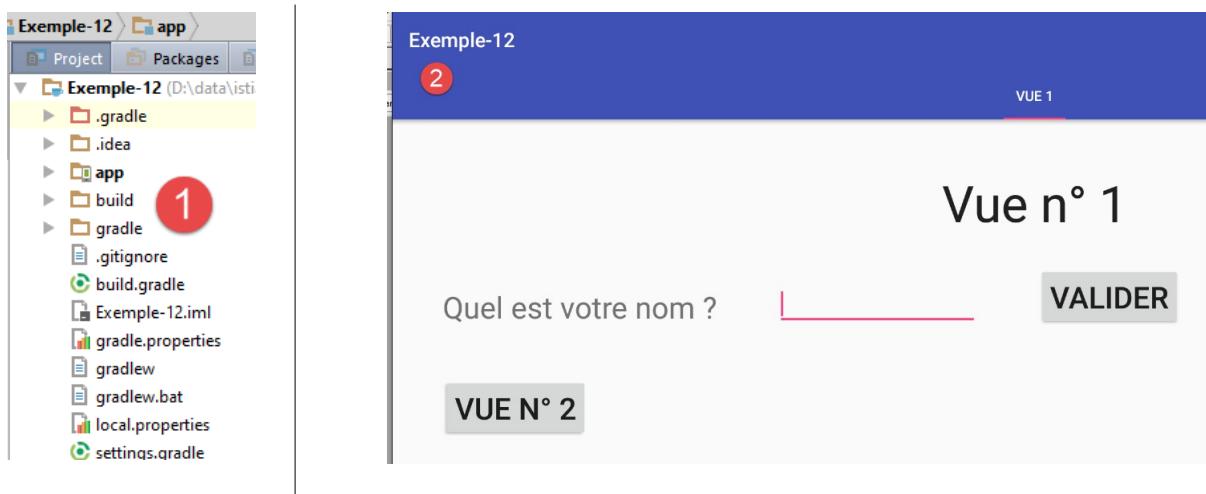
1.13 Exemple-12 : codifier les relations entre activité et fragments

Dans cet exemple, nous voulons définir les relations minimales entre activité et fragments. Pour cela, nous utiliserons :

- une interface [IMainActivity] qui définira ce que les fragments peuvent demander à l'activité ;
- une classe abstraite [AbstractFragment] qui définira l'état et les méthodes que devrait avoir tout fragment ;

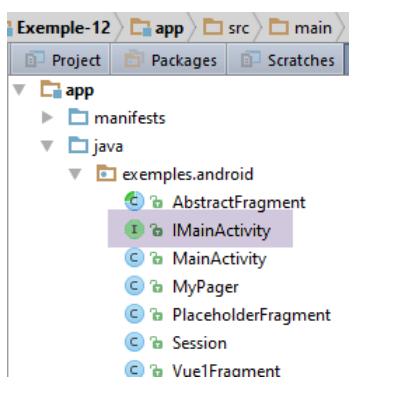
1.13.1 Crédation du projet

Nous dupliquons le projet [Exemple-11] dans [Exemple-12] en suivant la procédure du paragraphe 1.4, page 36. Nous obtenons le résultat suivant :



1.13.2 L'interface [IMainActivity]

Des exemples précédents, il apparaît que les fragments ont besoin d'avoir accès à la session instanciée par l'activité. Par ailleurs, pas visible dans ces exemples, mais prévisible : les gestionnaires des événements des fragments se terminent parfois par un changement de vue. On demandera à l'activité d'opérer ce changement. L'interface [IMainActivity] pourrait être alors la suivante :



```
1. package exemples.android;
2.
3. public interface IMainActivity {
4.
5.     // accès à la session
6.     Session getSession();
7.
8.     // changement de vue
9.     void navigateToView(int position);
10.
11.    // mode debug
12.    boolean IS_DEBUG_ENABLED = true;
13. }
```

Ligne 12, on notera la présence d'une constante qui était auparavant dans la classe [MainActivity]. On veut réduire le couplage entre les fragments et l'activité et le réduire à un couplage entre [AbstractFragment] et [IMainActivity]. L'activité pourra alors s'appeler autrement que [MainActivity]. La constante [IS_DEBUG_ENABLED] étant utilisée dans les fragments, elle est déplacée dans l'interface [IMainActivity].

1.13.3 La classe abstraite [AbstractFragment]

La classe abstraite [AbstractFragment] évolue très peu :

```

1.  // données accessibles aux classes filles
2.  protected boolean afterViewsDone = false;
3.  final protected boolean isDebugEnabled = IMainActivity.IS_DEBUG_ENABLED;
4.
5.  // activité
6.  protected IMainActivity mainActivity;
7.  protected Activity activity;
8.
9.  ...
10. // update fragment
11. protected void update() {
12.     // on récupère l'activité et la session
13.     if (mainActivity == null) {
14.         this.activity = getActivity();
15.         if (this.activity != null) {
16.             this.mainActivity = (IMainActivity) activity;
17.             this.session = this.mainActivity.getSession();
18.         }
19.     }
20.     // on demande à la classe fille de se mettre à jour
21.     updateFragment();
22. }
```

- lignes 6 et 7 : on entretient deux types de référence sur l'activité :
 - ligne 6 : une référence sur l'activité implémentant l'interface [IMainActivity] ;
 - ligne 7 : une référence sur l'activité héritant de la classe Android [Activity]. C'est le cas de toute activité ;

Ces deux références pointent bien sûr sur le même objet. Mais celui-ci est vu avec deux types différents. Cela nous évitera des transtypes à l'exécution ;

- ligne 14 : on récupère une référence sur l'activité au moyen de la méthode [getActivity] ;
- ligne 15 : si celle-ci est non nulle, alors on peut avoir accès à la session ;
- lignes 16-17 : on mémorise l'activité en tant qu'implémentant l'interface [IMainActivity] et la session ;

1.13.4 Modification du gestionnaire de fragments

Le gestionnaire de fragments [SectionsPagerAdapter] dans la classe [MainActivity] est modifié en un seul point : au lieu de gérer des fragments de type [Fragment], il gère désormais des fragments de type [AbstractFragment] :

```

1.  public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.      // les fragments
4.      private AbstractFragment[] fragments;
5.      // n° de fragment
6.      private static final String ARG_SECTION_NUMBER = "section_number";
7.
8.      // constructeur
9.      public SectionsPagerAdapter(FragmentManager fm) {
10.          // parent
11.          super(fm);
12.          // initialisation du tableau des fragments
13.          fragments = new AbstractFragment[FRAGMENTS_COUNT];
14.          for (int i = 0; i < fragments.length - 1; i++) {
15.              ...
16.          }
17.          // un fragment de +
18.          fragments[fragments.length - 1] = new Vue1Fragment_();
19.      }
20.
21.      // fragment n° position
22.      @Override
23.      public AbstractFragment getItem(int position) {
24.          ...
25.      }
26.
27.      // rend le nombre de fragments générés
28.      @Override
29.      public int getCount() {
30.          ...
31.      }
```

1.13.5 Modification de la classe [MainActivity]

La classe [MainActivity] doit implémenter l'interface [IMainActivity] :

```

1.  @EActivity(R.layout.activity_main)
2.  public class MainActivity extends AppCompatActivity implements IMainActivity{
3.
4.  ...
5.  // injection session
6.  @Bean(Session.class)
7.  protected Session session;
8.  ...
9.  // getter session
10. public Session getSession() {
11.     return session;
12. }
13.
14. @Override
15. public void navigateToView(int position) {
16.     // on affiche la vue position
17.     if(mViewPager.getCurrentItem()!=position){
18.         // affichage fragment
19.         mViewPager.setCurrentItem(position);
20.     }
21. }
22.
```

- lignes 10-12 : la méthode [getSession] existait déjà ;
- lignes 15-22 : la méthode [navigateToView] fait afficher le fragment n° [position] ;
- ligne 17 : on regarde s'il y a quelque chose à faire ;
- ligne 19 : le fragment n° [position] est affiché ;

A ce stade, exécutez l'application. Elle doit fonctionner.

1.13.6 Modification de l'affichage des fragments dans [MainActivity]

Actuellement, la classe [MainActivity] affiche un fragment par l'instruction :

```

1.      // affichage Vue1
2.      mViewPager.setCurrentItem(FRAGMENTS_COUNT - 1);
```

Comme la méthode [navigateToView] fait la même chose, on remplace ce type d'instruction partout (2 endroits) par :

```
navigateToView(...);
```

Exécutez ensuite l'application. Elle doit toujours fonctionner.

1.13.7 Conclusion

A partir de maintenant, nous utiliserons toujours l'architecture précédente :

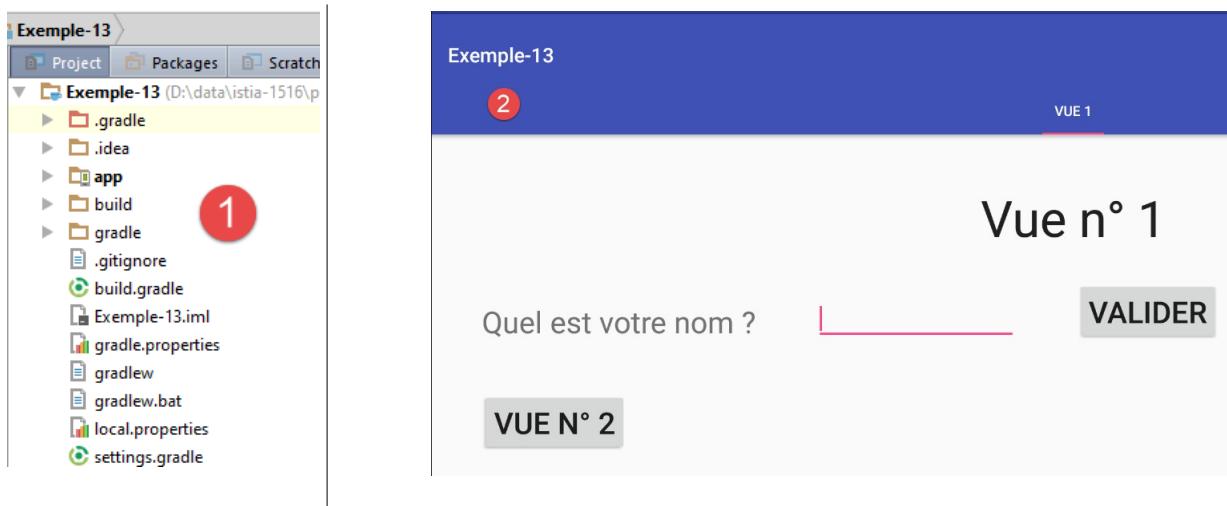
- une activité implémentant l'interface [IMainActivity] ;
- des fragments étendant la classe [AbstractFragment], ce qui leur impose d'implémenter la méthode [updateFragment]. Ceux-ci doivent avoir également une méthode [@AfterViews] dans laquelle elles mettent le booléen [afterViewsDone] à *true* ;
- une session encapsulant les données à partager entre fragments et activité ;

1.14 Exemple-13 : Exemple-05 avec des fragments

Dans le projet [Exemple-05] nous avons introduit la navigation entre vues. Il s'agissait alors d'une navigation entre activités : **1 vue = 1 activité**. Nous nous proposons ici d'avoir une seule activité avec plusieurs vues de type [AbstractFragment].

1.14.1 Crédation du projet

Nous dupliquons le projet précédent [Exemple-12] dans [Exemple-13] en suivant la procédure du paragraphe 1.4, page 36. Nous obtenons le résultat suivant :

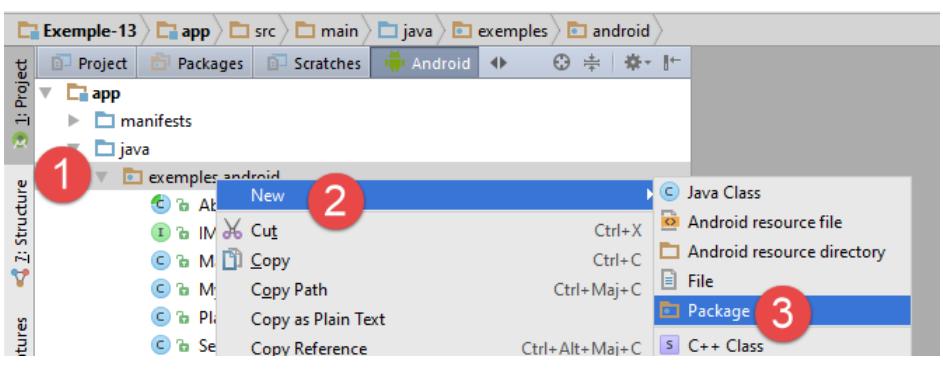


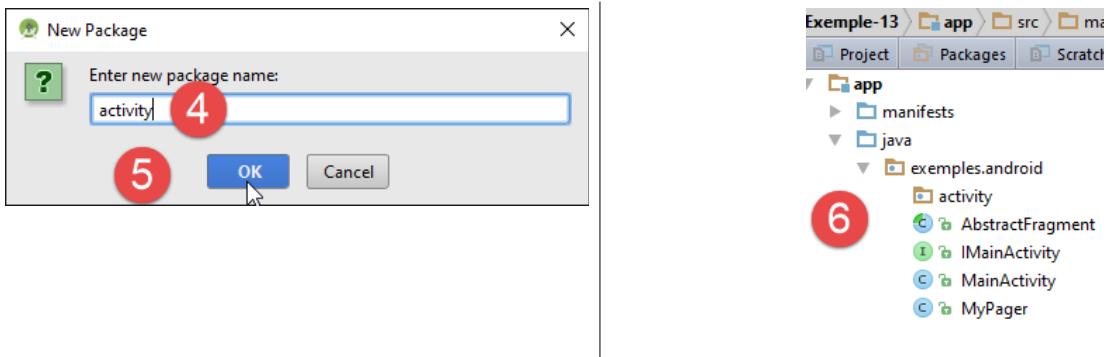
1.14.2 Structuration du projet

Nous allons commencer à utiliser des packages pour organiser le code. Pour l'instant, nous pouvons distinguer deux domaines distincts :

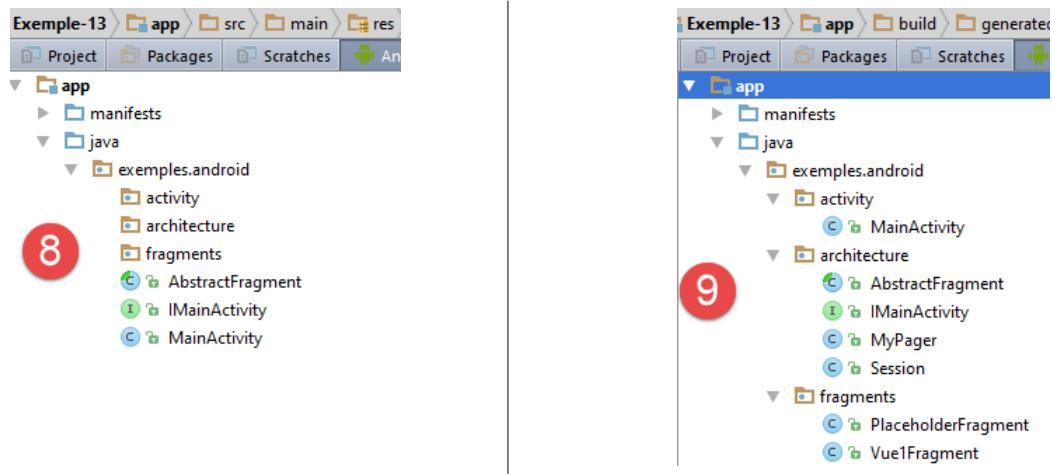
- la gestion de l'activité ;
- la gestion des fragments ;

Nous créons pour eux deux packages [exemples.android.activity] et [exemples.android.fragments] :





On fait de même pour créer le package [exemples.android.fragments] :



En [8], on crée un troisième package appelé [architecture] dans lequel nous mettrons les entités [IMainActivity, AbstractFragment, Session, MyPagerAdapter] qui sont les éléments de base de l'architecture de notre application. Ceci pour nous rappeler qu'on a fait un choix d'architecture précis. Ensuite, déplacez les éléments existants du projet comme indiqué en [9]. Chaque déplacement doit être validé par un clic sur le bouton [Refactor].

A ce stade, compilez l'application. On a les erreurs suivantes dans [MainActivity] :

```

12 import android.view.Menu;
13 import android.view.MenuItem;
14 import android.view.View;
15 import exemples.android.PlaceholderFragment_;
16 import exemples.android.R;
17 import exemples.android.Vue1Fragment_;
18 import exemples.android.architecture.AbstractFragment;
19 import exemples.android.architecture.IMainActivity;
20 import exemples.android.architecture.MyPagerAdapter;
21 import exemples.android.architecture.Session;
22 import org.androidannotations.annotations.*;
23

```

Lors des déplacements de classes vers les packages, Android Studio a fait les changements nécessaires dans les codes de l'application (lignes 18-21, par exemple). Les classes concernées par les lignes 15 et 17 n'ont pas été déplacées. Elles sont générées par la bibliothèque Android Annotations. Pour ces classes, il faut changer les *imports* à la main. Ces lignes deviennent donc :

```

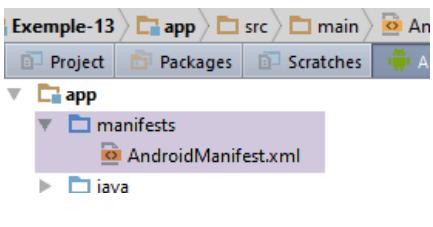
14 import android.view.View;
15 import exemples.android.fragments.PlaceholderFragment_;
16 import exemples.android.R;
17 import exemples.android.fragments.Vue1Fragment_;
18 import exemples.android.architecture.AbstractFragment;
19 import exemples.android.architecture.IMainActivity;
20 import exemples.android.architecture.MyPagerAdapter;
21 import exemples.android.architecture.Session;

```

Ceci fait, il n'y a plus d'erreur de compilation. Exécutez l'application. On a l'erreur suivante :

```
java.lang.RuntimeException: Unable to instantiate activity ComponentInfo{exemples.android/exemples.android.MainActivity_}:
```

Cette erreur provient du manifeste de l'application :



```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.           package="exemples.android">
4.
5.     <application
6.         android:allowBackup="true"
7.         android:icon="@mipmap/ic_launcher"
8.         android:label="@string/app_name"
9.         android:supportsRtl="true"
10.        android:theme="@style/AppTheme">
11.         <activity
12.             android:name=".MainActivity_"
13.             android:label="@string/app_name"
14.             android:theme="@style/AppTheme.NoActionBar">
15.             <intent-filter>
16.               <action android:name="android.intent.action.MAIN"/>
17.
18.               <category android:name="android.intent.category.LAUNCHER"/>
19.             </intent-filter>
20.         </activity>
21.     </application>
22.
23. </manifest>

```

Les lignes 3 et 12 font que l'activité désignée est [exemples.android.MainActivity_]. Or du fait que l'activité a migré dans le package [activity], la ligne 12 doit être désormais :

```
    android:name=".activity.MainActivity_"
```

Attention au . devant [activity]. De nouveau, Android Studio n'a pu mettre à jour le manifeste parce que celui-ci référence une classe Android Annotations qui elle n'a pas subi de déplacement. L'utilisation de la bibliothèque AA amène donc avec elle un certain nombre de désagréments.

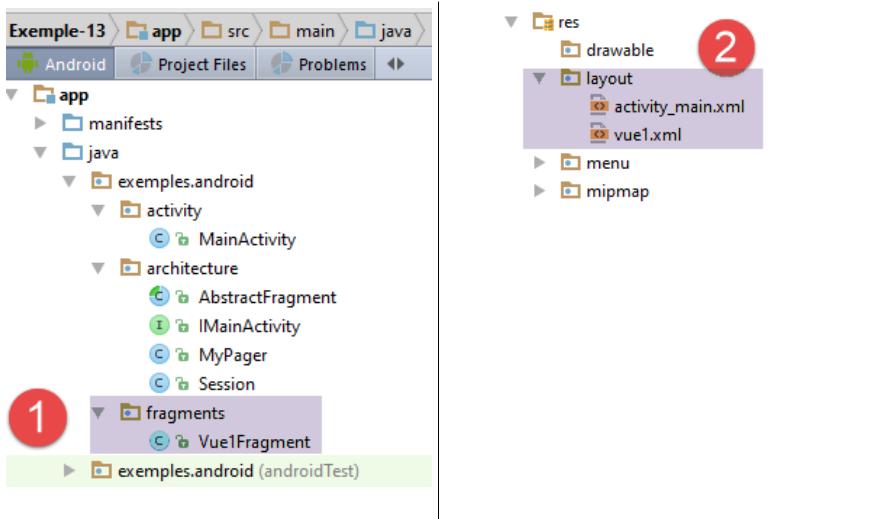
1.14.3 Nettoyage du projet

Dans le nouveau projet :

- il n'y a plus d'onglets, de bouton flottant, de menu ;
- les fragments [PlaceholderFragment] disparaissent. L'application va gérer deux fragments [Vue1Fragment] qu'on a déjà et [Vue2Fragment] qu'il faudra créer ;
- la session n'est plus la même ;

1.14.3.1 Nettoyage des fragments

Supprimez la classe [PlaceHolderFragment] [1] :



De même supprimez la vue [res / layout / fragment_main.xml] associée à ce fragment [2].

1.14.3.2 Nettoyage de la session

La session est actuellement la suivante :

```

1. package exemples.android.architecture;
2.
3. import org.androidannotations.annotations.EBean;
4.
5. @EBean(scope = EBean.Scope.Singleton)
6. public class Session {
7.     // nombre de fragments visités
8.     private int numVisit;
9.     // n° fragment de type [PlaceholderFragment] affiché dans second onglet
10.    private int numFragment;
11.
12.    // getters et setters
13.
14.    public int getNumVisit() {
15.        return numVisit;
16.    }
17.
18.    public void setNumVisit(int numVisit) {
19.        this.numVisit = numVisit;
20.    }
21.
22.    public int getNumFragment() {
23.        return numFragment;
24.    }
25.
26.    public void setNumFragment(int numFragment) {
27.        this.numFragment = numFragment;
28.    }
29. }
```

Nous ne gardons rien de cette session.

Compilez le projet. Les lignes erronées sont celles qui utilisaient le contenu de la session. Supprimez-les. Dans la classe [Vue1Fragment] on supprime également la variable [numVisit] du code qui devient le suivant :

```

1. package exemples.android.fragments;
2.
3. import android.util.Log;
4. import android.widget.EditText;
5. import android.widget.Toast;
6. import exemples.android.R;
7. import exemples.android.architecture.AbstractFragment;
8. import org.androidannotations.annotations.AfterViews;
9. import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.ViewById;
12.
13. @EFragment(R.layout.vue1)
14. public class Vue1Fragment extends AbstractFragment {
```

```

16. // les éléments de l'interface visuelle
17. @ViewById(R.id.editTextNom)
18. protected EditText editTextNom;
19.
20. @AfterViews
21. protected void afterViews() {
22.     // mémoire
23.     afterViewsDone = true;
24.     // log
25.     if (isDebugEnabled) {
26.         Log.d("Vue1Fragment", String.format("afterViews %s", getParentInfos()));
27.     }
28. }
29.
30. // gestionnaire d'évt
31. @Click(R.id.buttonValider)
32. protected void doValider() {
33.     // on affiche le nom saisi
34.     Toast.makeText(getActivity(), String.format("Bonjour %s", editTextNom.getText().toString()), Toast.LENGTH_LONG).show();
35. }
36.
37.
38. // mise à jour fragment
39. @Override
40. protected void updateFragment() {
41. }
42. }
```

1.14.3.3 Suppression des onglets, du bouton flottant et du menu

La suppression des onglets et du bouton flottant se fait à deux endroits :

- dans la vue [res / layout / activity-main.xml] qui définit ces éléments et leur emplacement dans la vue ;
- dans le code de l'activité [MainActivity] ;

La suppression du menu se fait également à deux endroits :

- dans la vue [res / menu / menu-main.xml] qui définit les options du menu ;
- dans le code de l'activité [MainActivity] ;

Le code de la vue la vue [res / layout / activity-main.xml] est actuellement le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     xmlns:app="http://schemas.android.com/apk/res-auto"
5.     android:id="@+id/main_content"
6.     android:layout_width="match_parent"
7.     android:layout_height="match_parent"
8.     android:fitsSystemWindows="true"
9.     tools:context=".activity.MainActivity">
10.
11.    <android.support.design.widget.AppBarLayout
12.        android:id="@+id/appbar"
13.        android:layout_width="match_parent"
14.        android:layout_height="wrap_content"
15.        android:paddingTop="@dimen/appbar_padding_top"
16.        android:theme="@style/AppTheme.AppBarOverlay">
17.
18.        <android.support.v7.widget.Toolbar
19.            android:id="@+id/toolbar"
20.            android:layout_width="match_parent"
21.            android:layout_height="?attr/actionBarSize"
22.            android:background="?attr/colorPrimary"
23.            app:popupTheme="@style/AppTheme.PopupOverlay"
24.            app:layout_scrollFlags="scroll|enterAlways">
25.
26.    </android.support.v7.widget.Toolbar>
27.
28.    <android.support.design.widget.TabLayout
29.        android:id="@+id/tabs"
30.        android:layout_width="match_parent"
31.        android:layout_height="wrap_content"/>
32.
33. </android.support.design.widget.AppBarLayout>
34.
35. <exemples.android.architecture.MyPagerAdapter
36.     android:id="@+id/container"
37.     android:layout_width="match_parent"
38.     android:layout_height="match_parent"
39.     app:layout_behavior="@string/appbar_scrolling_view_behavior"/>
40.
41. <android.support.design.widget.FloatingActionButton
```

```

42.     android:id="@+id/fab"
43.     android:layout_width="wrap_content"
44.     android:layout_height="wrap_content"
45.     android:layout_gravity="end|bottom"
46.     android:layout_margin="@dimen/fab_margin"
47.     android:src="@android:drawable/ic_dialog_email"/>
48.
49. </android.support.design.widget.CoordinatorLayout>

```

- on supprime les lignes [28-31, 41-47] ;
- on supprime également la barre d'outils des lignes 18-24 ;

Le code du menu [res / menu / menu_main.xml] est actuellement le suivant :

```

1.  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:app="http://schemas.android.com/apk/res-auto"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      tools:context=".activity.MainActivity">
5.      <item android:id="@+id/action_settings"
6.          android:title="@string/action_settings"
7.          android:orderInCategory="100"
8.          app:showAsAction="never"/>
9.      <item android:id="@+id/fragment1"
10.         android:title="@string/fragment1"
11.         android:orderInCategory="100"
12.         app:showAsAction="never"/>
13.      <item android:id="@+id/fragment2"
14.         android:title="@string/fragment2"
15.         android:orderInCategory="100"
16.         app:showAsAction="never"/>
17.      <item android:id="@+id/fragment3"
18.         android:title="@string/fragment3"
19.         android:orderInCategory="100"
20.         app:showAsAction="never"/>
21.      <item android:id="@+id/fragment4"
22.         android:title="@string/fragment4"
23.         android:orderInCategory="100"
24.         app:showAsAction="never"/>
25.  </menu>

```

- on supprimera les lignes 5-24. On laisse ainsi une option qu'on n'utilisera pas. Simplement pour avoir un exemple de déclaration d'une option de menu qu'on pourra reproduire par copier / coller ;

Dans la classe [MainActivity] on supprime tout ce qui fait référence aux onglets, au bouton flottant, à la barre d'outils et au menu. Pour trouver ces références, le plus simple est de supprimer leur déclaration :

```

1.  // le gestionnaire d'onglets
2.  @ViewById(R.id.tabs)
3.  protected TabLayout tabLayout;
4.  // le bouton flottant
5.  @ViewById(R.id.fab)
6.  protected FloatingActionButton fab;

```

et de recompiler l'application. Les lignes erronées sont celles qui font référence aux éléments disparus. Supprimez alors toutes ces lignes. Par ailleurs, modifiez le gestionnaire de fragments pour qu'il ne fasse plus référence au fragment [PlaceholderFragment] que nous avons supprimé :

```

1.  public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.      // les fragments
4.      private AbstractFragment[] fragments;
5.
6.      // constructeur
7.      public SectionsPagerAdapter(FragmentManager fm) {
8.          // parent
9.          super(fm);
10.     }
11.
12.     // fragment n° position
13.     @Override
14.     public AbstractFragment getItem(int position) {
15.         // log
16.         if (IS_DEBUG_ENABLED) {
17.             Log.d("SectionsPagerAdapter", String.format("getItem[%s]", position));
18.         }
19.         return fragments[position];
20.     }
21.
22.     // rend le nombre de fragments générés
23.     @Override

```

```

24.     public int getCount() {
25.         return fragments.length;
26.     }
27. }

```

- lignes 7-10 : on a supprimé toute la génération des fragments ;

A ce stade, il ne doit plus y avoir d'erreur de compilation. Dans la classe [MainActivity], on est arrivé au code intermédiaire suivant :

```

1. package exemples.android.activity;
2.
3. import android.os.Bundle;
4. import android.support.v4.app.FragmentManager;
5. import android.support.v4.app.FragmentPagerAdapter;
6. import android.support.v7.app.AppCompatActivity;
7. import android.util.Log;
8. import exemples.android.R;
9. import exemples.android.architecture.AbstractFragment;
10. import exemples.android.architecture.IMainActivity;
11. import exemples.android.architecture.MyPagerAdapter;
12. import exemples.android.architecture.Session;
13. import exemples.android.fragments.Vue1Fragment_;
14. import org.androidannotations.annotations.*;
15.
16. @EActivity(R.layout.activity_main)
17. public class MainActivity extends AppCompatActivity implements IMainActivity {
18.
19.     // le conteneur de fragments
20.     @ViewById(R.id.container)
21.     protected MyPagerAdapter mViewPagerAdapter;
22.     // la barre d'outils
23.     @ViewById(R.id.toolbar)
24.     protected Toolbar toolbar;
25.
26.     // injection session
27.     @Bean(Session.class)
28.     protected Session session;
29.
30.     // nombre de fragments
31.     private final int FRAGMENTS_COUNT = 5;
32.     // adjacence des fragments
33.     private final int OFF_SCREEN_PAGE_LIMIT = 2;
34.
35.     // mode debug
36.     public static final boolean IS_DEBUG_ENABLED = true;
37.
38.     // le gestionnaire de fragments
39.     private SectionsPagerAdapter mSectionsPagerAdapter;
40.
41.     // constructeur
42.     public MainActivity() {
43.         // log
44.         if (IS_DEBUG_ENABLED) {
45.             Log.d("MainActivity", "constructor");
46.         }
47.     }
48.
49.     @AfterViews
50.     protected void afterViews() {
51.         // log
52.         if (IS_DEBUG_ENABLED) {
53.             Log.d("MainActivity", "afterViews");
54.         }
55.
56.         // barre d'outils - c'est là qu'est affiché le nom de 'application'
57.         setSupportActionBar(toolbar);
58.
59.         // le gestionnaire de fragments
60.         mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
61.
62.         // le conteneur de fragments est associé au gestionnaire de fragments
63.         // c-à-d que le fragment n° i du conteneur de fragments est le fragment n° i délivré par le gestionnaire de fragments
64.         mViewPagerAdapter.setAdapter(mSectionsPagerAdapter);
65.
66.         // offset des fragments
67.         mViewPagerAdapter.setOffscreenPageLimit(OFF_SCREEN_PAGE_LIMIT);
68.
69.         // on inhibe le swipe entre fragments
70.         mViewPagerAdapter.setSwipeEnabled(false);
71.
72.         // pas de scrolling
73.         mViewPagerAdapter.setScrollingEnabled(false);
74.
75.         // affichage Vue1
76.         navigateToView(FRAGMENTS_COUNT - 1);

```

```

77.
78.    }
79.
80.    @AfterInject
81.    protected void afterInject() {
82.        // log
83.        if (IS_DEBUG_ENABLED) {
84.            Log.d("MainActivity", "afterInject");
85.        }
86.    }
87.
88.    // getter session
89.    public Session getSession() {
90.        return session;
91.    }
92.
93.    @Override
94.    public void navigateToView(int position) {
95.        // on affiche la vue position
96.        if (mViewPager.getCurrentItem() != position) {
97.            // affichage fragment
98.            mViewPager.setCurrentItem(position);
99.        }
100.    }
101.
102.   // le gestionnaire de fragments
103.   // c'est à lui qu'on demande les fragments à afficher dans la vue principale
104.   // doit définir les méthodes [getItem] et [getCount] - les autres sont facultatives
105.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
106.
107.       // les fragments
108.       private AbstractFragment[] fragments;
109.
110.      // constructeur
111.      public SectionsPagerAdapter(FragmentManager fm) {
112.          // parent
113.          super(fm);
114.      }
115.
116.      // fragment n° position
117.      @Override
118.      public AbstractFragment getItem(int position) {
119.          // log
120.          if (IS_DEBUG_ENABLED) {
121.              Log.d("SectionsPagerAdapter", String.format("getItem[%s]", position));
122.          }
123.          return fragments[position];
124.      }
125.
126.      // rend le nombre de fragments générés
127.      @Override
128.      public int getCount() {
129.          return fragments.length;
130.      }
131.  }
132. }
```

Il reste quelques modifications à faire :

- supprimez la ligne 31 qui n'a plus lieu d'être ;
- ligne 33 : mettez 1 comme adjacence de fragments ;
- ligne 76 : naviguez vers la vue 0. Ce sera elle qui sera affichée la première ;
- ligne 108 : initialisez le tableau avec le fragment [Vue1Fragment_] :

```

1.     // les fragments
2.     private AbstractFragment[] fragments = new AbstractFragment[]{new Vue1Fragment_();}
```

On n'a donc qu'un seul fragment. Exécutez l'application. Vous devez obtenir le résultat suivant :

Vue n° 1

Quel est votre nom ?

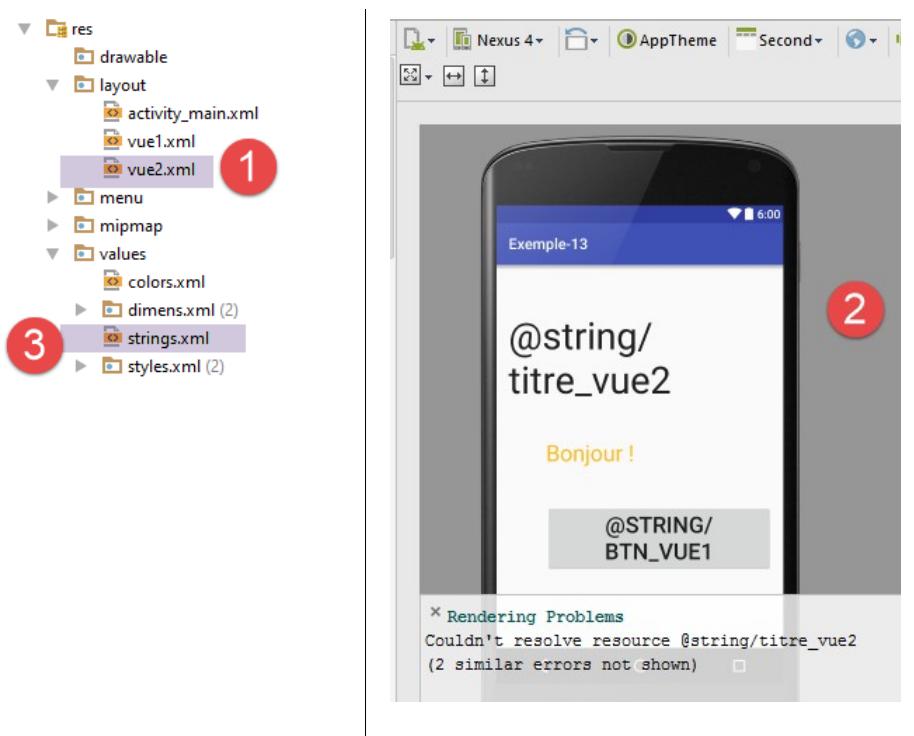
VALIDER

VUE N° 2

Le bouton [Valider] doit fonctionner.

1.14.4 Création des fragments et des vues associées

L'application aura deux vues, celles du projet [Exemple-05]. Nous avons déjà la vue [vue1.xml] dans le projet présent. Nous dupliquons maintenant [vue2.xml] de [Exemple-05] vers [Exemple-12] (ouvrez les deux projets et faites du copier / coller) entre eux.



- en [1], la nouvelle vue. Lorsqu'on essaie de l'éditer, des erreurs apparaissent [2]. Il nous faut modifier le fichier [strings.xml] [3] pour y ajouter les chaînes référencées par cette nouvelle vue :

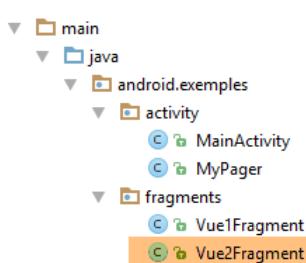
```
1. <resources>
2.   <string name="app_name">Exemple-13</string>
```

```

3. <string name="action_settings">Settings</string>
4. <string name="section_format">Hello World from section: %1$d</string>
5. <!-- vue 1 -->
6. <string name="titre_vue1">Vue n° 1</string>
7. <string name="txt_nom">Quel est votre nom ?</string>
8. <string name="btn_valider">Valider</string>
9. <!-- vue 2 -->
10. <string name="titre_vue2">Vue n° 2</string>
11. <string name="titre_vue2">Vue n° 2</string>
12. <string name="btn_vue1">Vue n° 1</string>
13. </resources>

```

Nous dupliquons la classe [Vue1Fragment] dans [Vue2Fragment] :



et nous modifions le code copié de la façon suivante :

```

1. package exemples.android.fragments;
2.
3. import android.util.Log;
4. import exemples.android.R;
5. import exemples.android.architecture.AbstractFragment;
6. import org.androidannotations.annotations.AfterViews;
7. import org.androidannotations.annotations.EFragment;
8.
9. @EFragment(R.layout.vue2)
10. public class Vue2Fragment extends AbstractFragment {
11.
12.     @AfterViews
13.     protected void afterViews() {
14.         // mémoire
15.         afterViewsDone = true;
16.         // log
17.         if (isDebugEnabled) {
18.             Log.d("Vue2Fragment", String.format("afterViews %s", getParentInfos()));
19.         }
20.     }
21.
22.     // mise à jour fragment
23.     @Override
24.     protected void updateFragment() {
25.     }
26. }

```

- ligne 9 : le fragment est associé à la vue [res / layout / vue2.xml] ;
- ligne 10 : la classe étend la classe abstraite [AbstractFragment] ;
- lignes 12-20 : la méthode [@AfterViews] obligatoire ;
- lignes 23-25 : la méthode [updateFragment] obligatoire ;

1.14.5 Mise en place des fragments et de la navigation entre-eux

L'activité va gérer désormais deux fragments. Sa classe [SectionsPagerAdapter] évolue comme suit :

```

1. public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.     // les fragments
4.     private AbstractFragment[] fragments = new AbstractFragment[]{new Vue1Fragment_(), new Vue2Fragment_()};
5.
6.     ...
7. }

```

L'interface [IMainActivity] assure la navigation entre vues avec sa méthode [navigateToView]. Nous allons gérer le clic sur le bouton [Vue n° 2] du fragment [Vue1Fragment] :

```

1. package exemples.android.fragments;
2.
3. import android.util.Log;
4. import android.widget.EditText;
5. import android.widget.Toast;
6. import exemples.android.R;
7. import exemples.android.architecture.AbstractFragment;
8. import org.androidannotations.annotations.AfterViews;
9. import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.ViewById;
12.
13. @EFragment(R.layout.vue1)
14. public class Vue1Fragment extends AbstractFragment {
15.
16.     // les éléments de l'interface visuelle
17.     @ViewById(R.id.editTextNom)
18.     protected EditText editTextNom;
19.
20.     @AfterViews
21.     protected void afterViews() {
22.         // mémoire
23.         afterViewsDone = true;
24.         // log
25.         if (isDebugEnabled) {
26.             Log.d("Vue1Fragment", String.format("afterViews %s", getParentInfos()));
27.         }
28.     }
29.
30.     // gestionnaires d'évts -----
31.     @Click(R.id.buttonValider)
32.     protected void doValider() {
33.         // on affiche le nom saisi
34.         Toast.makeText(activity, String.format("Bonjour %s", editTextNom.getText().toString()), Toast.LENGTH_LONG).show();
35.     }
36.
37.     @Click(R.id.buttonVue2)
38.     protected void showVue2() {
39.         mainActivity.navigateToView(1);
40.     }
41.
42.     // mise à jour fragment
43.     @Override
44.     protected void updateFragment() {
45.     }
46. }
```

- lignes 37-40 : la méthode [showVue2] gère l'événement 'clic' sur le bouton [Vue n° 2] ;
- ligne 39 : on navigue avec la méthode [navigateToView] de l'activité. On rappelle ici que l'activité a été mémorisée dans la classe parent sous la forme :

```

1.     // activité
2.     protected IMainActivity mainActivity;
```

et que cette activité a déjà été initialisée lorsqu'on arrive dans un quelconque gestionnaire d'événement.

- ligne 34 : l'instruction utilise la variable [activity] de la classe parent qui est une référence de l'activité en tant qu'instance du type Android [Activity] ;

```
1.     protected Activity activity;
```

On retrouve un code analogue pour le fragment [Vue2Fragment] :

```

1. package exemples.android.fragments;
2.
3. import android.util.Log;
4. import exemples.android.R;
5. import exemples.android.architecture.AbstractFragment;
6. import org.androidannotations.annotations.AfterViews;
7. import org.androidannotations.annotations.Click;
8. import org.androidannotations.annotations.EFragment;
9.
10. @EFragment(R.layout.vue2)
11. public class Vue2Fragment extends AbstractFragment {
12.
13.     @AfterViews
14.     protected void afterViews() {
15.         // mémoire
16.         afterViewsDone = true;
17.         // log
18.         if (isDebugEnabled) {
19.             Log.d("Vue2Fragment", String.format("afterViews %s", getParentInfos()));
```

```

20.      }
21.    }
22.
23.    // gestionnaires d'évts -----
24.    @Click(R.id.buttonVue1)
25.    protected void showVue1() {
26.      mainActivity.navigateToView(0);
27.    }
28.
29.    // mise à jour fragment
30.    @Override
31.    protected void updateFragment() {
32.    }
33.  }

```

- lignes 24-27 : la méthode [showVue1] gère l'événement 'clic' sur le bouton [Vue n° 1] ;

Exécutez le projet et vérifiez que la navigation entre vues fonctionne.

1.14.6 Définition de la session

Le fonctionnement de l'application est le suivant :

- saisie d'un nom dans la vue n° 1 ;
- affichage de ce nom dans la vue n° 2 ;

Pour que la vue n° 1 puisse communiquer le nom saisi à la vue n° 2, nous utiliserons la session suivante ;

```

1. package exemples.android.architecture;
2.
3. import org.androidannotations.annotations.EBean;
4.
5. @EBean(scope = EBean.Scope.Singleton)
6. public class Session {
7.   // nom
8.   private String nom;
9.
10.  // getters et setters
11. ...
12. }

```

- ligne 8 : le nom saisi ;

La classe [MainActivity] va initialiser la session de la façon suivante :

```

1.  // injection session
2.  @Bean(Session.class)
3.  protected Session session;
4. ...
5.  @AfterInject
6.  protected void afterInject() {
7.    // log
8.    if (IS_DEBUG_ENABLED) {
9.      Log.d("MainActivity", "afterInject");
10.    }
11.   // init session
12.   session.setNom("");
13. }

```

1.14.7 Ecriture finale des fragments

Dans le fragment [Vue1Fragment] nous modifions le code du gestionnaire du clic sur le bouton [Valider] :

```

1. package exemples.android.fragments;
2.
3. import android.util.Log;
4. import android.widget.EditText;
5. import android.widget.Toast;
6. import exemples.android.R;
7. import exemples.android.architecture.AbstractFragment;
8. import org.androidannotations.annotations.AfterViews;
9. import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.ViewById;
12.
13. @EFragment(R.layout.vue1)
14. public class Vue1Fragment extends AbstractFragment {
15.
16.   // les éléments de l'interface visuelle
17.   @ViewById(R.id.editTextNom)

```

```

18.    protected EditText editTextNom;
19.
20.    ...
21.    // gestionnaires d'évts -----
22.
23.    @Click(R.id.buttonValider)
24.    protected void doValider() {
25.        // on mémorise le nom saisi
26.        String nom = editTextNom.getText().toString();
27.        // on l'affiche
28.        Toast.makeText(activity, nom, Toast.LENGTH_LONG).show();
29.    }
30.
31.    @Click(R.id.buttonVue2)
32.    protected void showVue2() {
33.        // on met le nom saisi dans la session
34.        session.setNom(editTextNom.getText().toString());
35.        // on navigue vers la vue n° 2
36.        MainActivity.navigateToView(1);
37.    }
38.
39.    // mise à jour fragment
40.    @Override
41.    protected void updateFragment() {
42.
43.    }
44. }
```

- lignes : 31-37 : on gère le clic sur le bouton [Vue n° 2] ;
- ligne 34 : avant de naviguer vers la vue n° 2, on met le nom saisi dans la session pour que la nouvelle vue y ait accès ;

La vue [Vue2Fragment] évolue de la façon suivante :

```

1.  package exemples.android.fragments;
2.
3.  import android.util.Log;
4.  import android.widget.TextView;
5.  import exemples.android.R;
6.  import exemples.android.architecture.AbstractFragment;
7.  import org.androidannotations.annotations.AfterViews;
8.  import org.androidannotations.annotations.Click;
9.  import org.androidannotations.annotations.EFragment;
10. import org.androidannotations.annotations.ViewById;
11.
12. @EFragment(R.layout.vue2)
13. public class Vue2Fragment extends AbstractFragment {
14.
15.     // composants de l'interface visuelle
16.     @ViewById(R.id.textViewBonjour)
17.     protected TextView textViewBonjour;
18.
19.     @AfterViews
20.     protected void afterViews() {
21.         // mémoire
22.         afterViewsDone = true;
23.         // log
24.         if (isDebugEnabled) {
25.             Log.d("Vue2Fragment", String.format("afterViews %s", getParentInfos()));
26.         }
27.     }
28.
29.     // gestionnaires d'évts -----
30.     @Click(R.id.buttonVue1)
31.     protected void showVue1() {
32.         MainActivity.navigateToView(0);
33.     }
34.
35.     // mise à jour fragment
36.     @Override
37.     protected void updateFragment() {
38.         // on récupère le nom saisi dans la session
39.         String nom = session.getNom();
40.         // on l'affiche
41.         textViewBonjour.setText(String.format("Bonjour %s !", nom));
42.     }
43. }
```

Lorsque la vue n° 2 s'affiche, il faut afficher le nom saisi dans la vue n° 1. On sait que juste après son affichage, sa méthode [updateFragment] va être exécutée. C'est donc dans cette méthode (lignes 36-42) qu'on peut mettre le code d'affichage du nom.

- lignes 16-17 : déclaration du seul composant visuel de la vue ;
- ligne 39 : le nom saisi dans la vue n°1 est récupéré dans la session ;

- ligne 41 : le libellé [textViewBonjour] est modifié ;

Exécutez le projet et vérifiez qu'il fonctionne.

1.14.8 Gestion du cycle de vie des fragments

Dans le fragment [Vue1Fragment], la méthode [@AfterViews] est la suivante :

```

1.  @AfterViews
2.  protected void afterViews() {
3.      // mémoire
4.      afterViewsDone = true;
5.      // log
6.      if (isDebugEnabled) {
7.          Log.d("Vue1Fragment", String.format("afterViews %s", getParentInfos()));
8.      }
9.  }
```

Cette méthode est incomplète. En effet, il faut toujours prévoir le cas où le fragment est recyclé après une opération [onDestroyView]. Dans ce cas, la vue du fragment 1 est régénérée et le nom qui a pu être saisi précédemment va disparaître de la vue. On ne veut pas ça. Actuellement le nom saisi reste affiché parce que l'adjacence des fragments de 1 fait que le cycle de vie du fragment [Vue1Fragment] n'est exécuté qu'une fois. Il est cependant préférable de prévoir le cas du recyclage du fragment.

Il y a plusieurs façons de résoudre ce problème :

- on peut profiter du fait que la méthode [update] soit exécutée systématiquement à chaque affichage du fragment pour mettre à jour le nom saisi ;
- on peut faire cette mise à jour uniquement lorsque la méthode [@AfterViews] est réexécutée. C'est cette dernière voie que nous prenons ;

On modifie le code de [Vue1Fragment] de la façon suivante :

```

1.  // les éléments de l'interface visuelle
2.  @ViewById(R.id.editTextNom)
3.  protected EditText editTextNom;
4.
5.  // data
6.  private String nom;
7.
8.  @AfterViews
9.  protected void afterViews() {
10.     // mémoire
11.     afterViewsDone = true;
12.     // log
13.     if (isDebugEnabled) {
14.         Log.d("Vue1Fragment", String.format("afterViews %s", getParentInfos()));
15.     }
16.     // on (ré)initialise le texte affiché
17.     editTextNom.setText(nom);
18. }
19.
20. // gestionnaires d'évts -----
21.
22. ...
23.
24. @Click(R.id.buttonVue2)
25. protected void showVue2() {
26.     // on note le nom saisi pour pouvoir le récupérer si le fragment est recyclé
27.     nom = editTextNom.getText().toString();
28.     // on met le nom saisi dans la session
29.     session.setNom(nom);
30.     // on navigue vers la vue n° 2
31.     activity.navigateToView(1);
32. }
```

- ligne 27 : alors qu'on s'apprête à quitter la vue 1 pour la vue 2, on mémorise le nom saisi ;
- ligne 17 : à chaque nouvelle exécution du cycle de vie du fragment, le dernier nom saisi est réaffiché ;

Pour le fragment [Vue2Fragment], le code existant suffit :

```

1.  // composants de l'interface visuelle
2.  @ViewById(R.id.textViewBonjour)
3.  protected TextView textViewBonjour;
4.
5.  @AfterViews
6.  protected void afterViews() {
7.     // mémoire
```

```

8.     afterViewsDone = true;
9.     // log
10.    if (isDebugEnabled) {
11.        Log.d("Vue2Fragment", String.format("afterViews %s", getParentInfos()));
12.    }
13. }
14.
15. // mise à jour fragment
16. @Override
17. protected void updateFragment() {
18.     // on récupère le nom saisi dans la session
19.     String nom = session.getNom();
20.     // on l'affiche
21.     textViewBonjour.setText(String.format("Bonjour %s !", nom));
22. }

```

- le seul composant visuel de la vue (ligne 3) est mis à jour à chaque fois que la vue est affichée (ligne 21). La méthode [`@AfterViews`] n'a donc rien à ajouter ;

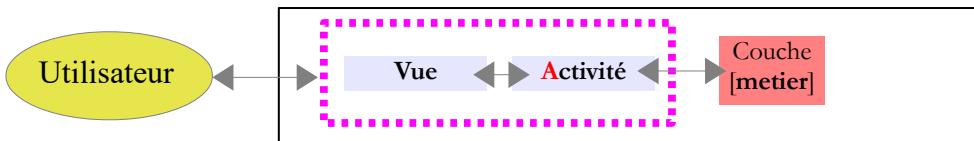
1.14.9 Conclusion

A ce point, nous avons de nouveau montré la pertinence de notre architecture :

- une activité implémentant l'interface [`IMainActivity`] ;
- des fragments étendant la classe [`AbstractFragment`], ce qui leur impose d'implémenter la méthode [`updateFragment`]. Ceux-ci doivent avoir également une méthode [`@AfterViews`] dans laquelle elles mettent le booléen [`afterViewsDone`] à `true` ;
- une session encapsulant les données à partager entre fragments et activité ;

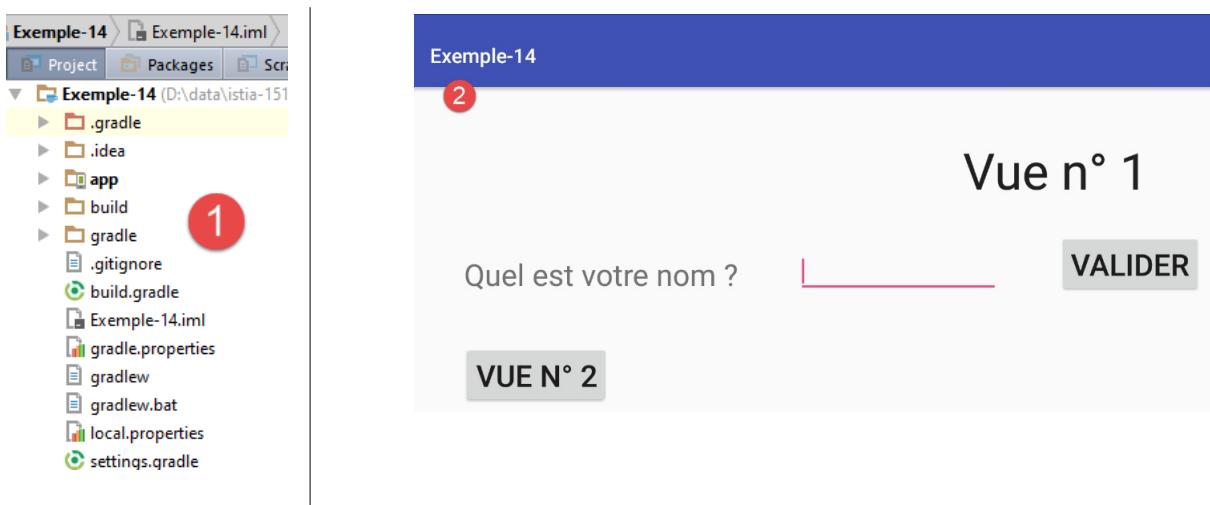
1.15 Exemple-14 : une architecture à deux couches

Nous allons construire une application à une vue ayant l'architecture suivante :



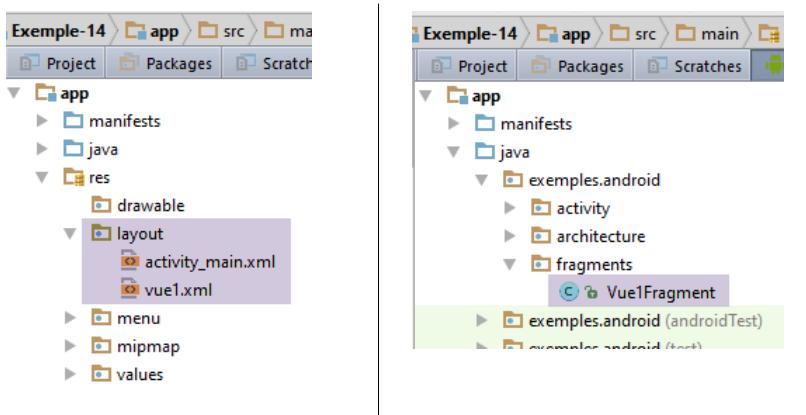
1.15.1 Crédation du projet

Nous dupliquons le projet précédent [Exemple-12] dans [Exemple-13] en suivant la procédure du paragraphe 1.4, page 36. Nous obtenons le résultat suivant :

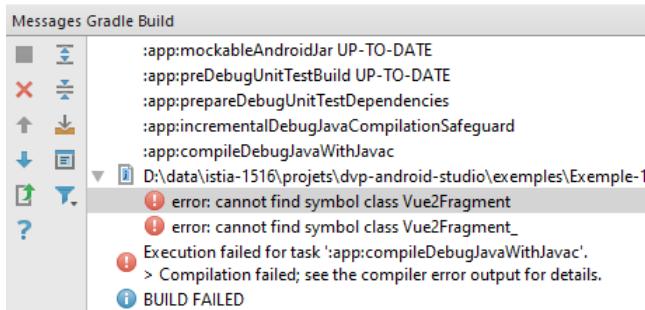


1.15.2 La vue [vue1]

L'application n'aura qu'une vue [vue1.xml]. Aussi supprimons-nous l'autre vue [vue2.xml] ainsi que son fragment associé :



Compilez l'application. Des erreurs apparaissent dans [MainActivity] :



Corrigez la ligne 4 ci-dessous dans le gestionnaire de fragments [SectionsPagerAdapter]

```
1.     public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.     // les fragments
4.     private AbstractFragment[] fragments = new AbstractFragment[]{new Vue1Fragment_(), new Vue2Fragment_()};
5.     ...
```

La ligne 4 ci-dessus devient :

```
1.     // les fragments
2.     private AbstractFragment[] fragments = new AbstractFragment[]{new Vue1Fragment_()};
```

Supprimez les imports devenus inutiles [Ctrl-Shift-O]. Il ne doit plus y avoir d'erreur de compilation. Exécutez le projet : la vue n° 1 doit apparaître. Nous allons maintenant modifier celle-ci.

Nous allons créer la vue [vue1.xml] qui permettra de générer des nombres aléatoires :

Exemple-14

Génération de N nombres aléatoires

Valeur de N : 10 1

Intervalle [a,b] de génération, a : 300 2 b : 400 3

EXÉCUTER 4

Liste des réponses

394	5
369	
364	
321	
Exception aléatoire	

Ses composants sont les suivants :

N°	Id	Type	Rôle
1	edtNbAleas	EditText	nombre de nombres aléatoires à générer dans l'intervalle entier [a,b]
2	edtA	EditText	valeur de a

```
2 edtB          EditText      valeur de b
4 btnExécuter   Button       lance la génération des nombres
5 ListView      lstReponses liste des nombres générés dans l'ordre inverse de leur génération. On voit d'abord le dernier
                     générée ;
```

Son code XML est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      android:id="@+id/RelativeLayout1"
5.      android:layout_width="match_parent"
6.      android:layout_height="match_parent"
7.      android:layout_marginLeft="20dp"
8.      android:orientation="vertical" >
9.
10. <TextView
11.     android:id="@+id/txt_Titre2"
12.     android:layout_width="wrap_content"
13.     android:layout_height="wrap_content"
14.     android:layout_marginTop="20dp"
15.     android:text="@string/aleas"
16.     android:textAppearance="?android:attr/textAppearanceLarge" />
17.
18. <TextView
19.     android:id="@+id/txt_nbAleas"
20.     android:layout_width="wrap_content"
21.     android:layout_height="wrap_content"
22.     android:layout_below="@+id/txt_Titre2"
23.     android:layout_marginTop="20dp"
24.     android:text="@string/txt_nbaleas" />
25.
26. <EditText
27.     android:id="@+id/edt_nbaleas"
28.     android:layout_width="wrap_content"
29.     android:layout_height="wrap_content"
30.     android:layout_alignBaseline="@+id/edt_nbaleas"
31.     android:layout_marginLeft="20dp"
32.     android:layout_toRightOf="@+id/txt_nbaleas"
33.     android:inputType="number" />
34.
35. <TextView
36.     android:id="@+id/txt_errorNbAleas"
37.     android:layout_width="wrap_content"
38.     android:layout_height="wrap_content"
39.     android:layout_alignBaseline="@+id/edt_nbaleas"
40.     android:layout_marginLeft="20dp"
41.     android:layout_toRightOf="@+id/edt_nbaleas"
42.     android:text="@string/txt_errorNbAleas"
43.     android:textColor="@color/red" />
44.
45. <TextView
46.     android:id="@+id/txt_a"
47.     android:layout_width="wrap_content"
48.     android:layout_height="wrap_content"
49.     android:layout_below="@+id/txt_nbaleas"
50.     android:layout_marginTop="20dp"
51.     android:text="@string/txt_a" />
52.
53. <EditText
54.     android:id="@+id/edt_a"
55.     android:layout_width="wrap_content"
56.     android:layout_height="wrap_content"
57.     android:layout_alignBaseline="@+id/txt_a"
58.     android:layout_marginLeft="20dp"
59.     android:layout_toRightOf="@+id/txt_a"
60.     android:inputType="number" />
61.
62. <TextView
63.     android:id="@+id/txt_b"
64.     android:layout_width="wrap_content"
65.     android:layout_height="wrap_content"
66.     android:layout_alignBaseline="@+id/txt_a"
67.     android:layout_marginLeft="20dp"
68.     android:layout_toRightOf="@+id/edt_a"
69.     android:text="@string/txt_b" />
70.
71. <EditText
72.     android:id="@+id/edt_b"
73.     android:layout_width="wrap_content"
74.     android:layout_height="wrap_content"
75.     android:layout_alignBaseline="@+id/txt_a"
76.     android:layout_marginLeft="20dp"
77.     android:layout_toRightOf="@+id/txt_b"
78.     android:inputType="number" />
79.
```

```

80.    <TextView
81.        android:id="@+id/txt_errorIntervalle"
82.        android:layout_width="wrap_content"
83.        android:layout_height="wrap_content"
84.        android:layout_alignBaseline="@+id/edt_b"
85.        android:layout_marginLeft="20dp"
86.        android:layout_toRightOf="@+id/edt_b"
87.        android:text="@string/txt_errorIntervalle"
88.        android:textColor="@color/red" />
89.
90.
91.    <Button
92.        android:id="@+id	btn_Executer"
93.        android:layout_width="wrap_content"
94.        android:layout_height="wrap_content"
95.        android:layout_alignParentLeft="true"
96.        android:layout_below="@+id/txt_a"
97.        android:layout_marginTop="20dp"
98.        android:text="@string/btn_executer" />
99.
100.
101.   <TextView
102.       android:id="@+id/txt_Reponses"
103.       android:layout_width="wrap_content"
104.       android:layout_height="wrap_content"
105.       android:layout_below="@+id/button1"
106.       android:layout_marginTop="30dp"
107.       android:text="@string/list_reponses"
108.       android:textAppearance="?android:attr/textAppearanceLarge"
109.       android:textColor="@color/blue" />
110.
111.   <ListView
112.       android:id="@+id/Lst_reponses"
113.       android:layout_width="match_parent"
114.       android:layout_height="match_parent"
115.       android:layout_alignParentLeft="true"
116.       android:layout_below="@+id/txt_Reponses"
117.       android:layout_marginTop="40dp"
118.       android:background="@color/wheat"
119.       android:clickable="true"
120.       tools:listitem="@android:layout/simple_list_item_1" >
121.   </ListView>
122.
123. </RelativeLayout>

```

La vue précédente utilise des libellés définis dans le fichier [res / values / **strings.xml**] :

```

1.  <resources>
2.      <string name="app_name">Exemple-14</string>
3.      <string name="action_settings">Settings</string>
4.      <string name="section_format">Hello World from section: %1$d</string>
5.      <!-- vue 1 -->
6.      <string name="titre_vue1">Vue n° 1</string>
7.      <string name="list_reponses">Liste des réponses</string>
8.      <string name="btn_executer">Exécuter</string>
9.      <string name="aleas">Génération de N nombres aléatoires</string>
10.     <string name="txt_nbAleas">Valeur de N :</string>
11.     <string name="txt_a">"Intervalle [a,b] de génération, a : "</string>
12.     <string name="txt_b">"b : "</string>
13.     <string name="txt_dummy">Dummy</string>
14.     <string name="txt_errorNbAleas">Tapez un nombre entier >=1</string>
15.     <string name="txt_errorIntervalle">Les bornes de l'intervalle doivent être entières et b>=a</string>
16. </resources>

```

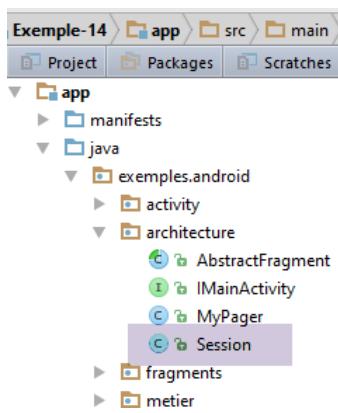
Les couleurs utilisées dans [vue1.xml] sont définies dans le fichier [res / values / **colors.xml**] :

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.      <color name="colorPrimary">#3F51B5</color>
4.      <color name="colorPrimaryDark">#303F9F</color>
5.      <color name="colorAccent">#FF4081</color>
6.      <!-- couleurs appli -->
7.      <color name="red">#FF0000</color>
8.      <color name="blue">#0000FF</color>
9.      <color name="wheat">#FFEFD5</color>
10.     <color name="floral_white">#FFF8AF0</color>
11. </resources>

```

1.15.3 La session

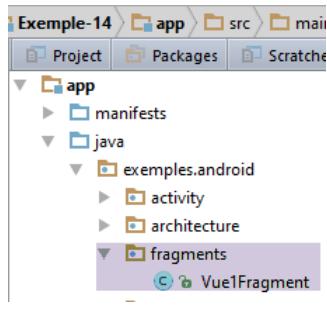


Comme ici, il n'y a qu'un fragment, il n'y a pas à prévoir de communication inter-fragments. La session sera donc vide :

```
1. package exemples.android.architecture;
2.
3. import org.androidannotations.annotations.EBean;
4.
5. @EBean(scope = EBean.Scope.Singleton)
6. public class Session {
7. }
```

A ce stade, compilez l'application. Des erreurs apparaissent sur les lignes qui utilisaient la session qui vient d'être supprimée. Supprimez ces lignes et vérifiez que la compilation ne produit plus d'erreurs.

1.15.4 Le fragment [Vue1Fragment]



Nous modifions le fragment [Vue1Fragment] existant de la façon suivante :

```
1. package exemples.android.fragments;
2.
3. import android.util.Log;
4. import android.view.View;
5. import android.widget.ArrayAdapter;
6. import android.widget.EditText;
7. import android.widget.ListView;
8. import android.widget.TextView;
9. import exemples.android.R;
10. import exemples.android.architecture.AbstractFragment;
11. import org.androidannotations.annotations.AfterViews;
12. import org.androidannotations.annotations.Click;
13. import org.androidannotations.annotations.EFragment;
14. import org.androidannotations.annotations.ViewById;
15.
16. import java.util.ArrayList;
17. import java.util.List;
18.
19. @EFragment(R.layout.vue1)
20. public class Vue1Fragment extends AbstractFragment {
21.
22.     // les éléments de l'interface visuelle
```

```

23.    @ViewById(R.id.lst_reponses)
24.    protected ListView listReponses;
25.    @ViewById(R.id.edt_nbAleas)
26.    protected EditText edtNbAleas;
27.    @ViewById(R.id.edt_a)
28.    protected EditText edtA;
29.    @ViewById(R.id.edt_b)
30.    protected EditText edtB;
31.    @ViewById(R.id.txt_errorNbAleas)
32.    protected TextView txtErrorAleas;
33.    @ViewById(R.id.txt_errorIntervalle)
34.    protected TextView txtErrorIntervalle;
35.
36.    // liste des réponses à une commande
37.    private List<String> reponses = new ArrayList<>();
38.    // adaptateur du listview
39.    private ArrayAdapter<String> adapterReponses;
40.
41.    // les saisies
42.    private int nbAleas;
43.    private int a;
44.    private int b;
45.
46.    @AfterViews
47.    protected void afterViews() {
48.        // mémoire
49.        afterViewsDone = true;
50.        // log
51.        if (isDebugEnabled) {
52.            Log.d("Vue1Fragment", String.format("afterViews %s", getParentInfos()));
53.        }
54.        // on cache les messages d'erreur
55.        txtErrorAleas.setVisibility(View.INVISIBLE);
56.        txtErrorIntervalle.setVisibility(View.INVISIBLE);
57.    }
58.
59.    @Click(R.id.btn_Executer)
60.    void doExecuter() {
61.        // on cache les éventuels msg d'erreur précédents
62.        txtErrorAleas.setVisibility(View.INVISIBLE);
63.        txtErrorIntervalle.setVisibility(View.INVISIBLE);
64.        // on teste la validité des saisies
65.        if (!isPageValid()) {
66.            return;
67.        }
68.    }
69.
70.    // on vérifie la validité des données saisies
71.    private boolean isPageValid() {
72.        ...
73.    }
74.
75.    @Override
76.    protected void updateFragment() {
77.        // log
78.        if (isDebugEnabled) {
79.            Log.d("Vue1Fragment", String.format("updateFragment %s", getParentInfos()));
80.        }
81.    }
82. }

```

- il n'y a ici qu'un fragment dont le cycle de vie ne sera exécuté qu'une unique fois, au démarrage de l'application. pour cette raison, les méthodes [`@AfterViews`] (lignes 46-57) et [`updateFragment`] (lignes 75-81) ne seront exécutées qu'une fois au démarrage de l'application ;
- lignes 55-56 : on cache les deux messages d'erreur de la vue (représentés ci-dessous) [1-2] ;

Exemple-14

Génération de N nombres aléatoires

Valeur de N : Tapez un nombre entier >=1 1

Intervalle [a,b] de génération, a : b : Les bornes de l'intervalle doivent être entières et b>=a 2

EXÉCUTER

- lignes 59-60 : la méthode exécutée lors d'un clic sur le bouton [Exécuter] ;
- lignes 71-73 : on vérifie la validité des saisies ;

La méthode [isValidPage] est la suivante :

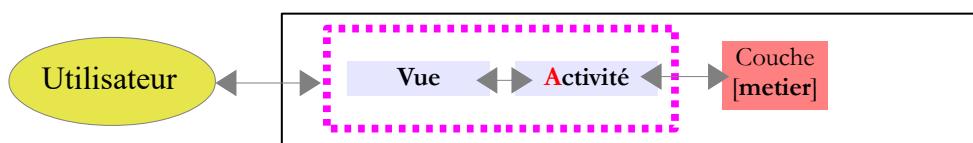
```

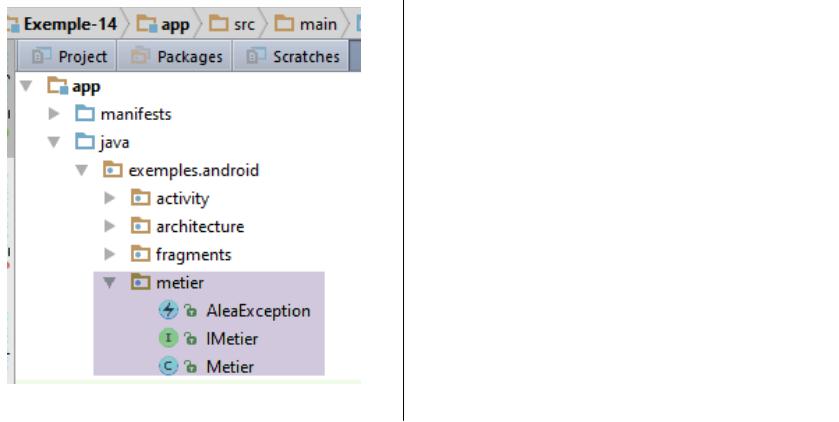
1.  // les saisies
2.  private int nbAleas;
3.  private int a;
4.  private int b;
5.
6. ...
7.
8. // on vérifie la validité des données saisies
9. private boolean isValidPage() {
10.    // saisie du nombre de nombres aléatoires
11.    nbAleas = 0;
12.    Boolean erreur;
13.    int nbErreurs = 0;
14.    try {
15.      nbAleas = Integer.parseInt(edtNbAleas.getText().toString());
16.      erreur = (nbAleas < 1);
17.    } catch (Exception ex) {
18.      erreur = true;
19.    }
20.    // erreur ?
21.    if (erreur) {
22.      nbErreurs++;
23.      txtErrorAleas.setVisibility(View.VISIBLE);
24.    }
25.    // saisie de a
26.    a = 0;
27.    erreur = false;
28.    try {
29.      a = Integer.parseInt(edtA.getText().toString());
30.    } catch (Exception ex) {
31.      erreur = true;
32.    }
33.    // erreur ?
34.    if (erreur) {
35.      nbErreurs++;
36.      txtErrorIntervalle.setVisibility(View.VISIBLE);
37.    }
38.    // saisie de b
39.    b = 0;
40.    erreur = false;
41.    try {
42.      b = Integer.parseInt(edtB.getText().toString());
43.      erreur = b < a;
44.    } catch (Exception ex) {
45.      erreur = true;
46.    }
47.    // erreur ?
48.    if (erreur) {
49.      nbErreurs++;
50.      txtErrorIntervalle.setVisibility(View.VISIBLE);
51.    }
52.    // retour
53.    return (nbErreurs == 0);
54.  }
55.
```

- lignes 2-4 : ces trois champs sont initialisés par la méthode [isValidPage]. Par ailleurs, cette méthode rend *true* si toutes les saisies sont valides, *false* sinon. Si des saisies sont invalides, alors les messages d'erreur associés sont affichés ;

A ce stade, l'application est exécutable. Vérifiez le fonctionnement de la méthode [isValidPage] en entrant des données incorrectes.

1.15.5 La couche [métier]





La couche [métier] présente l'interface [IMetier] suivante :

```

1. package exemples.android.metier;
2.
3. import java.util.List;
4.
5. public interface IMetier {
6.
7.     List<Object> getAleas(int a, int b, int n);
8. }
```

La méthode [getAleas(a,b,n)] renvoie normalement n nombres entiers aléatoires dans l'intervalle $[a,b]$. On a prévu également qu'elle renvoie une fois sur trois une exception, exception également insérée dans les réponses rendues par la méthode. Au final celle-ci rend une liste d'objets de type [Exception] ou [Integer].

L'implémentation [Metier] de cette interface est la suivante :

```

1. package exemples.android.metier;
2.
3. import org.androidannotations.annotations.EBean;
4.
5. import java.util.ArrayList;
6. import java.util.List;
7. import java.util.Random;
8.
9. @EBean(scope = EBean.Scope.Singleton)
10. public class Metier implements IMetier {
11.
12.     public List<Object> getAleas(int a, int b, int n) {
13.         // la liste des objets
14.         List<Object> réponses = new ArrayList<Object>();
15.         // qqs vérifications
16.         if (n < 1) {
17.             réponses.add(new AleaException("Le nombre d'entier aléatoires demandé doit être supérieur ou égal à 1"));
18.         }
19.         if (a < 0) {
20.             réponses.add(new AleaException("Le nombre a de l'intervalle [a,b] doit être supérieur à 0"));
21.         }
22.         if (b < 0) {
23.             réponses.add(new AleaException("Le nombre b de l'intervalle [a,b] doit être supérieur à 0"));
24.         }
25.         if (a >= b) {
26.             réponses.add(new AleaException("Dans l'intervalle [a,b], on doit avoir a < b"));
27.         }
28.         // erreur ?
29.         if (réponses.size() != 0) {
30.             return réponses;
31.         }
32.         // on génère les nombres aléatoires
33.         Random random = new Random();
34.         for (int i = 0; i < n; i++) {
35.             // on génère une exception aléatoire 1 fois / 3
36.             int nombre = random.nextInt(3);
37.             if (nombre == 0) {
38.                 réponses.add(new AleaException("Exception aléatoire"));
39.             } else {
40.                 // sinon on rend un nombre aléatoire entre deux bornes [a,b]
41.                 réponses.add(Integer.valueOf(a + random.nextInt(b - a + 1)));
42.             }
43.         }
44.     // résultat
45. }
```

```

45.         return réponses;
46.     }
47. }

```

- ligne 9 : on utilise l'annotation AA `[@EBean]` sur la classe `[Metier]` afin de pouvoir injecter des références de celle-ci dans la couche `[Présentation]`. L'attribut `(scope = EBean.Scope.Singleton)` fait que la classe `[Metier]` ne sera instanciée qu'en un seul exemplaire. C'est donc toujours la même référence qui est injectée si on l'injecte plusieurs fois dans la couche `[Présentation]` ;
- le reste du code est classique ;

Le type `[AleaException]` utilisé par la classe `[Metier]` est la suivante :

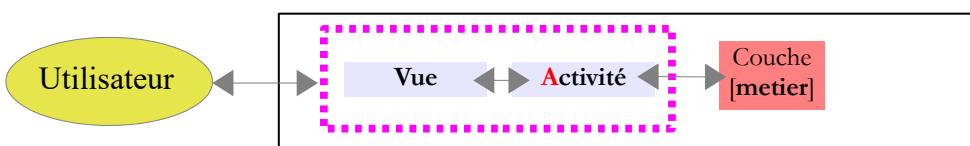
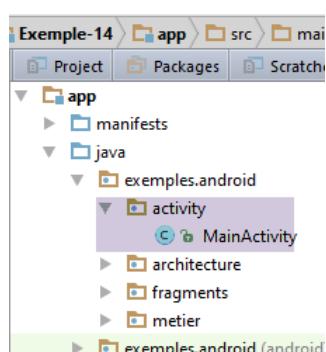
```

1. package exemples.android.metier;
2.
3. public class AleaException extends RuntimeException {
4.
5.     private static final long serialVersionUID = 1L;
6.
7.     public AleaException() {
8.     }
9.
10.    public AleaException(String detailMessage) {
11.        super(detailMessage);
12.    }
13.
14.    public AleaException(Throwable throwable) {
15.        super(throwable);
16.    }
17.
18.    public AleaException(String detailMessage, Throwable throwable) {
19.        super(detailMessage, throwable);
20.    }
21.
22. }

```

- ligne 3 : la classe `[AleaException]` étend la classe système `[RuntimeException]`, ce qui en fait une exception non contrôlée : on n'est pas obligé de la gérer dans un `try / catch`, ni de la mettre dans la signature des méthodes ;

1.15.6 L'activité `[MainActivity]` revisitée

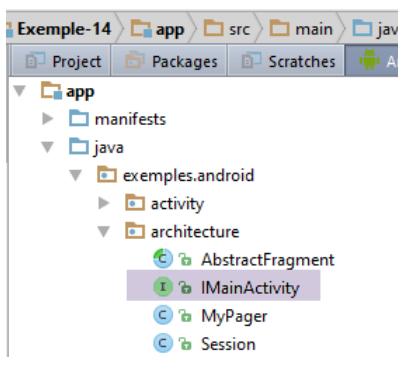


L'activité implémentera l'interface `[IMetier]` de la couche `[métier]`. Ainsi le fragment / vue n'aura-t-il que l'activité comme interlocuteur.

L'activité `[MainActivity]` implémente déjà l'interface `[IMainActivity]`. Pour qu'elle implémente également l'interface `[IMetier]`, on peut :

- ajouter l'interface `[IMetier]` aux interfaces implementées par l'activité ;
- faire en sorte que l'interface `[IMainActivity]` étende elle-même l'interface `[IMetier]`. C'est cette voie que nous prenons ;

L'interface [IMainActivity] devient la suivante :



```
1. package exemples.android.architecture;
2.
3. import exemples.android.metier.IMetier;
4.
5. public interface IMainActivity extends IMetier {
6.
7.     // accès à la session
8.     Session getSession();
9.
10.    // changement de vue
11.    void navigateToView(int position);
12.
13.    // mode debug
14.    public static final boolean IS_DEBUG_ENABLED = true;
15.
16. }
```

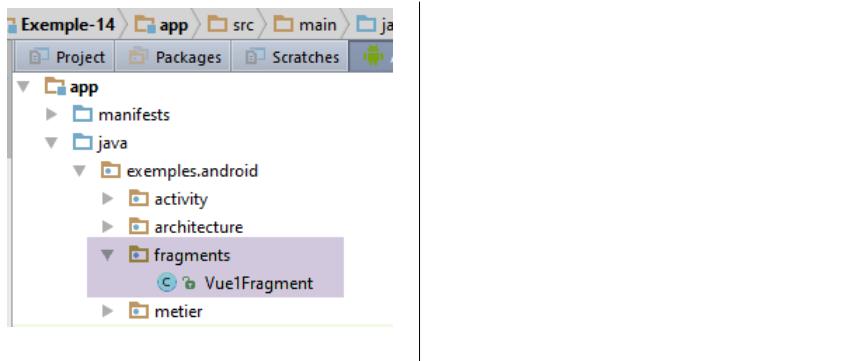
- ligne 5 : l'interface [IMainActivity] étende l'interface [IMetier]

La classe [MainActivity] évolue de la façon suivante :

```
1. @EActivity(R.layout.activity_main)
2. public class MainActivity extends AppCompatActivity implements IMainActivity {
3.
4.     ...
5.
6.     // injection session
7.     @Bean(Session.class)
8.     protected Session session;
9.
10.    // injection métier
11.    @Bean(Metier.class)
12.    protected IMetier metier;
13.
14. ...
15.    // implémentation IMetier -----
16.    @Override
17.    public List<Object> getAleas(int a, int b, int n) {
18.        return metier.getAleas(a, b, n);
19.    }
}
```

- lignes 11-12 : la couche [métier] est injectée dans l'activité. On utilise pour cela l'annotation AA [@Bean] dont le paramètre est la classe portant l'annotation AA [@EBean] ;
- ligne 2 : l'activité implémente l'interface [IMainActivity] et donc l'interface [IMetier] de la couche [métier] ;
- lignes 16-19 : implémentation de l'unique méthode de l'interface [IMetier]. On se contente de déléguer l'appel à la couche [métier] ;

1.15.7 Le fragment [Vue1Fragment] revisité



Le code de la classe [Vue1Fragment] évolue de la façon suivante :

```

1. package exemples.android.fragments;
2.
3. import android.util.Log;
4. import android.view.View;
5. import android.widget.ArrayAdapter;
6. import android.widget.EditText;
7. import android.widget.ListView;
8. import android.widget.TextView;
9. import exemples.android.R;
10. import exemples.android.architecture.AbstractFragment;
11. import org.androidannotations.annotations.AfterViews;
12. import org.androidannotations.annotations.Click;
13. import org.androidannotations.annotations.EFragment;
14. import org.androidannotations.annotations.ViewById;
15.
16. import java.util.ArrayList;
17. import java.util.List;
18.
19. @EFragment(R.layout.vue1)
20. public class Vue1Fragment extends AbstractFragment {
21.
22.     // les éléments de l'interface visuelle
23.     @ViewById(R.id.lst_reponses)
24.     protected ListView listReponses;
25.     @ViewById(R.id.edt_nbaleas)
26.     protected EditText edtNbAleas;
27.     @ViewById(R.id.edt_a)
28.     protected EditText edtA;
29.     @ViewById(R.id.edt_b)
30.     protected EditText edtB;
31.     @ViewById(R.id.txt_errorNbAleas)
32.     protected TextView txtErrorNbAleas;
33.     @ViewById(R.id.txt_errorIntervalle)
34.     protected TextView txtErrorIntervalle;
35.
36.     // liste des réponses à une commande
37.     private List<String> reponses = new ArrayList<>();
38.     // adaptateur du listview
39.     private ArrayAdapter<String> adapterReponses;
40.
41.     // les saisies
42.     private int nbAleas;
43.     private int a;
44.     private int b;
45.
46.     @AfterViews
47.     protected void afterViews() {
48.         ...
49.     }
50.
51.     @Click(R.id.btn_Executer)
52.     void doExecuter() {
53.         ...
54.     }
55.
56.     // on vérifie la validité des données saisies
57.     private boolean isPageValid() {
58.         ...
59.     }
60.
61.     @Override
62.     protected void updateFragment() {
63.         // log
64.         if (isDebugEnabled) {
65.             Log.d("Vue1Fragment", String.format("updateFragment %s", getParentInfos()));

```

```

66.     }
67.     // ne sera exécuté qu'une fois au démarrage de l'application
68.     // on crée l'adaptateur du ListView - il faut pour cela que la variable [activity] ait été initialisée
69.     adapterReponses=new ArrayAdapter<>(activity, android.R.layout.simple_list_item_1, android.R.id.text1, reponses);
70.     listReponses.setAdapter(adapterReponses);
71.   }
72. }
```

- lignes 69-70 : on fixe l'adaptateur du composant de type [ListView] ;

Le composant [ListView] sert à afficher une liste d'éléments. Il le fait au moyen d'un adaptateur de type [ListAdapter] lui-même relié à la source de données qui doit alimenter le [ListView]. Pour définir l'adaptateur d'un [ListView], on dispose de la méthode [ListView.setAdapter] suivante :

```
public void setAdapter (ListAdapter adapter)
```

[ListAdapter] est une interface. La classe [ArrayAdapter] est une classe implémentant cette interface. Le constructeur utilisé ligne 69 ci-dessus est le suivant :

```
public ArrayAdapter (Context context, int resource, int textViewResourceId, List<T> objects)
```

- [context] est l'activité qui affiche le [ListView] ;
- [resource] est l'entier identifiant la vue utilisée pour afficher un élément du [ListView]. Cette vue peut avoir une complexité quelconque. C'est le développeur qui la construit en fonction de ses besoins ;
- [textViewResourceId] est l'entier identifiant un composant [TextView] dans la vue [resource]. La chaîne affichée le sera par ce composant ;
- [objects] : la liste d'objets affichés par le [ListView]. La méthode [toString] des objets est utilisée pour afficher l'objet dans le [TextView] identifié par [textViewResourceId] dans la vue identifiée par [resource].

Le travail du développeur est de créer la vue [resource] qui va afficher chaque élément du [ListView]. Pour le cas simple où on ne désire afficher qu'une simple chaîne de caractères comme ici, Android fournit la vue identifiée par [android.R.layout.simple_list_item_1]. Celle-ci contient un composant [TextView] identifié par [android.R.id.text1]. C'est la méthode utilisée ligne 69 pour créer l'adaptateur du [ListView]. Cet adaptateur n'a besoin d'être défini qu'une fois. Pour permettre sa réutilisation, on l'a défini comme variable d'instance de la classe (ligne 39). Regardons de nouveau, la ligne 69 :

```
adapterReponses=new ArrayAdapter<>(activity, android.R.layout.simple_list_item_1, android.R.id.text1, reponses);
```

Le 1er paramètre du constructeur [ArrayAdapter] est l'activité obtenue dans un fragment par [getActivity] et qui a été ici mémorisée dans la variable [activity] de la classe parent. Ce champ n'a pas toujours de valeur. Ainsi les logs montrent que lorsqu'on arrive dans la méthode [@AfterViews] il n'a pas encore été initialisé et donc on ne peut pas mettre les lignes 69-70 dans cette méthode. Dans la méthode [updateFragment] c'est possible car on sait que lorsque cette méthode est exécutée, on a forcément [activity!=null]. L'adaptateur est ici associé à la source de données [reponses] définie ligne 37 ;

La méthode [doExecuter] traite le clic sur le bouton [Exécuter]. Son code est le suivant :

```

1.  @Click(R.id.btn_Executer)
2.  void doExecuter() {
3.    // on cache les éventuels msg d'erreur précédents
4.    txtErrorAleas.setVisibility(View.INVISIBLE);
5.    txtErrorIntervalle.setVisibility(View.INVISIBLE);
6.    // on efface les réponses précédentes
7.    reponses.clear();
8.    adapterReponses.notifyDataSetChanged();
9.    // on teste la validité des saisies
10.   if (!isValidPage()) {
11.     return;
12.   }
13.   // on demande les nombres aléatoires à l'activité
14.   List<Object> data = mainActivity.getAleas(a, b, nbAleas);
15.   // on crée une liste de String à partir de ces données
16.   for (Object o : data) {
17.     if (o instanceof Exception) {
18.       reponses.add(((Exception) o).getMessage());
19.     } else {
20.       reponses.add(o.toString());
21.     }
22.   }
23.   // refresh listview
24.   adapterReponses.notifyDataSetChanged();
25. }
```

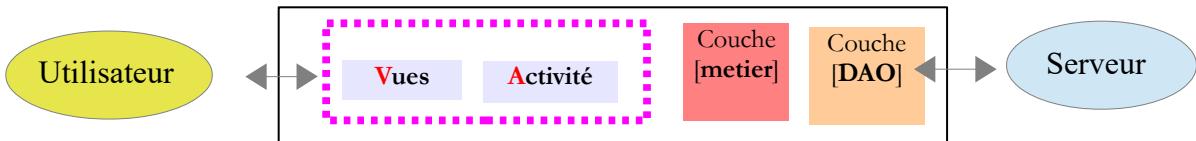
- lignes 7-8 : on veut vider le ListView. Pour cela, on vide la source de données [reponses] et on demande à l'adaptateur associé au ListView de se rafraîchir ;
- lignes 10-12 : avant d'exécuter l'action demandée, on vérifie que les valeurs saisies sont correctes ;
- ligne 14 : la liste des nombres aléatoires est demandée à l'activité. On obtient une liste d'objets où chaque objet est de type [Integer] ou [AleaException] ;
- lignes 16-22 : à partir de la liste d'objets obtenue, on met à jour la source de données [reponses] que le ListView affiche ;
- ligne 24 : on demande à l'adaptateur du ListView de se rafraîchir ;

1.15.8 Exécution

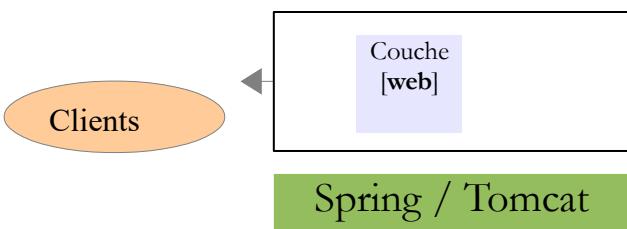
Exécutez le projet et vérifiez son bon fonctionnement.

1.16 Exemple-15 : architecture client / serveur

Nous abordons une architecture courante pour une application Android, celle où l'application Android communique avec des services web distants. On aura maintenant l'architecture suivante :



On a ajouté à l'application Android une couche [DAO] pour communiquer avec le serveur distant. Elle communiquera avec le serveur qui génère les nombres aléatoires affichés par la tablette Android. Ce serveur aura une architecture à deux couches suivante :



Les clients interrogent certaines URL de la couche [web / jSON] et reçoivent une réponse texte au format jSON (JavaScript Object Notation). Ici notre service web traitera une unique URL de type $[/a/b]$ qui renverra un nombre aléatoire dans l'intervalle $[a,b]$. Nous allons décrire l'application dans l'ordre suivant :

Le serveur

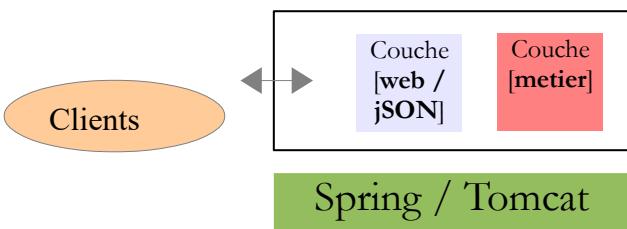
- sa couche [métier] ;
- son service [web / jSON] implémenté avec Spring MVC ;

Le client

- sa couche [DAO]. Il n'y aura pas de couche [métier] ;

1.16.1 Le serveur [web / jSON]

Nous voulons construire l'architecture suivante :



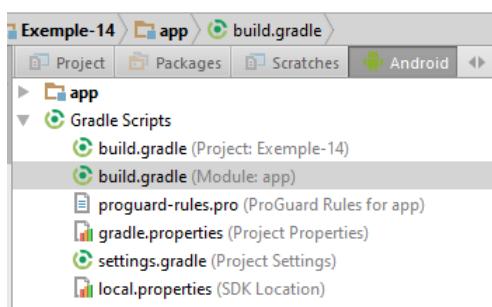
1.16.1.1 Crédit du projet

Nous allons construire le service web avec l'écosystème Spring [<http://spring.io/>]. Nous allons sur le site [<http://start.spring.io/>] (juin 2016) qui va nous permettre de générer un projet Gradle avec les dépendances nécessaires à notre projet, qui n'est pas un projet Android et pour la construction duquel Android Studio n'offre alors aucune aide :

The screenshot shows the 'Generate a Cradle Project with Spring Boot' interface. At the top, there's a dropdown for 'Cradle Project' set to '1.3.5'. Below it, the 'Project Metadata' section has 'Artifact coordinates' with 'Group' set to 'istia.st.examples.android' (marked with red circle 2) and 'Artifact' set to 'server-01' (marked with red circle 3). The 'Dependencies' section has a search bar with 'Web, Security, JPA, Act...' (marked with red circle 4), a 'Selected Dependencies' list with 'Web' (marked with red circle 5), and a green 'Generate Project alt + ⌘' button (marked with red circle 6).

- en [1] : choisissez un projet Gradle ;
- en [2-3] : les caractéristiques de la dépendance jar générée par le projet (voir ci-dessous) ;
- en [4] : choisir la dépendance web [5] pour que les binaires nécessaires pour créer notre service web soient disponibles ;
- en [6] : générez le projet. Le zip d'un projet Gradle squelette est alors généré et proposé au téléchargement ;

Que mettre en [2-3] ? Nous avons déjà utilisé des dépendances Gradle. Celui du projet précédent était par exemple le suivant :



```

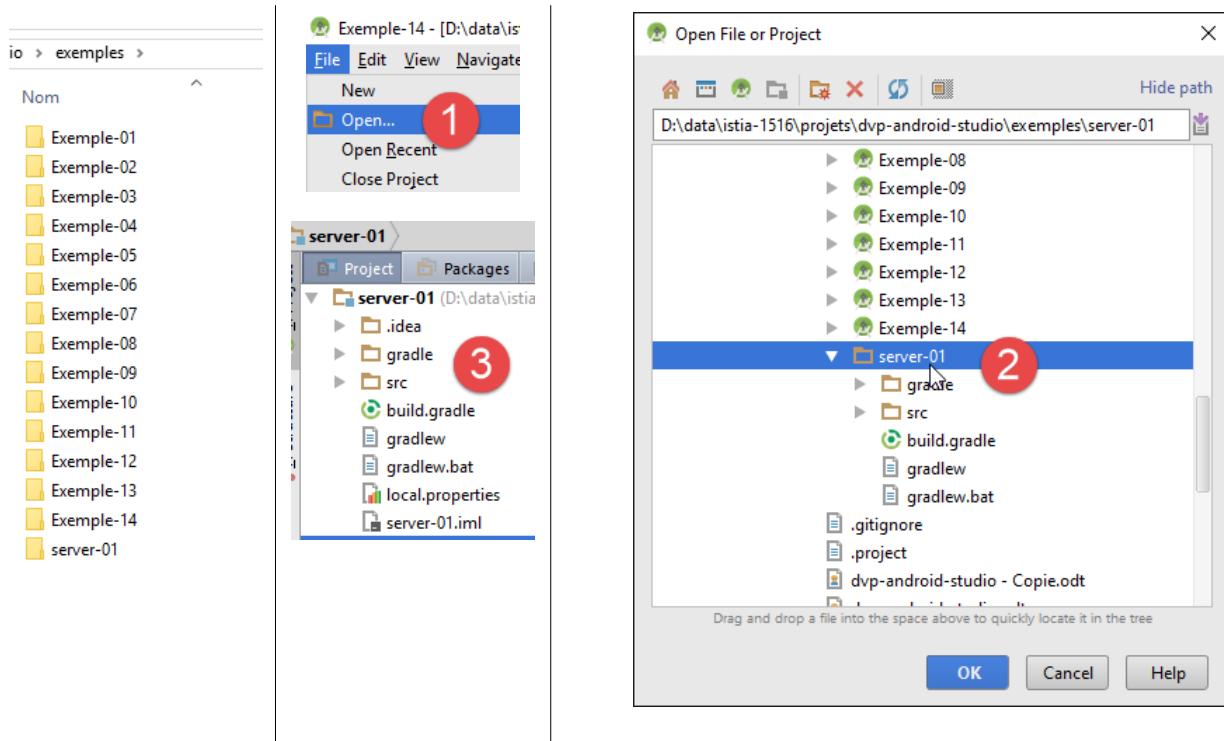
1. buildscript {
2.     repositories {
3.         mavenCentral()
4.     }
5.     dependencies {
6.         // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
7.         classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
8.     }
9. }
10.
11. apply plugin: 'com.android.application'
12. apply plugin: 'android-apt'
13.
14. android {
15.     compileSdkVersion 23
16.     ...
17. }
18. def AAVersion = '4.0.0'
19. dependencies {
20.     apt "org.androidannotations:androidannotations:$AAVersion"
21.     compile "org.androidannotations:androidannotations-api:$AAVersion"
22.     compile 'com.android.support:appcompat-v7:23.4.0'
23.     compile 'com.android.support:design:23.4.0'
24.     compile fileTree(dir: 'libs', include: ['*.jar'])
25.     testCompile 'junit:junit:4.12'
26. }

```

- ligne 22 : une dépendance se présente sous la forme [groupId:artifactId:version]. Ce qui est demandé sur le formulaire du site [<http://start.spring.io/>]:
 - en [2] est [groupId] ;

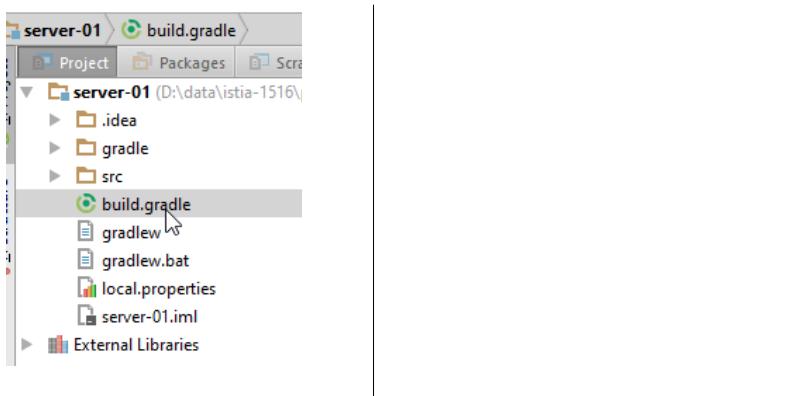
- en [3] est [artifactId] ;

Décompressez dans le dossier des autres projets, le fichier zip obtenu :



Avec android Studio, ouvrez le projet Gradle [server-01] [1-2]. Le projet ouvert est en [3] (perspective Project).

1.16.1.2 Configuration Gradle



Le fichier Gradle généré (juin 2016) est le suivant :

```

1. buildscript {
2.     ext {
3.         springBootVersion = '1.3.5.RELEASE'
4.     }
5.     repositories {
6.         mavenCentral()
7.     }
8.     dependencies {
9.         classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
10.    }
11. }
12.
13. apply plugin: 'java'
14. apply plugin: 'eclipse'
15. apply plugin: 'spring-boot'

```

```

16.
17. jar {
18.     baseName = 'server-01'
19.     version = '0.0.1-SNAPSHOT'
20. }
21.
22. sourceCompatibility = 1.8
23. targetCompatibility = 1.8
24.
25. repositories {
26.     mavenCentral()
27. }
28.
29. dependencies {
30.     compile('org.springframework.boot:spring-boot-starter-web')
31.     testCompile('org.springframework.boot:spring-boot-starter-test')
32. }
33.
34. eclipse {
35.     classpath {
36.         containers.remove('org.eclipse.jdt.launching.JRE_CONTAINER')
37.         containers 'org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-
1.8'
38.     }
39. }

```

- les lignes 14 et 34-38 sont pour l'IDE Eclipse. Nous les supprimons ;
- les lignes 1-11 et 15 servent à ajouter un plugin appelé [spring-boot] à notre projet Gradle. Spring Boot est un projet de l'écosystème Spring [<http://projects.spring.io/spring-boot/>]. Ce plugin définit les versions des dépendances les plus couramment utilisées avec Spring. Cela permet de ne pas préciser leurs versions (lignes 30 et 31). La version est alors celle définie par la version Spring Boot utilisée (ligne 3) ;
- lignes 22-23 : la version de Java à utiliser, ici la version 1.8 ;
- lignes 25-27 : les dépôts de binaires à utiliser pour télécharger les dépendances ;
- ligne 26 : désigne le dépôt Maven central. C'est actuellement le plus grand dépôt de binaires open source disponible ;
- lignes 29-32 : les dépendances nécessaires au projet :
- ligne 30 : cette dépendance amène avec elle tous les binaires nécessaires pour construire une service web Spring ;
- ligne 31 : cette dépendance amène avec elle tous les binaires nécessaires aux tests, notamment aux tests JUnit ;
- une dépendance [compile] indique qu'on a besoin de la dépendance pour faire la compilation du projet. Une dépendance [testCompile] indique qu'on a besoin de la dépendance uniquement pour l'exécution des tests. Elle n'est alors pas incluse dans le binaire du projet ;

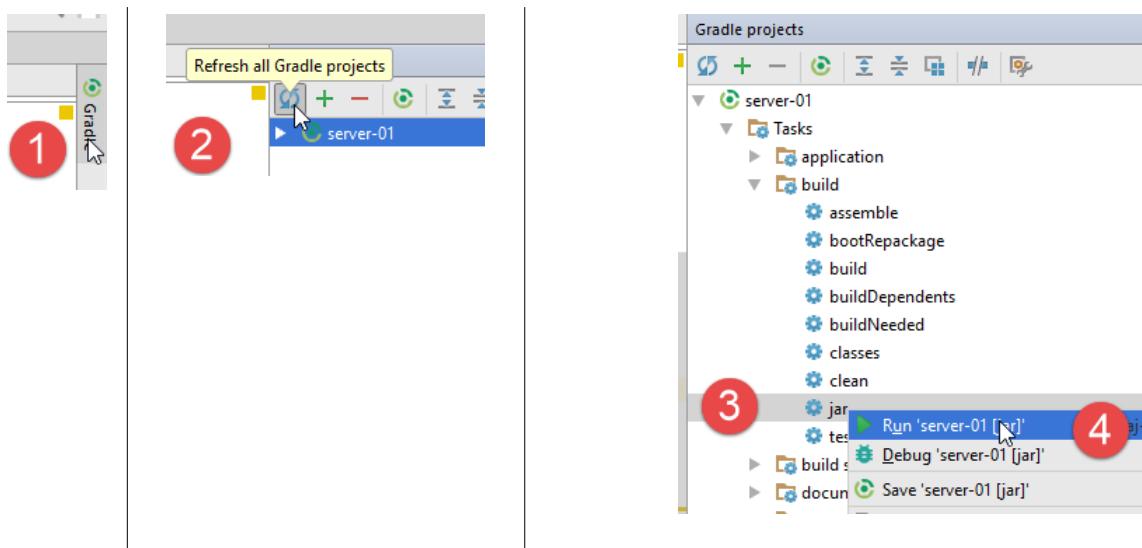
Nous faisons un premier nettoyage du fichier Gradle :

```

1. // spring boot
2. buildscript {
3.     ext {
4.         springBootVersion = '1.3.5.RELEASE'
5.     }
6.     repositories {
7.         mavenCentral()
8.     }
9.     dependencies {
10.         classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
11.     }
12. }
13.
14. // plugins
15. apply plugin: 'java'
16. apply plugin: 'spring-boot'
17.
18. // binaire du projet
19. jar {
20.     baseName = 'server-01'
21.     version = '0.0.1-SNAPSHOT'
22. }
23.
24. // versions Java
25. sourceCompatibility = 1.8
26. targetCompatibility = 1.8
27.
28. // dépôts Maven
29. repositories {
30.     mavenLocal()
31.     mavenCentral()
32. }
33.
34. // dépendances
35. dependencies {
36.     compile('org.springframework.boot:spring-boot-starter-web')
37.     testCompile('org.springframework.boot:spring-boot-starter-test')

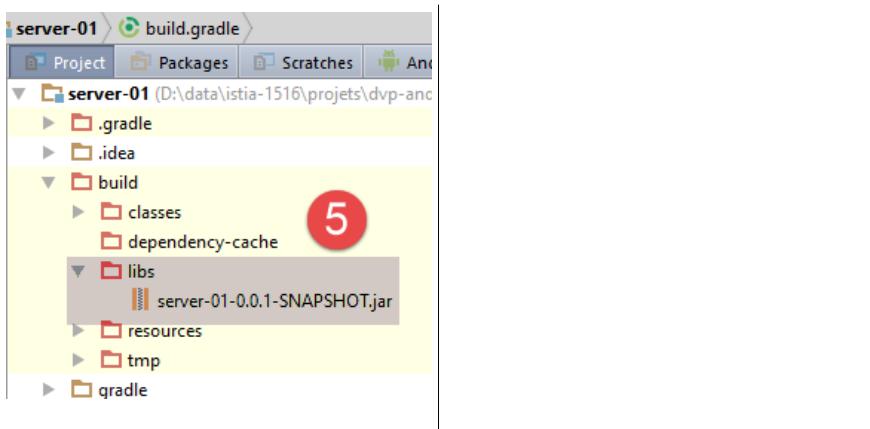
```

- ligne 30 : nous avons ajouté le dépôt Maven local du poste de développement. Celui-ci est créé lorsqu'on installe Maven (cf paragraphe 6.10, page 483). Si la dépendance demandée est déjà dans le dépôt Maven local, elle ne sera pas demandée au dépôt Maven central ;
- lignes 19-22 : une tâche Gradle permettant de générer le binaire du projet. Nous allons l'utiliser pour voir ce qui est fait ;



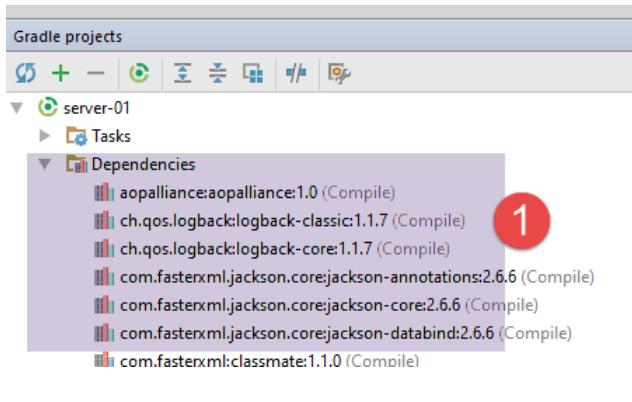
- en [1-4], exécutez la tâche [jar] définie dans le fichier [build.gradle] ([1] se trouve en haut à droite et sur le côté de l'IDE) ;

L'opération précédente crée l'archive jar du projet et place celui-ci dans le dossier [build / libs] [5] :



Le nom de l'archive provient directement des informations données à la tâche [jar] du fichier [build.gradle] (lignes 19-22).

L'ensemble des dépendances du projet peuvent être vues de la façon suivante :



On peut voir en [1] que l'unique dépendance du projet [compile('org.springframework.boot:spring-boot-starter-web')] a amené avec elle des dizaines de binaires. Spring Boot pour le web a inclus les dépendances dont une application web Spring MVC aura probablement besoin. Cela veut dire que certaines sont peut être inutiles. Spring Boot est idéal pour un tutoriel :

- il amène les dépendances dont nous aurons probablement besoin ;
- il amène un serveur Tomcat embarqué [1] ce qui nous évite le déploiement de l'application sur un serveur web externe ;

On trouvera de nombreux exemples utilisant Spring Boot sur le site de l'écosystème Spring [<http://spring.io/guides>].

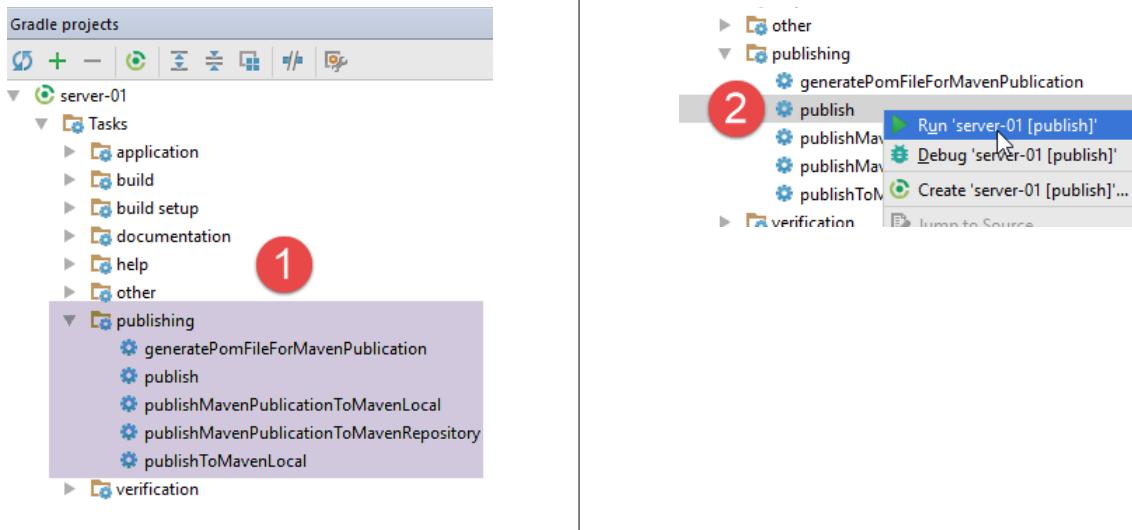
Nous complétons maintenant le fichier [build.gradle] de la façon suivante :

```

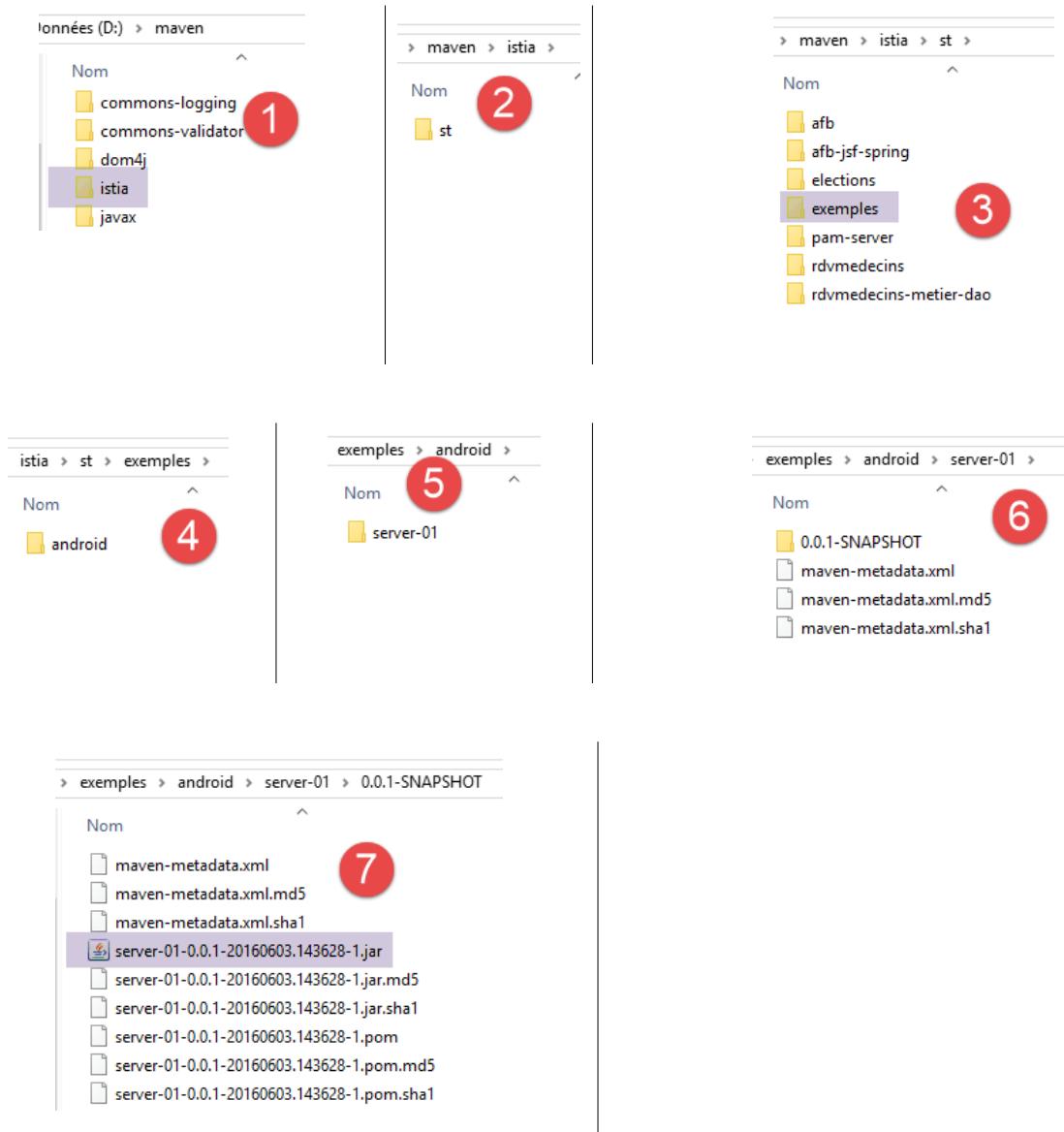
1. // spring boot
2. ...
3. // dépendances
4. dependencies {
5.     compile('org.springframework.boot:spring-boot-starter-web')
6.     testCompile('org.springframework.boot:spring-boot-starter-test')
7. }
8.
9. // plugin pour créer un binaire aux normes Maven dans le dépôt Maven local
10. apply plugin: 'maven-publish'
11. publishing {
12.     publications {
13.         maven(MavenPublication) {
14.             groupId 'istia.st.exemples.android'
15.             artifactId 'server-01'
16.             version '0.0.1-SNAPSHOT'
17.             from components.java
18.         }
19.     }
20.     repositories {
21.         maven {
22.             // change to point to your repo, e.g. http://my.org/repo
23.             url 'file:///D:\\maven'
24.         }
25.     }
26. }
```

- ligne 10 : on importe un plugin Gradle appelé [maven-publish] qui permet de publier le binaire du projet dans un dépôt Maven en respectant les normes Maven ;
- ligne 11 : une tâche Gradle appelée [publishing] ;
- lignes 14-15 : les caractéristiques du binaire Maven qui va être créé ;
- ligne 23 : le dépôt Maven sur lequel il sera publié, ici un dépôt Maven local ;

L'ajout du plugin [maven-publish] a créé de nouvelles tâches dans le projet Gradle :



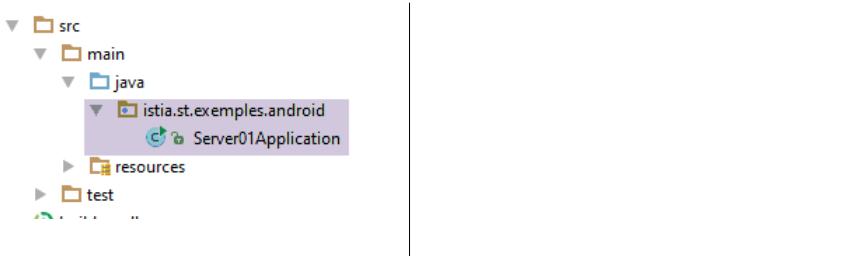
Si en [2], on exécute la tâche [publish], le binaire du projet est créé et installé dans le dossier indiqué en ligne 23 de la tâche [1-7] :



La tâche [jar] permet de générer le binaire du projet. Ce binaire est sans ses dépendances donc non exécutable. Il est possible de générer un binaire avec toutes ses dépendances et exécutable. Pour cela, nous ajoutons au fichier [build.gradle] le code suivant :

```
1. // créer un binaire avec toutes ses dépendances
2. version = '1.0'
3. task fatJar(type: Jar) {
4.     manifest {
5.         attributes 'Implementation-Title': 'Gradle Quickstart', 'Implementation-Version': version
6.         attributes 'Main-Class': 'istia.st.examples.android.Server01Application'
7.     }
8.     baseName = project.name + '-all'
9.     from { configurations.compile.collect { it.isDirectory() ? it : zipTree(it) } }
10.    with jar
11. }
```

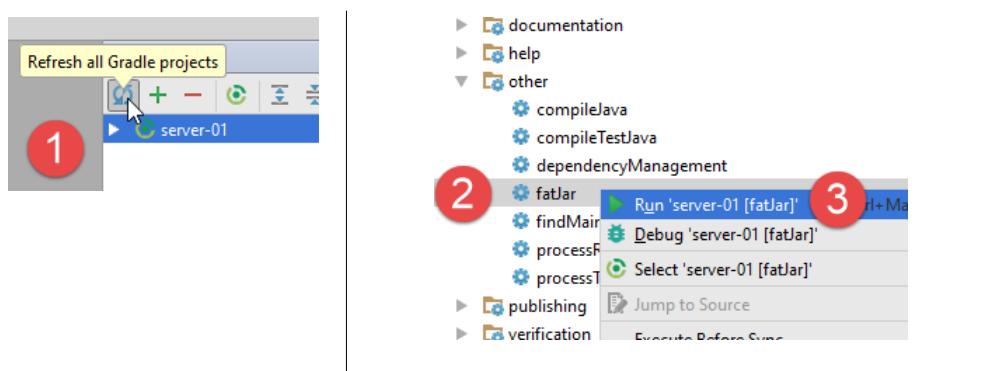
- ligne 6 : il faut mettre le nom complet de la classe exécutable du projet :



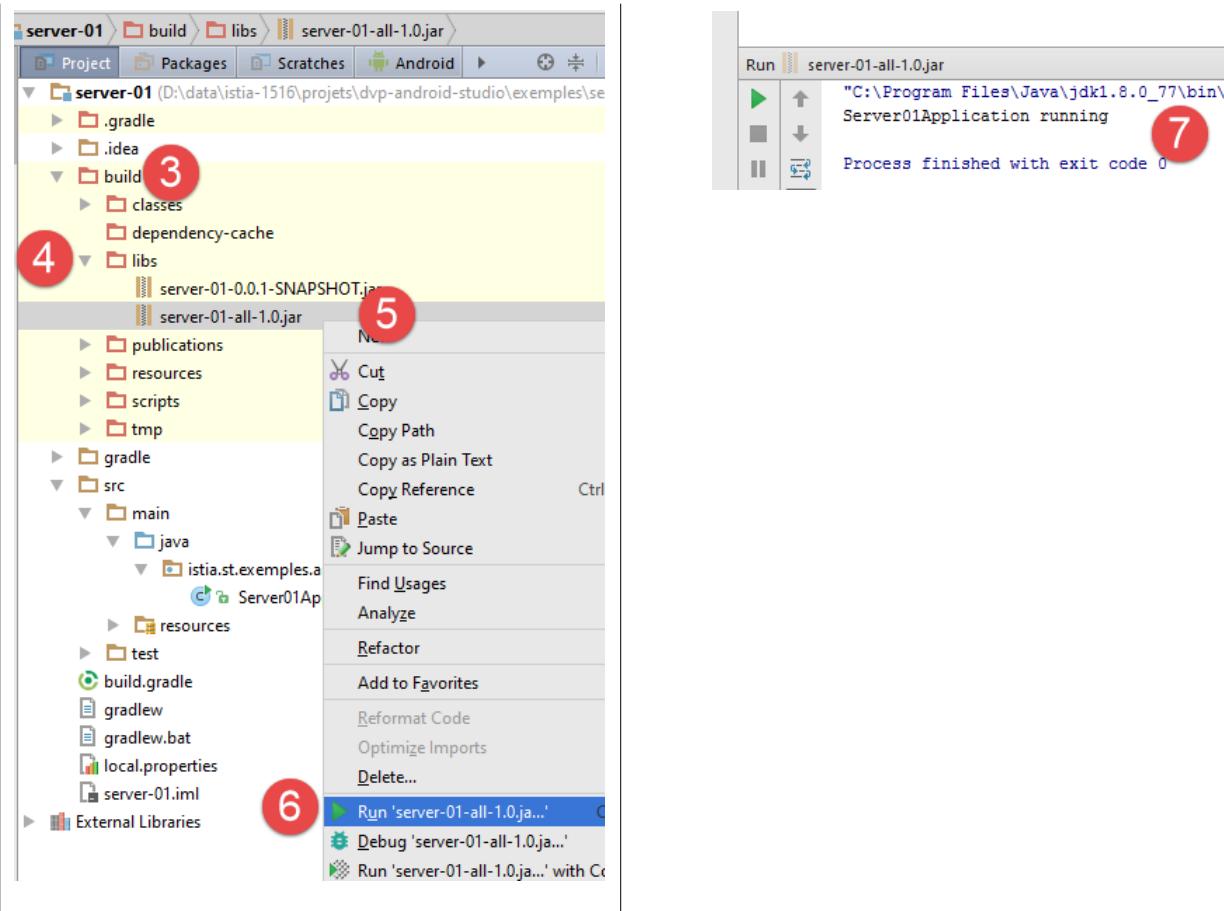
Le code de cette classe sera le suivant :

```
1. package istia.st.examples.android;
2.
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
5. import org.springframework.boot.autoconfigure.SpringBootApplication;
6.
7. @SpringBootApplication
8. public class Server01Application {
9.
10.    public static void main(String[] args) {
11.        System.out.println("Server01Application running");
12.        SpringApplication.run(Server01Application.class, args);
13.    }
14. }
```

Rafraîchissez le projet Gradle puis exécutez la tâche [fatJar] :

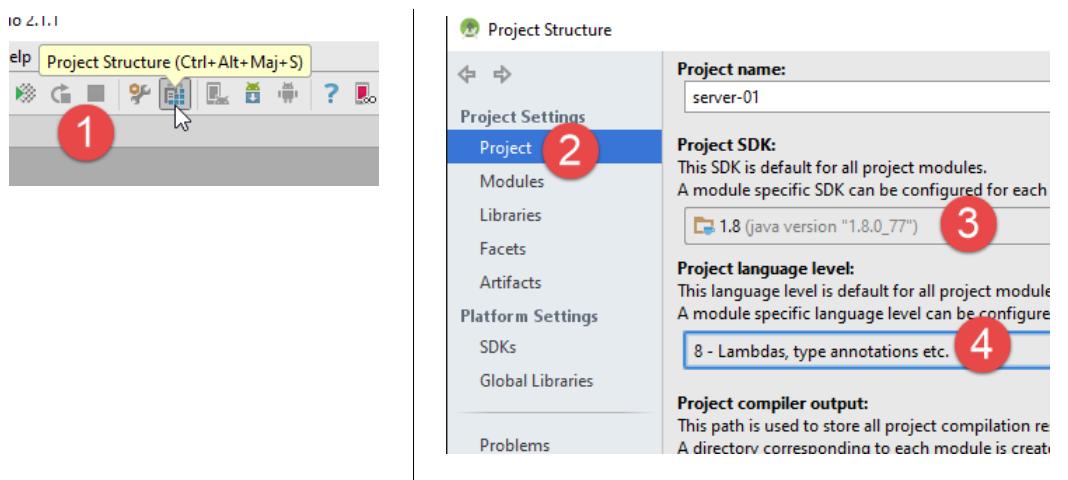


Le binaire est généré dans le dossier [build / libs] et peut être exécuté [1-7] :



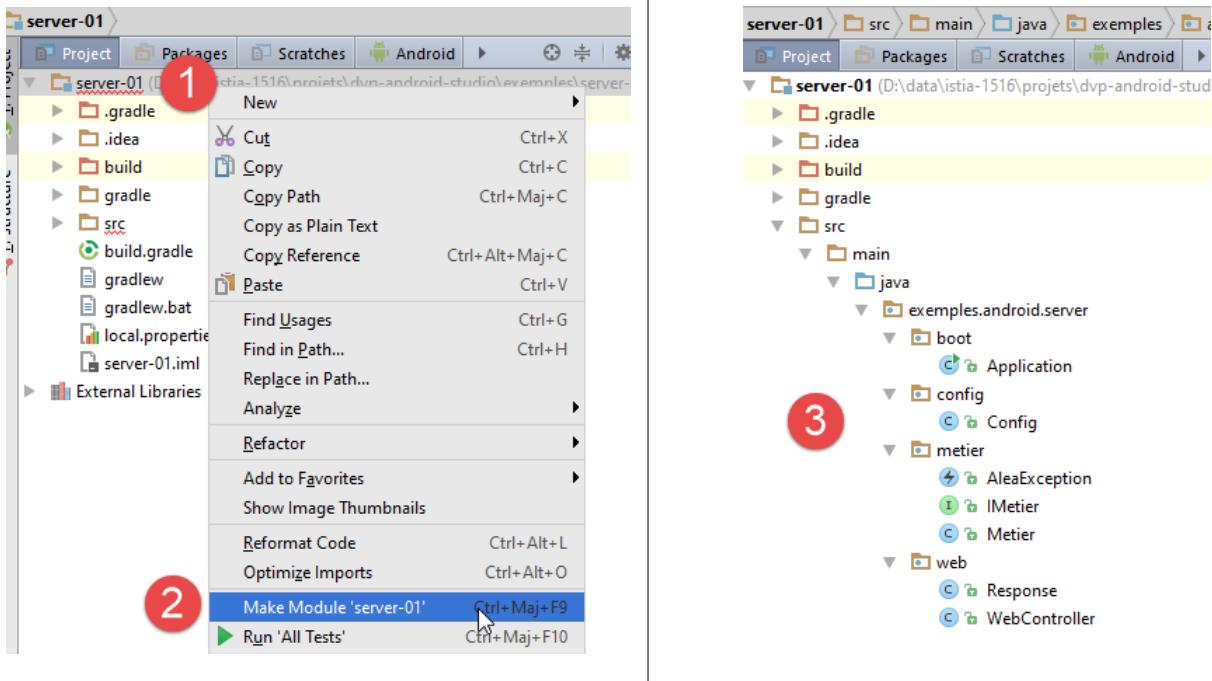
1.16.1.3 Configuration du projet

La configuration Gradle ne suffit pas. Il nous faut également configurer le projet. Comme ce n'est pas un projet Android généré par l'IDE, cette configuration que nous ne faisons pas doit être faite ici.



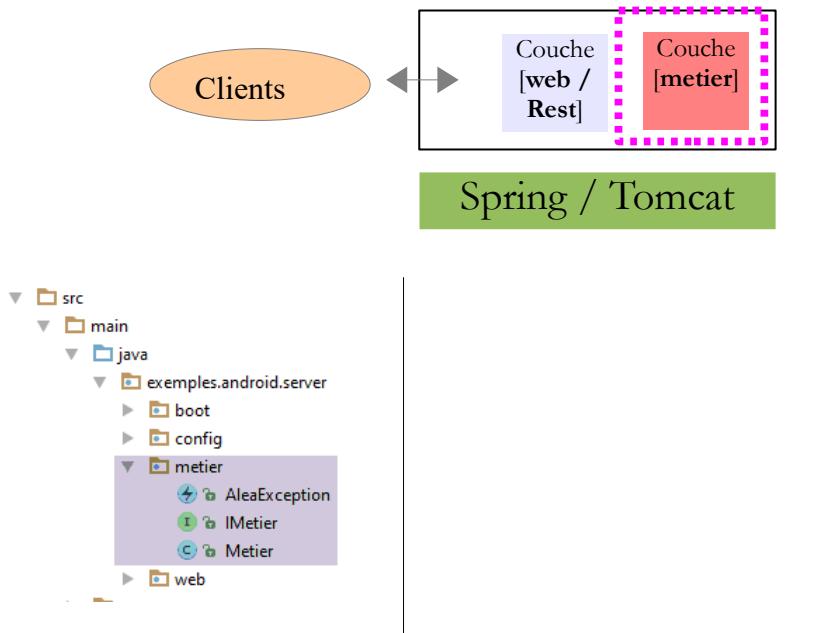
- en [3-4] : prenez un JDK 1.8 ;

Pour compiler le projet, le bouton disponible pour les projets Android n'est plus présent. On utilisera une option du menu [1-2] :



Dans la suite, le lecteur est invité à créer le projet qui suit. Nous commentons le code final du projet [3].

1.16.1.4 La couche [métier]



La couche [métier] reprend l'esprit de la couche [métier] de l'exemple précédent. Elle aura l'interface [IMetier] suivante :

```

1. package exemples.android.server.metier;
2.
3. public interface IMetier {
4.     // nombre aléatoire dans [a,b]
5.     int getAlea(int a, int b);
6. }

```

- ligne 5 : la méthode qui génère 1 nombre aléatoire dans [a,b]

Le code de la classe [Metier] implémentant cette interface est le suivant :

```
1. package exemples.android.server.metier;
2.
3. import org.springframework.stereotype.Service;
4.
5. import java.util.Date;
6. import java.util.Random;
7.
8. @Service
9. public class Metier implements IMetier {
10.
11.     @Override
12.     public int getAlea(int a, int b) {
13.         // qqs vérifications
14.         if (a < 0) {
15.             throw new AleaException("Le nombre a de l'intervalle [a,b] doit être supérieur à 0", 2);
16.         }
17.         if (b < 0) {
18.             throw new AleaException("Le nombre b de l'intervalle [a,b] doit être supérieur à 0", 3);
19.         }
20.         if (a >= b) {
21.             throw new AleaException("Dans l'intervalle [a,b], on doit avoir a < b", 4);
22.         }
23.         // génération résultat
24.         Random random=new Random();
25.         random.setSeed(new Date().getTime());
26.         return a + random.nextInt(b - a + 1);
27.     }
28. }
```

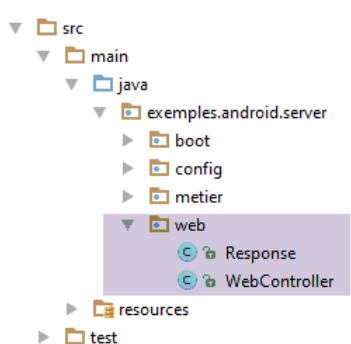
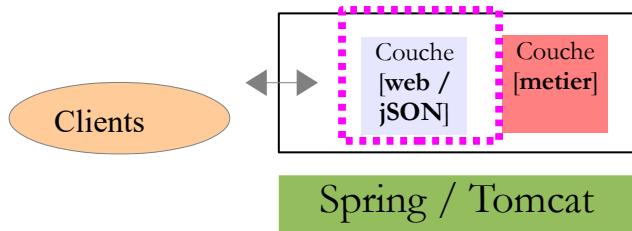
Nous ne commentons pas la classe : elle est analogue à celle rencontrée dans l'exemple précédent si ce n'est qu'elle ne lance pas d'exceptions aléatoirement. On notera simplement ligne 8 l'annotation Spring [`@Service`] qui va faire que Spring va instancier la classe en un unique exemplaire (singleton) et rendre sa référence disponible pour d'autres composants Spring. D'autres annotations Spring auraient pu être utilisées ici pour le même effet. Les composants Spring ont des noms par défaut qui peut être précisé comme attribut de l'annotation utilisée. Sans cet attribut, comme ici, le composant Spring porte le nom de la classe avec son premier caractère en minuscule. Ainsi ici, le composant Spring porte le nom [metier] ;

La classe [Metier] lance des exceptions de type [AleaException] :

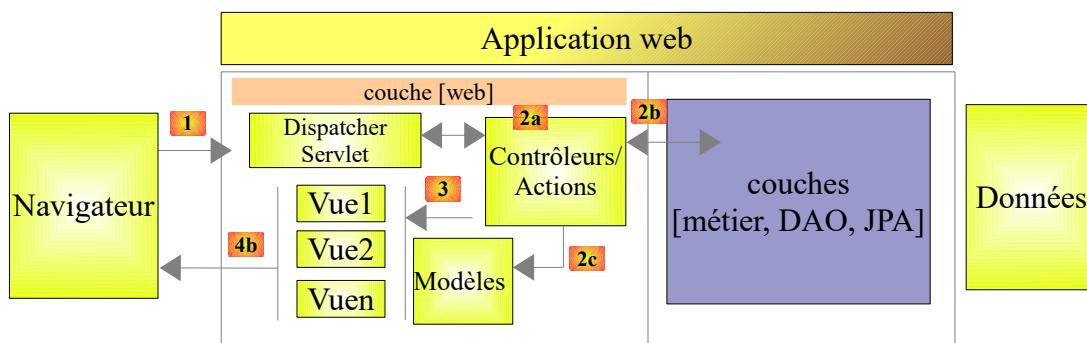
```
1. package exemples.android.server.metier;
2.
3. public class AleaException extends RuntimeException {
4.
5.     // code d'erreur
6.     private int code;
7.
8.     // constructeurs
9.     public AleaException() {
10.    }
11.
12.    public AleaException(String detailMessage, int code) {
13.        super(detailMessage);
14.        this.code = code;
15.    }
16.
17.    public AleaException(Throwable throwable, int code) {
18.        super(throwable);
19.        this.code = code;
20.    }
21.
22.    public AleaException(String detailMessage, Throwable throwable, int code) {
23.        super(detailMessage, throwable);
24.        this.code = code;
25.    }
26.
27.    // getters et setters
28.    ....
29. }
```

- ligne 3 : [AleaException] étend la classe [RuntimeException]. C'est donc une exception non contrôlée (pas d'obligation de la gérer avec un try / catch) ;
- ligne 6 : on ajoute à la classe [RuntimeException] un code d'erreur ;

1.16.1.5 Le service web / JSON



Le service web / JSON est implémenté par Spring MVC. Spring MVC implémente le modèle d'architecture dit MVC (Modèle – Vue – Contrôleur) de la façon suivante :

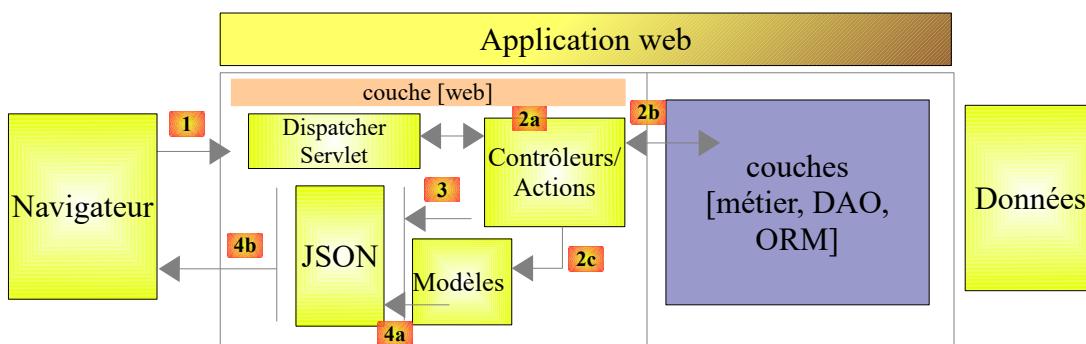


Le traitement d'une demande d'un client se déroule de la façon suivante :

1. **demande** - les URL demandées sont de la forme `http://machine:port/contexte/Action/param1/param2/...?p1=v1&p2=v2...`. La [Dispatcher Servlet] est la classe de Spring qui traite les URL entrantes. Elle "route" l'URL vers l'action qui doit la traiter. Ces actions sont des méthodes de classes particulières appelées [Contrôleurs]. Le C de MVC est ici la chaîne [Dispatcher Servlet, Contrôleur, Action]. Si aucune action n'a été configurée pour traiter l'URL entrante, la servlet [Dispatcher Servlet] répondra que l'URL demandée n'a pas été trouvée (erreur 404 NOT FOUND) ;
2. **traitement**
 - l'action choisie peut exploiter les paramètres *parami* que la servlet [Dispatcher Servlet] lui a transmis. Ceux-ci peuvent provenir de plusieurs sources :
 - du chemin `[/param1/param2/...]` de l'URL,
 - des paramètres `[p1=v1&p2=v2]` de l'URL,
 - de paramètres postés par le navigateur avec sa demande ;
 - dans le traitement de la demande de l'utilisateur, l'action peut avoir besoin de la couche [metier] [2b]. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
 - une page d'erreur si la demande n'a pu être traitée correctement
 - une page de confirmation sinon
 - l'action demande à une certaine vue de s'afficher [3]. Cette vue va afficher des données qu'on appelle le **modèle de la vue**. C'est le M de MVC. L'action va créer ce modèle M [2c] et demander à une vue V de s'afficher [3] ;

3. **réponse** - la vue **V** choisie utilise le modèle **M** construit par l'action pour initialiser les parties dynamiques de la réponse HTML qu'elle doit envoyer au client puis envoie cette réponse.

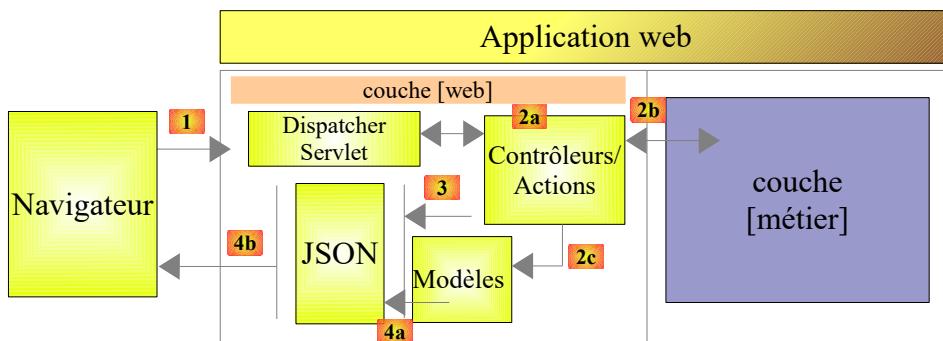
Pour un service web / JSON, l'architecture précédente est légèrement modifiée :



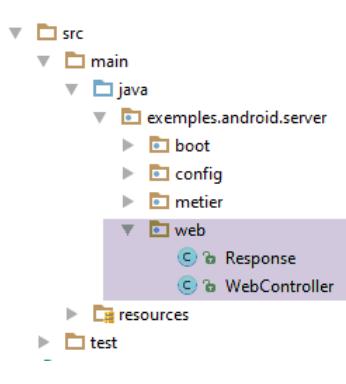
- en [4a], le modèle qui est une classe Java est transformé en chaîne JSON par une bibliothèque JSON ;
- en [4b], cette chaîne JSON est envoyée au navigateur ;

Un exemple de sérialisation d'un objet Java en chaîne JSON et de désérialisation d'une chaîne JSON en objet Java est présenté en annexes au paragraphe 6.14, page 502.

Revenons à la couche [web] de notre application :



Dans notre application, il n'y a qu'un contrôleur :



Le service web / JSON enverra à ses clients une réponse de type [Response] suivant :

```

1. package exemples.android.server.web;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.     // ----- propriétés

```

```

8.    // statut de l'opération
9.    private int status;
10.   // les éventuels messages d'erreur
11.   private List<String> messages;
12.   // le corps de la réponse
13.   private T body;
14.
15.   // constructeurs
16.   public Response() {
17.
18.   }
19.
20.  public Response(int status, List<String> messages, T body) {
21.    this.status = status;
22.    this.messages = messages;
23.    this.body = body;
24.  }
25.
26.  // getters et setters
27. ...
28. }
```

- ligne 13 : le champ [T body] est la réponse attendue par le client. On a décidé d'avoir ici une réponse générique de type T, plutôt que le type *Integer* du nombre aléatoire attendu. Nous voulons pouvoir réutiliser cette classe dans d'autres situations. Lors du traitement pour traiter la demande du client, le serveur peut rencontrer un problème qui est alors résumé dans les deux autres champs ;
- ligne 8 : un code d'état (0 si pas d'erreur) ;
- ligne 9 : si status!=0, une liste de messages d'erreur, usuellement ceux de la pile d'exceptions si exception il y a eu, *null* si pas d'erreur ;

Le contrôleur [WebController] est le suivant :

```

1. package exemples.android.server.web;
2.
3. import com.fasterxml.jackson.core.JsonProcessingException;
4. import com.fasterxml.jackson.databind.ObjectMapper;
5. import exemples.android.server.metier.AleaException;
6. import exemples.android.server.metier.IMetier;
7. import org.springframework.beans.factory.annotation.Autowired;
8. import org.springframework.stereotype.Controller;
9. import org.springframework.web.bind.annotation.PathVariable;
10. import org.springframework.web.bind.annotation.RequestMapping;
11. import org.springframework.web.bind.annotation.RequestMethod;
12. import org.springframework.web.bind.annotation.ResponseBody;
13.
14. import java.util.ArrayList;
15. import java.util.List;
16.
17. @Controller
18. public class WebController {
19.
20.   // couche métier
21.   @Autowired
22.   private IMetier metier;
23.   // mapper JSON
24.   @Autowired
25.   private ObjectMapper mapper;
26.
27.   // nombres aléatoires
28.   @RequestMapping(value = "/{a}/{b}", method = RequestMethod.GET, produces = "application/json; charset=UTF-8")
29.   @ResponseBody
30.   public String getAlea(@PathVariable("a") int a, @PathVariable("b") int b) throws JsonProcessingException {
31.
32.     // la réponse
33.     Response<Integer> response = new Response<>();
34.     // on utilise la couche métier
35.     try {
36.       response.setBody(metier.getAlea(a, b));
37.       response.setStatus(0);
38.     } catch (AleaException e) {
39.       response.setStatus(e.getCode());
40.       response.setMessages(getMessagesFromException(e));
41.     }
42.     // on rend la réponse
43.     return mapper.writeValueAsString(response);
44.   }
45.
46.   private List<String> getMessagesFromException(Throwable e) {
47.     // liste des messages
48.     List<String> messages = new ArrayList<String>();
49.     // on parcourt la pile des exceptions
50.     Throwable th = e;
51.     while (th != null) {
```

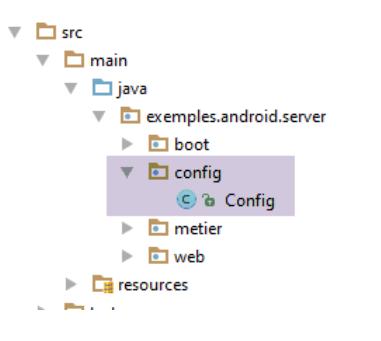
```

52.     messages.add(e.getMessage());
53.     th = th.getCause();
54.   }
55.   // on rend le résultat
56.   return messages;
57. }
58.
59. }

```

- ligne 17 : l'annotation `[@Controller]` indique que la classe est un contrôleur MVC dont les méthodes traitent des requêtes pour certaines URL de l'application web ;
- lignes 21-22 : l'annotation `[@Autowired]` demande à Spring d'injecter dans le champ, un composant de type `[IMetier]`. Ce sera la classe `[Metier]` précédente. C'est parce que nous avons mis à celle-ci l'annotation `[@Service]` qu'elle est gérée comme un composant Spring ;
- lignes 24-25 : nous faisons de même avec un mappeur JSON que nous définirons ultérieurement. Notre service web va envoyer sa réponse sous la forme d'une chaîne JSON. C'est ce mappeur qui opèrera la sérialisation de la réponse en JSON ;
- ligne 30 : la méthode qui génère le nombre aléatoire. Son nom n'a pas d'importance. Lorsqu'elle s'exécute, ses paramètres ont été initialisés par Spring MVC. Nous verrons comment. Par ailleurs, si elle s'exécute, c'est parce que le serveur web a reçu une requête HTTP GET pour l'URL de la ligne 28 ;
- ligne 28 : l'annotation `[@RequestMapping]` définit certaines propriétés de la méthode annotée :
 - `[value]` : l'URL acceptée par la méthode ;
 - `[method]` : la méthode HTTP acceptée par la méthode. Il y en a principalement deux, GET et POST. La méthode `[POST]` est utilisée lorsque le client veut joindre un document à sa requête HTTP ;
 - `[produces]` : fixe un des entêtes de la réponse HTTP qui sera faite au client. Ici, dans les entêtes HTTP envoyées avec la réponse du client, il y en aura un qui lui dira que la réponse lui est envoyée sous la forme d'une chaîne JSON. Cet entête n'est pas obligatoire. Elle est donnée à titre informatif au client si celui-ci attend des réponses qui peuvent avoir diverses formes ;
 - `[consumes]` : n'est pas présente ici. Elle permet d'indiquer les entêtes HTTP qui doivent accompagner la requête HTTP du client pour qu'elle soit acceptée ;
- ligne 29 : l'annotation `[@ResponseBody]` indique que le résultat produit par la méthode doit être envoyé au client. Sans cette annotation, la réponse de la méthode est considérée comme une clé permettant de sélectionner la page HTML à envoyer au client. Dans un service web / JSON, il n'y a pas de pages HTML ;
- ligne 28 : l'URL traitée est de la forme `/{a}/{b}` où `{x}` représente une variable. Les variables `{a}` et `{b}` sont affectées aux paramètres de la méthode ligne 30. Cela se fait via l'annotation `@PathVariable("x")`. On notera que `{a}` et `{b}` sont des composantes d'une URL et sont donc de type `String`. La conversion de `String` vers le type des paramètres peut échouer. Spring MVC lance alors une exception. **Résumons** : si avec un navigateur je demande l'URL `/100/200`, la méthode `getAlea` de la ligne 30 s'exécutera avec les paramètres entiers `a=100, b=200` ;
- ligne 36 : on demande à la couche [métier] un nombre aléatoire dans l'intervalle `[a,b]`. On se souvient que la méthode `[metier].getAlea` peut lancer une exception ;
- ligne 37 : pas d'erreur ;
- ligne 39 : code d'erreur ;
- ligne 40 : la liste des messages de la réponse est celle de la pile d'exceptions (lignes 46-57). Ici, nous savons que la pile ne contient qu'une exception mais nous avons voulu montrer une méthode plus générique ;
- ligne 43 : la réponse de type `[Response<Integer>]` est rendue sous la forme d'une chaîne JSON ;

1.16.1.6 Configuration du projet Spring



Il existe diverses façons de configurer Spring :

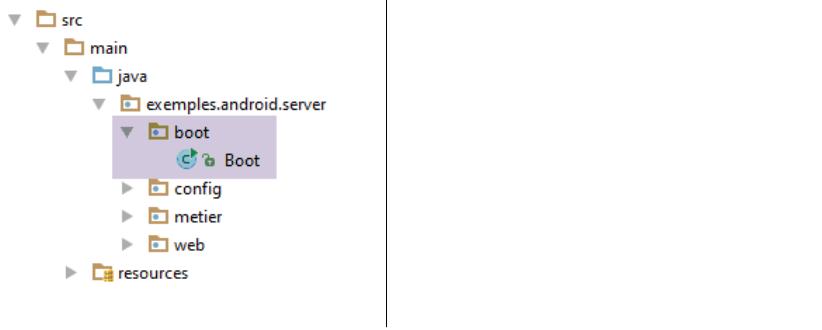
- avec des fichiers XML ;
- avec du code Java ;
- avec un mix des deux ;

Nous choisissons de configurer notre application web avec du code Java. C'est la classe [Config] suivante qui assure cette configuration :

```
1. package exemples.android.server.config;
2.
3. import com.fasterxml.jackson.databind.ObjectMapper;
4. import org.springframework.boot.context.embedded.EmbeddedServletContainerFactory;
5. import org.springframework.boot.context.embedded.ServletRegistrationBean;
6. import org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainerFactory;
7. import org.springframework.context.annotation.Bean;
8. import org.springframework.context.annotation.ComponentScan;
9. import org.springframework.web.servlet.DispatcherServlet;
10. import org.springframework.web.servlet.config.annotation.EnableWebMvc;
11.
12. @ComponentScan(basePackages = { "exemples.android.server.metier", "exemples.android.server.web" })
13. @EnableWebMvc
14. public class Config {
15.     // configuration web -----
16.     @Bean
17.     public DispatcherServlet dispatcherServlet() {
18.         DispatcherServlet servlet = new DispatcherServlet();
19.         return servlet;
20.     }
21.
22.     @Bean
23.     public ServletRegistrationBean servletRegistrationBean(DispatcherServlet dispatcherServlet) {
24.         return new ServletRegistrationBean(dispatcherServlet, "*");
25.     }
26.
27.     @Bean
28.     public EmbeddedServletContainerFactory embeddedServletContainerFactory() {
29.         return new TomcatEmbeddedServletContainerFactory("", 8080);
30.     }
31.
32.     // mappeur JSON
33.     @Bean
34.     public ObjectMapper jsonMapper() {
35.         return new ObjectMapper();
36.     }
37.
38. }
```

- ligne 12 : on dit à Spring dans quels packages il va trouver les deux composants qu'il doit gérer :
 - le composant [Metier] annoté [`@Service`] dans le package [`exemples.android.server.metier`] ;
 - le composant [WebController] annoté [`@Controller`] dans le package [`exemples.android.server.web`] ;
- ligne 13 : l'annotation [`@EnableWebMvc`] permet à Spring Boot de faire lui-même un certain nombre de configurations standard pour une application Spring MVC. Cela décharge d'autant le développeur ;
- lignes 16, 22, 27 et 33 : l'annotation [`@Bean`] définit elle-aussi des composants (beans) Spring au même titre que les deux annotations rencontrées (`@Service`, `@Controller`). Ici l'annotation [`@Bean`] annote une méthode et non une classe et c'est le résultat de la méthode qui est le composant Spring. En l'absence d'attribut de nommage au sein de l'annotation [`@Bean`], le composant Spring créé porte le nom de la méthode annotée ;
- lignes 16-20 : définissent le bean [`dispatcherServlet`]. C'est un nom prédéfini de Spring MVC qui définit le *front controller* de l'application MVC, un objet par qui passent toutes les requêtes des clients et qui les dispatche (d'où son nom) aux différents [`@Controller`] de l'application Spring MVC ;
- ligne 18 : le bean [`dispatcherServlet`] est une instance de la classe [`DispatcherServlet`] fournie par Spring MVC ;
- lignes 22-25 : le bean [`ServletRegistrationBean`] sert à définir quelles URL sont acceptées par l'application. Ligne 24, on accepte toutes les URL ;
- lignes 27-30 : le bean [`embeddedServletContainerFactory`] sert à définir le serveur embarqué dans les dépendances qui doit héberger l'application web. La ligne 29 indique que c'est un serveur Tomcat et que celui travaillera sur le port 8080. Par défaut, les binaires de ce serveur web sont amenés par la dépendance [`org.springframework.boot:spring-boot-starter-web`] du fichier Gradle ;

1.16.1.7 Exécution du service web / JSON



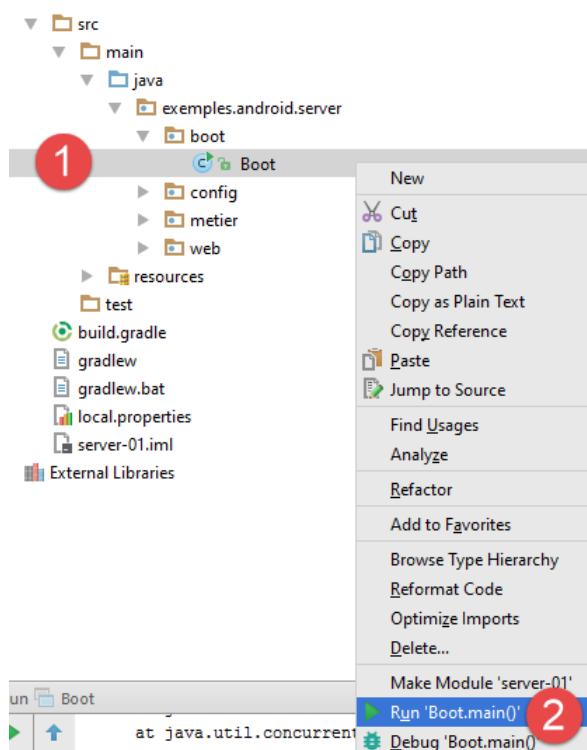
Le projet s'exécute à partir de la classe exécutable [Boot] suivante :

```

1. package exemples.android.server.boot;
2.
3. import exemples.android.server.config.Config;
4. import org.springframework.boot.SpringApplication;
5.
6. public class Boot {
7.     public static void main(String[] args) {
8.         // exécution application
9.         SpringApplication.run(Config.class, args);
10.    }
11.
12. }
```

- la classe [Boot] est une classe exécutable (lignes 7-10) ;
- ligne 9 : la méthode statique [SpringApplication.run] est une méthode de [spring Boot] (ligne 4) qui va lancer l'application. Son premier paramètre est la classe Java qui configure le projet. Ici la classe [Config] que nous venons de décrire. Le second paramètre est le tableau d'arguments passé à la méthode [main] (ligne 7) ;

On peut lancer l'application web de diverses façons dont la suivante :

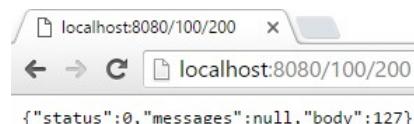


Dans la console, apparaissent alors un certain nombre de logs :

```
1. .
```

- lignes 12-14 : le serveur embarqué Tomcat est lancé ;
 - lignes 15-19 : la servlet [DispatcherServlet] de Spring MVC est chargée et configurée ;
 - ligne 20 : l'URL [/ {a} / {b}] du serveur web est détectée ;

Maintenant, prenons un navigateur et testons l'URL du service web / JSON :



The image contains three separate browser windows, each showing a JSON response to a request. The first window shows a status of 4 with messages about a < b. The second window shows a status of 2 with a message about a being less than or equal to 0. The third window shows a status of 3 with a message about b being less than or equal to 0.

```

localhost:8080/200/100
{
  "status": 4,
  "messages": ["Dans l'intervalle [a,b], on doit avoir a < b"],
  "body": null
}

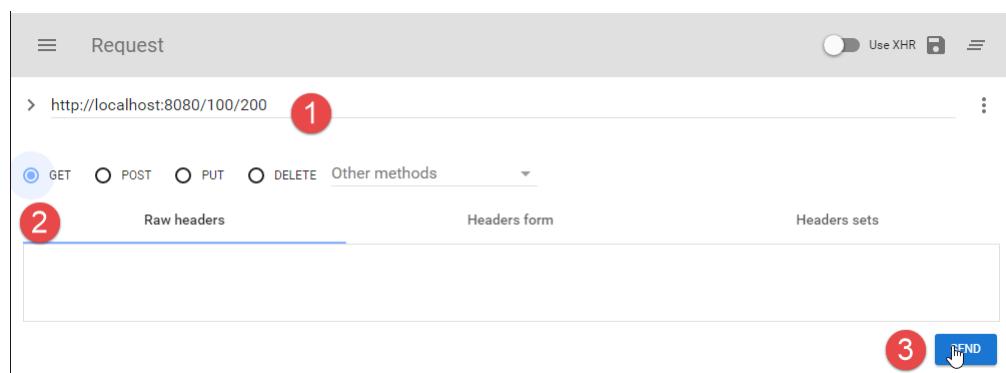
localhost:8080/-100/200
{
  "status": 2,
  "messages": ["Le nombre a de l'intervalle [a,b] doit être supérieur à 0"],
  "body": null
}

localhost:8080/100/-200
{
  "status": 3,
  "messages": ["Le nombre b de l'intervalle [a,b] doit être supérieur à 0"],
  "body": null
}

```

Nous obtenons à chaque fois, la représentation JSON d'un objet de type [Response<Integer>].

Au lieu de prendre un navigateur standard, prenons maintenant l'extension [Advanced Rest Client] du navigateur Chrome (voir annexes, paragraphe 6.13, page 501) :



- en [1], l'URL demandée ;
- en [2], au moyen d'un GET ;
- en [3], on envoie la requête ;

Status: 200: OK ? Loading time: 25 ms

Response headers (4)

Request

Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Content-Length: 39
Date: Sat, 04 Jun 2016 14:02:23 GMT

Raw

```
{
  "status": 0
  "messages": null
  "body": 107
}
```

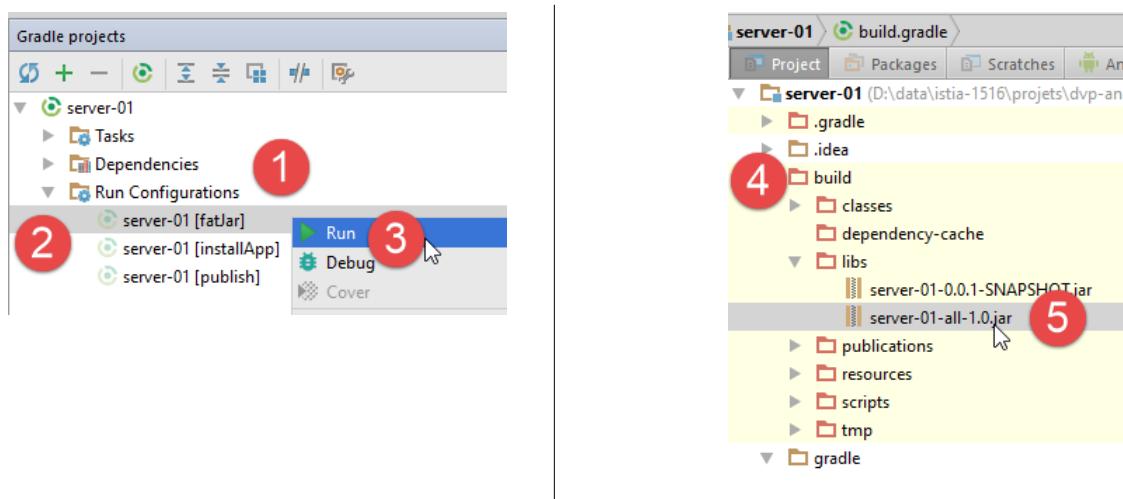
- en [4], les entêtes HTTP de la réponse du serveur. On remarquera que celui-ci indique que le document envoyé est une chaîne JSON ;
- en [5], la chaîne JSON reçue ;

1.16.1.8 Génération du jar exécutable du projet

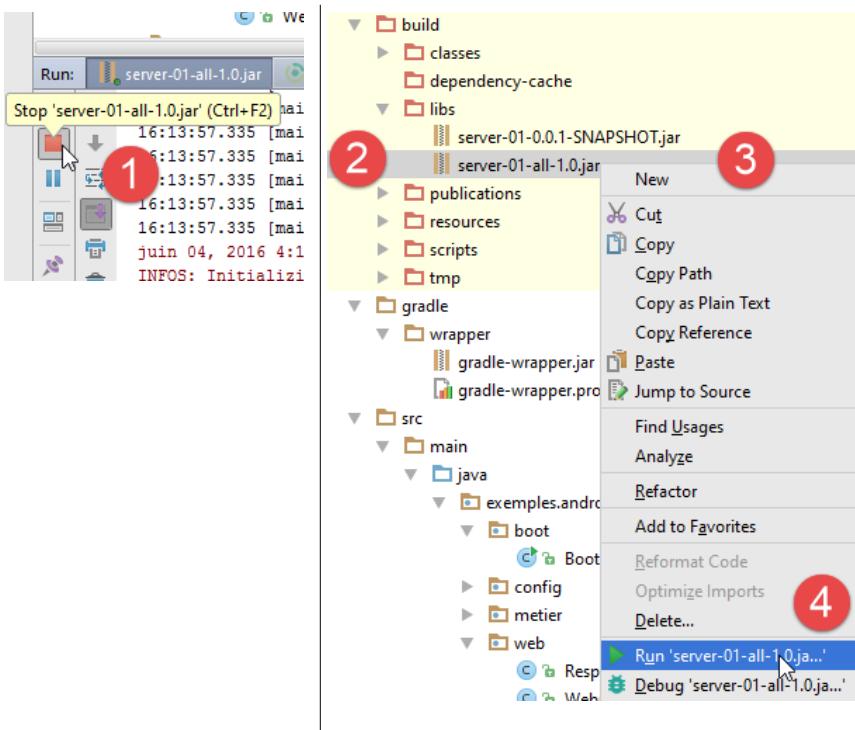
Au paragraphe 1.16.1.2, page 159, nous avons montré comment configurer le fichier Gradle pour générer un exécutable de l'application avec toutes ses dépendances. Adaptée à l'application présente, cet configuration devient la suivante :

```
1. // créer un binaire avec toutes ses dépendances
2. version = '1.0'
3. task fatJar(type: Jar) {
4.     manifest {
5.         attributes 'Implementation-Title': 'Gradle Quickstart', 'Implementation-Version': version
6.         attributes 'Main-Class': 'exemples.android.server.boot.Boot'
7.     }
8.     baseName = project.name + '-all'
9.     from { configurations.compile.collect { it.isDirectory() ? it : zipTree(it) } }
10.    with jar
11. }
```

Pour générer cet exécutable, on peut procéder comme suit [1-5] :



Pour l'exécuter, on arrêtera le service web s'il est lancé [1], puis on exécutera l'archive [2-4] :



Prenez un navigateur et demandez l'URL [localhost:8080/100/200]. Vous devez obtenir les mêmes résultats qu'auparavant.

1.16.1.9 Gestion des logs

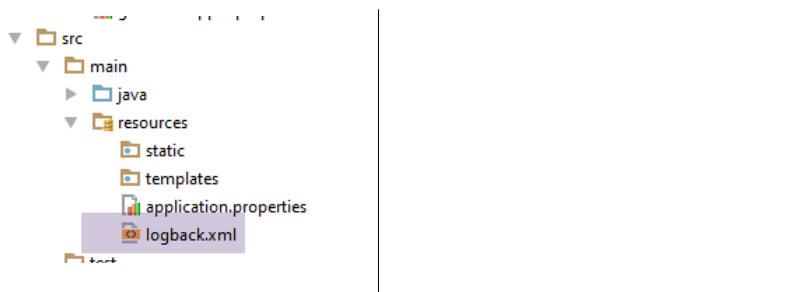
Lorsqu'on exécute l'archive exécutable, on constate qu'on n'a pas les mêmes logs que si on exécute le projet à partir de l'IDE. On a des logs en mode [DEBUG] :

```

1. ...
2. 09:32:03.741 [main] DEBUG org.springframework.core.env.PropertySourcesPropertyResolver - Searching for key
   'spring.liveBeansView.mbeanDomain' in [servletConfigInitParams]
3. 09:32:03.742 [main] DEBUG org.springframework.core.env.PropertySourcesPropertyResolver - Searching for key
   'spring.liveBeansView.mbeanDomain' in [servletContextInitParams]
4. 09:32:03.742 [main] DEBUG org.springframework.core.env.PropertySourcesPropertyResolver - Searching for key
   'spring.liveBeansView.mbeanDomain' in [systemProperties]
5. 09:32:03.742 [main] DEBUG org.springframework.core.env.PropertySourcesPropertyResolver - Searching for key
   'spring.liveBeansView.mbeanDomain' in [systemEnvironment]
6. 09:32:03.742 [main] DEBUG org.springframework.core.env.PropertySourcesPropertyResolver - Could not find key
   'spring.liveBeansView.mbeanDomain' in any property source. Returning [null]
7. juin 07, 2016 9:32:03 AM org.apache.coyote.AbstractProtocol init
8. INFO: Initializing ProtocolHandler ["http-nio-8080"]
9. juin 07, 2016 9:32:03 AM org.apache.coyote.AbstractProtocol start
10. INFO: Starting ProtocolHandler ["http-nio-8080"]
11. juin 07, 2016 9:32:03 AM org.apache.tomcat.util.net.NioSelectorPool getSharedSelector
12. INFO: Using a shared selector for servlet write/read
13. 09:32:03.810 [main] INFO org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainer - Tomcat started
   on port(s): 8080 (http)
14. 09:32:03.813 [main] INFO exemples.android.server.boot.Boot - Started Boot in 1.984 seconds (JVM running for 2.206)

```

On peut gérer le niveau des logs en ajoutant un fichier [logback.xml] dans le dossier [resources] du projet :



Ce fichier pourrait avoir le contenu suivant :

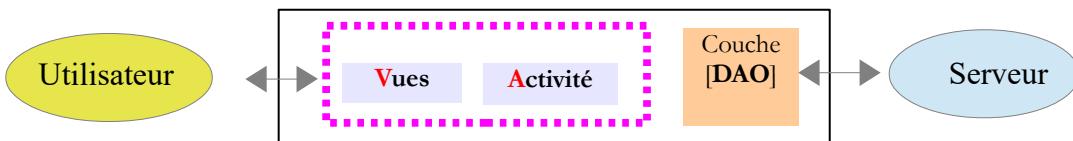
```
1. <configuration>
2.
3.   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
4.     <!-- encoders are by default assigned the type
5.         ch.qos.logback.classic.encoder.PatternLayoutEncoder -->
6.     <encoder>
7.       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
8.     </encoder>
9.   </appender>
10.
11.  <!-- contrôle niveau des logs -->
12.  <root level="info"> <!-- info, debug, warn -->
13.    <appender-ref ref="STDOUT" />
14.  </root>
15. </configuration>
```

Le niveau des logs est contrôlé ligne 12. Si maintenant, on régénère l'archive exécutable et qu'on exécute celle-ci, on n'a que des logs de niveau [info] :

```
1. ...
2. 09:36:52.433 [main] INFO o.h.validator.internal.util.Version - HV000001: Hibernate Validator 5.2.4.Final
3. 09:36:52.762 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerAdapter - Looking for @ControllerAdvice:
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@7085bdee: startup date [Tue Jun 07
09:36:51 CEST 2016]; root of context hierarchy
4. 09:36:52.811 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped "{{[/a]/
{b]},methods=[GET],produces=[application/json;charset=UTF-8]}" onto public java.lang.String
exemples.android.server.web.WebController.getAlea(int,int) throws com.fasterxml.jackson.core.JsonProcessingException
5. juin 07, 2016 9:36:52 AM org.apache.coyote.AbstractProtocol init
6. INFOS: Initializing ProtocolHandler ["http-nio-8080"]
7. juin 07, 2016 9:36:52 AM org.apache.coyote.AbstractProtocol start
8. INFOS: Starting ProtocolHandler ["http-nio-8080"]
9. juin 07, 2016 9:36:52 AM org.apache.tomcat.util.net.NioSelectorPool getSharedSelector
10. INFOS: Using a shared selector for servlet write/read
11. 09:36:52.923 [main] INFO o.s.b.c.e.TomcatEmbeddedServletContainer - Tomcat started on port(s): 8080 (http)
12. 09:36:52.926 [main] INFO exemples.android.server.boot.Boot - Started Boot in 1.865 seconds (JVM running for 2.203)
```

1.16.2 Le client Android du serveur web / JSON

Le client Android aura l'architecture suivante :



Le client aura deux composantes :

1. une couche [Présentation] (vue+activité) analogue à celle que nous avons étudiée dans l'exemple [Exemple-14] ;
2. la couche [DAO] qui s'adresse au service [web / JSON] que nous avons étudié précédemment.

1.16.2.1 Crédit du projet

Nous dupliquons le projet précédent [Exemple-14] dans [Exemple-15] en suivant la procédure du paragraphe 1.4, page 36. Nous obtenons le résultat suivant :

The screenshot shows the Android Studio interface. On the left, the Project tool window displays the project structure for 'Exemple-15' with files like .gradle, .idea, app, build, gradle, .gitignore, build.gradle, Exemple-15.iml, gradle.properties, gradlew, gradlew.bat, and local.properties. A red circle with the number '1' is placed over the .gradle file. On the right, the main editor area contains a Java code snippet for generating random numbers:

```

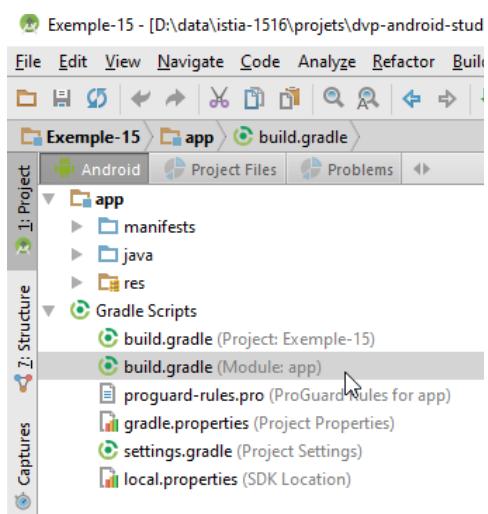
1 package com.example;
2
3 import java.util.Random;
4
5 public class Main {
6     public static void main(String[] args) {
7         int N = 10; // Valeur de N
8         int a = 1; // Intervalle [a,b] de génération
9         int b = 100; // Intervalle [a,b] de génération
10
11        Random rand = new Random();
12        for (int i = 0; i < N; i++) {
13            int nb = rand.nextInt(b - a + 1) + a;
14            System.out.println("Nombre " + (i + 1) + " : " + nb);
15        }
16    }
17}

```

The code includes annotations: 'Valeur de N:' above the variable declaration, 'Intervalle [a,b] de génération, a :' and 'b :' above the range variables, and 'EXÉCUTER' (EXECUTE) button below the code. A red circle with the number '2' is placed over the 'EXÉCUTER' button.

Dans la suite, le lecteur est invité à créer le projet qui suit.

1.16.2.2 Configuration Gradle



Le fichier [build.gradle] est le suivant :

```

1. buildscript {
2.     repositories {
3.         mavenCentral()
4.     }
5.     dependencies {
6.         // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
7.         classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
8.     }
9. }
10.
11. apply plugin: 'com.android.application'
12. apply plugin: 'android-apt'
13.
14. android {
15.     compileSdkVersion 23
16.     buildToolsVersion "23.0.3"
17.     defaultConfig {
18.         applicationId "exemples.android"
19.         minSdkVersion 15
20.         targetSdkVersion 23

```

```

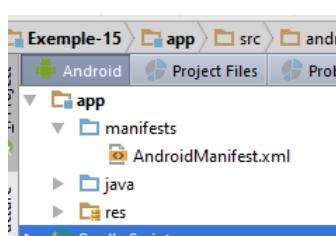
21.     versionCode 1
22.     versionName "1.0"
23. }
24.
25. buildTypes {
26.     release {
27.         minifyEnabled false
28.         proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
29.     }
30. }
31.
32. // options de packaging nécessaires pour être capable de produire L'APK
33. packagingOptions {
34.     exclude 'META-INF/ASL2.0'
35.     exclude 'META-INF/NOTICE'
36.     exclude 'META-INF/LICENSE'
37.     exclude 'META-INF/notice.txt'
38.     exclude 'META-INF/license.txt'
39. }
40. }
41.
42. def AAVersion = '4.0.0'
43. dependencies {
44.     apt "org.androidannotations:androidannotations:$AAVersion"
45.     compile "org.androidannotations:androidannotations-api:$AAVersion"
46.     apt "org.androidannotations:rest-spring:$AAVersion"
47.     compile "org.androidannotations:rest-spring-api:$AAVersion"
48.     compile 'com.android.support:appcompat-v7:23.4.0'
49.     compile 'com.android.support:design:23.4.0'
50.     compile 'org.springframework.android:spring-android-rest-template:2.0.0.M3'
51.     compile 'com.fasterxml.jackson.core:jackson-databind:2.7.4'
52.     compile fileTree(include: ['*.jar'], dir: 'libs')
53.     testCompile 'junit:junit:4.12'
54. }
55.
56. repositories {
57.     maven {
58.         url 'https://repo.spring.io/libs-milestone'
59.     }
60. }

```

Nous ne commentons que ce qui n'a pas déjà été rencontré :

- lignes 46-47 : insertion d'un plugin AA. Le plugin [rest-spring-api] permet de déléguer à la bibliothèque AA les échanges client / serveur ;
- ligne 50 : la bibliothèque [spring-android-rest-template] est la bibliothèque utilisée par AA pour assurer les échanges client / serveur. La version [2.0.0.M3] est une version dite 'milestone' qui ne se trouve pas dans les dépôts Maven habituels. Aussi doit-on préciser, lignes 56-59, le dépôt à utiliser (ligne 58) pour trouver la bibliothèque ;
- ligne 51 : une bibliothèque JSON ;
- lignes 33-39 : sans cette propriété, des erreurs apparaissent au moment de la génération du binaire APK du projet ;

1.16.2.3 Le manifeste de l'application Android



Le fichier [AndroidManifest.xml] doit évoluer. En effet, par défaut, les accès Internet sont désactivés. Il faut les activer par une directive spéciale :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.             package="exemples.android">
4.
5.     <uses-permission android:name="android.permission.INTERNET"/>
6.
7.     <application
8.             android:allowBackup="true"

```

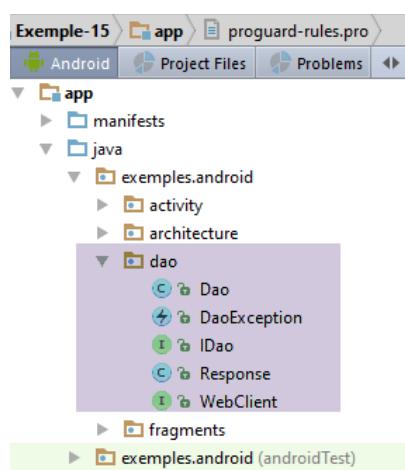
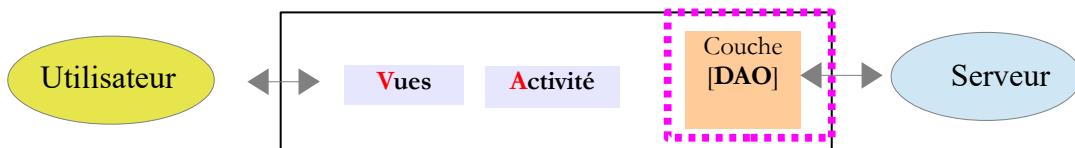
```

9.     android:icon="@mipmap/ic_launcher"
10.    android:label="@string/app_name"
11.    android:supportsRtl="true"
12.    android:theme="@style/AppTheme">
13.        <activity
14.            android:name=".activity.MainActivity_"
15.            android:label="@string/app_name"
16.            android:windowSoftInputMode="stateHidden"
17.            android:theme="@style/AppTheme.NoActionBar">
18.                <intent-filter>
19.                    <action android:name="android.intent.action.MAIN"/>
20.
21.                <category android:name="android.intent.category.LAUNCHER"/>
22.            </intent-filter>
23.        </activity>
24.    </application>
25.
26. </manifest>

```

- ligne 5 : les accès internet sont autorisés ;

1.16.2.4 La couche [DAO]



1.16.2.4.1 L'interface [IDao] de la couche [DAO]

L'interface de la couche [DAO] sera la suivante :

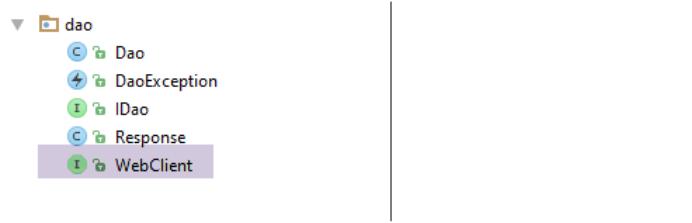
```

1. package exemples.android.dao;
2.
3. public interface IDao {
4.
5.     // nombre aléatoire
6.     int getAlea(int a, int b);
7.
8.     // URL du service web
9.     void setUrlServiceWebJson(String url);
10.
11.    // délai d'attente (ms) max de la réponse du serveur
12.    void setTimeout(int timeout);
13.
14.    // délai d'attente en millisecondes du client avant requête
15.    void setDelay(int delay);
16.
17. }

```

- ligne 6 : la méthode du service web / JSON pour obtenir un nombre aléatoire dans l'intervalle [a,b] de ce service web ;
- ligne 9 : l'URL du service web / JSON de génération de nombres aléatoires ;
- ligne 12 : on se fixe un délai d'attente maximal pour attendre la réponse du serveur ;
- ligne 15 : on veut fixer un délai d'attente avant exécution de la requête au serveur, ceci pour laisser le temps à l'utilisateur d'annuler sa demande ;

1.16.2.4.2 L'interface [WebClient]



L'interface [WebClient] s'occupe de dialoguer avec le service web. Son code est le suivant :

```

1. package exemples.android.dao;
2.
3. import org.androidannotations.rest.spring.annotations.Get;
4. import org.androidannotations.rest.spring.annotations.Path;
5. import org.androidannotations.rest.spring.annotations.Rest;
6. import org.androidannotations.rest.spring.api.RestClientRootUrl;
7. import org.androidannotations.rest.spring.api.RestClientSupport;
8. import org.springframework.http.converter.StringHttpMessageConverter;
9. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
10.
11. @Rest(converters = {MappingJackson2HttpMessageConverter.class})
12. public interface WebClient extends RestClientRootUrl, RestClientSupport {
13.
14.     // 1 nombre aléatoire dans l'intervalle [a,b]
15.     @Get("/{a}/{b}")
16.     Response<Integer> getAlea(@Path("a") int a, @Path("b") int b);
17. }
```

- ligne 12 : [WebClient] est une interface que la bibliothèque AA va implémenter elle-même grâce aux annotations qu'on va y mettre. Cette interface doit implémenter les appels aux URL exposées par le service web / JSON :

```

a)    // nombre aléatoire
b)    @RequestMapping(value = "/{a}/{b}", method = RequestMethod.GET, produces = "application/json; charset=UTF-8")
c)    @ResponseBody
d)    public String getAlea(@PathVariable("a") int a, @PathVariable("b") int b) throws JsonProcessingException {
```

- ligne 11 : l'annotation [@Rest] est une annotation AA. La valeur de l'attribut [converters] est un tableau de convertisseurs. Ici le convertisseur [MappingJackson2HttpMessageConverter.class] fait que lorsque le serveur envoie une chaîne JSON, celle-ci est automatiquement déserialisée. Ainsi on voit en ligne (d), que l'URL [/a/b] renvoie un type String qui est en fait une chaîne JSON (ligne b). Avec ces informations et celle du type attendu ligne 16, l'instance [WebClient] du client va déserialiser la chaîne qu'il va recevoir en un type [Response<Integer>] ;
- ligne 15 : une annotation AA indiquant que l'URL doit être appelée avec une méthode HTTP GET. Le paramètre de l'annotation [@Get] est la forme de l'URL attendue par le service web. Il suffit de reprendre le paramètre [value] de l'annotation [@RequestMapping] de la méthode appelée dans le contrôleur [WebController] du serveur. Les accolades {} entourent les paramètres de l'URL qui doivent être repris dans les paramètres de la méthode ligne 16. La syntaxe [@Path("a") int a] fait que le paramètre [a] de la méthode est affecté à la valeur {a} de l'URL. Lorsque le paramètre de l'URL et celui de la méthode portent le même nom comme ici, on peut écrire plus simplement [@Path int a] ;

Dans le cas d'une requête HTTP POST, la méthode d'appel aurait la signature suivante :

```

18.    @Post("/{a}/{b}")
19.    Response<Integer> getAlea(@Body T body, @Path("a") int a, @Path("b") int b);
```

C'est l'annotation [@Body] qui désigne la valeur postée. Celle-ci sera automatiquement sérialisée en JSON. Côté serveur, on aura la signature suivante :

```

1.    // nombres aléatoires
2.    @RequestMapping(value = "/{a}/{b}", method = RequestMethod.POST, consumes = "application/json; charset=UTF-8", produces =
   "application/json; charset=UTF-8")
3.    @ResponseBody
4.    public String getAlea(@PathVariable("a") int a, @PathVariable("b") int b, @RequestBody T body) {
```

- ligne 2 : on précise qu'on attend une requête HTTP POST et que le corps de cette requête (objet posté) doit être transmis sous forme d'une chaîne JSON (attribut *consumes*) ;
- ligne 4 : La valeur postée sera récupérée dans le paramètre [@RequestBody T body] de la méthode ;

Revenons au code de la classe [WebClient] :

```
1. @Rest(converters = {MappingJackson2HttpMessageConverter.class})
2. public interface WebClient extends RestClientRootUrl, RestClientSupport {
```

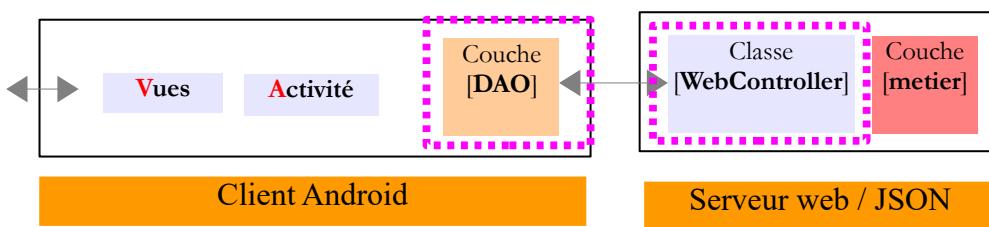
- il nous faut pouvoir indiquer l'URL du service web à contacter. Ceci est obtenu en étendant l'interface [RestClientRootUrl] fourni par AA. Cette interface expose une méthode [setRootUrl(urlServiceWeb)] qui permet de fixer l'URL du service web à contacter ;
- par ailleurs, nous voulons contrôler l'appel au service web car nous voulons limiter le temps d'attente de la réponse. Pour cela, nous étendons l'interface [RestClientSupport] qui expose la méthode [setRestTemplate] qui nous permettra de :
 - créer nous-mêmes l'objet [RestTemplate] qui sert à gérer les échanges client / serveur ;
 - paramétriser cet objet pour fixer le temps d'attente maximal de la réponse ;

1.16.2.4.3 La classe [Response]

La méthode [getAlea] de l'interface [IDao] rend une réponse du type [Response] suivant :

```
1. package exemples.android.dao;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.     // ----- propriétés
8.     // statut de l'opération
9.     private int status;
10.    // les éventuels messages d'erreur
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.    ...
28. }
```

C'est la classe [Response] déjà utilisée côté serveur (paragraphe 1.16.1.5, page 167). En fait, d'un point de vue programmation, tout se passe comme si la couche [DAO] du client communiquait directement avec le contrôleur [WebController] du service web :



La communication réseau entre client et serveur ainsi que la sérialisation / désérialisation des objets Java côté client sont transparents pour le programmeur.

1.16.2.4.4 Implémentation de la couche [DAO]



L'interface [IDao] est implémentée avec la classe [Dao] suivante :

```

1. package exemples.android.dao;
2.
3. import com.fasterxml.jackson.databind.ObjectMapper;
4. import exemples.android.architecture.Utils;
5. import org.androidannotations.annotations.EBean;
6. import org.androidannotations.rest.spring.annotations.RestService;
7. import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
8. import org.springframework.http.client.SimpleClientHttpRequestFactory;
9. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
10. import org.springframework.web.client.RestTemplate;
11.
12. import java.util.ArrayList;
13. import java.util.List;
14.
15. @EBean
16. public class Dao implements IDao {
17.
18.     // client du service REST
19.     @RestService
20.     protected WebClient webClient;
21.
22.     // mappeur JSON
23.     private ObjectMapper mapper = new ObjectMapper();
24.     // délay d'attente avant exécution requête
25.     private int delay;
26.
27. // interface IDao -----
28.     @Override
29.     public int getAlea(int a, int b) {
30.         ...
31.     }
32.
33.     @Override
34.     public void setUrlServiceWebJson(String urlServiceWebJson) {
35.         ...
36.     }
37.
38.     @Override
39.     public void setTimeout(int timeout) {
40.         ...
41.     }
42.
43.     @Override
44.     public void setDelay(int delay) {
45.         this.delay = delay;
46.     }
47.
48. }
```

- ligne 15 : nous annotons la classe [Dao] avec l'annotation [@EBean] pour en faire un bean AA qu'on va pouvoir injecter ailleurs ;
- lignes 19-20 : nous injectons l'implémentation qui sera faite de l'interface [WebClient] que nous avons décrite. C'est l'annotation [@RestService] qui assure cette injection ;
- les autres méthodes implémentent l'interface [IDao] (lignes 27-46) ;

Méthode [setTimeout]

La méthode [setTimeout] est la suivante :

```

1.     @Override
2.     public void setTimeout(int timeout) {
3.         // on fixe le timeout des requêtes du client REST
4.         SimpleClientHttpRequestFactory factory = new SimpleClientHttpRequestFactory();
5.         factory.setReadTimeout(timeout);
6.         factory.setConnectTimeout(timeout);
7.         // on construit le restTemplate
8.         RestTemplate restTemplate = new RestTemplate(factory);
```

```

9.     // on fixe le convertisseur JSON
10.    restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
11.    // on fixe le restTemplate du client web
12.    webClient.setRestTemplate(restTemplate);
13. }
```

- l'interface [WebClient] va être implémentée par une classe AA utilisant la dépendance Gradle [org.springframework.android:spring-android-rest-template]. [spring-android-rest-template] implémente le dialogue du client avec le serveur web / jSON au moyen d'une classe de type [RestTemplate] ;
- ligne 4 : la classe [SimpleClientHttpRequestFactory] est fournie par la dépendance [spring-android-rest-template]. Elle va nous permettre de fixer le délai d'attente maximum de la réponse du serveur (lignes 5-6) ;
- ligne 8 : nous construisons l'objet de type [RestTemplate] qui va être le support de la communication avec le service web. Nous lui passons comme paramètre l'objet [factory] qui vient d'être construit ;
- ligne 10 : le dialogue client / serveur peut prendre diverses formes. Les échanges se font par lignes de texte et nous devons indiquer à l'objet de type [RestTemplate] ce qu'il doit faire avec cette ligne de texte. Pour cela, nous lui fournissons des convertisseurs, des classes capables de traiter les lignes de texte. Le choix du convertisseur se fait en général via les entêtes HTTP qui accompagnent la ligne de texte. Ici, nous savons que nous recevons uniquement des lignes de texte au format jSON. Par ailleurs, nous avons vu page 175, que le serveur envoyait l'entête HTTP :

Content-Type: application/json; charset=UTF-8

Ligne 10, l'unique convertisseur du [RestTemplate] sera un convertisseur jSON implémenté avec la bibliothèque [Jackson]. Il y a une bizarrerie à propos de ces convertisseurs : AA nous impose de l'avoir également dans l'annotation du client web [WebClient] :

```

1. @Rest(converters = {MappingJacksonHttpMessageConverter.class})
2. public interface WebClient extends RestClientRootUrl, RestClientSupport {
```

Ligne 1, on est obligé de préciser un convertisseur alors même que nous le précisons par programmation.

- ligne 12 : l'objet [RestTemplate] ainsi construit est injecté dans l'implémentation de l'interface [WebClient] et c'est cet objet qui va opérer le dialogue client / serveur ;

Méthode [getAlea]

La méthode [getAlea] est la suivante :

```

1.  @Override
2.  public int getAlea(int a, int b) {
3.      // exécution service
4.      Response<Integer> info;
5.      DaoException ex;
6.      try {
7.          // attente
8.          waitSomeTime(delay);
9.          // exécution service
10.         info = webClient.getAlea(a, b);
11.         int status = info.getStatus();
12.         if (status == 0) {
13.             // on rend le résultat
14.             return info.getBody();
15.         } else {
16.             // on note l'exception
17.             ex = new DaoException(mapper.writeValueAsString(info.getMessages()), status);
18.         }
19.     } catch (JsonProcessingException | RuntimeException e) {
20.         // on note l'exception
21.         ex = new DaoException(e, 100);
22.     }
23.     // on lance l'exception
24.     throw ex;
25. }
26. ...
27. // méthodes privées -----
28. private void waitSomeTime(int delay) {
29.     try {
30.         Thread.sleep(delay);
31.     } catch (InterruptedException e) {
32.         e.printStackTrace();
33.     }
34. }
```

- ligne 8 : on attend [delay] millisecondes ;
- ligne 10 : on se contente d'appeler la méthode de même signature dans la classe implementant l'interface [WebClient] ;
- ligne 11 : on analyse la réponse obtenue du serveur en regardant son [status] ;

- lignes 12-14 : s'il n'y a pas eu d'erreur côté serveur (status=0), alors on rend le résultat de la méthode ;
- ligne 17 : s'il y a eu erreur côté serveur (status!=0), alors on prépare une exception sans la lancer. Le serveur a transmis une liste de messages d'erreur. Nous créons une exception avec comme unique message, la chaîne JSON de la liste des messages du serveur ;
- lignes 19-22 : autres cas d'exception ;
- ligne 24 : lorsqu'on arrive là, il y a forcément eu une exception. Alors on la lance ;

L'exception [DaoException] utilisée par ce code est le suivant :

```

1. package exemples.android.dao;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. public class DaoException extends RuntimeException {
7.
8.     // code d'erreur
9.     private int code;
10.
11.    // constructeurs
12.    public DaoException() {
13.    }
14.
15.    public DaoException(String detailMessage, int code) {
16.        super(detailMessage);
17.        this.code = code;
18.    }
19.
20.    public DaoException(Throwable throwable, int code) {
21.        super(throwable);
22.        this.code = code;
23.    }
24.
25.    // getters et setters
26.    ...
27. }
```

- ligne 6 : l'exception [DaoException] est une exception non contrôlée ;

Méthode [setUrlServiceWebJson]

La méthode [setUrlServiceWebJson] est la suivante :

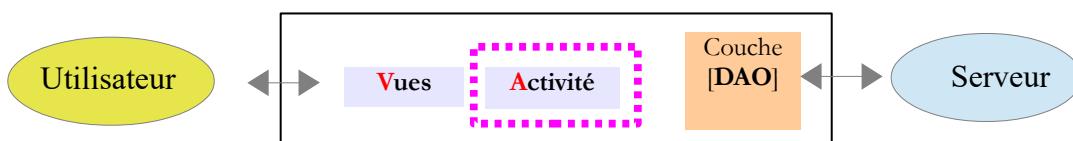
```

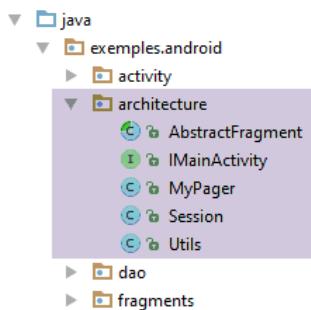
1.     @Override
2.     public void setUrlServiceWebJson(String urlServiceWebJson) {
3.         // on fixe l'URL du service REST
4.         webClient.setRootUrl(urlServiceWebJson);
5.     }
```

- ligne 4 : on fixe l'URL du service web via la méthode [setRootUrl] de l'interface [WebClient]. C'est parce que cette interface étend l'interface [RestClientRootUrl] que cette méthode existe ;

1.16.2.5 Le package [architecture]

Le package [architecture] regroupe les éléments qui structurent l'application :





1.16.2.5.1 L'interface [IMainActivity]

L'interface [IMainActivity] liste les méthodes que doit implémenter l'activité de l'application :

```

1. package exemples.android.architecture;
2.
3. import exemples.android.dao.IDao;
4.
5. public interface IMainActivity extends IDao {
6.
7.     // accès à la session
8.     Session getSession();
9.
10.    // changement de vue
11.    void navigateToView(int position);
12.
13.    // attente
14.    void beginWaiting();
15.
16.    void cancelWaiting();
17.
18.    // mode debug
19.    boolean IS_DEBUG_ENABLED = true;
20.    // délai d'attente de la réponse
21.    int TIMEOUT = 1000;
22.    // adjacence des fragments
23.    int OFF_SCREEN_PAGE_LIMIT = 1;
24.
25. }
```

- ligne 5 : l'interface [IMainActivity] étend l'interface [IDao] ;
- lignes 13-16 : aux méthodes déjà présentes dans les exemples précédents (lignes 7-11), nous avons ajouté deux méthodes pour gérer l'image d'attente de l'application (lignes 14, 16) ;
- ligne 21 : on fixe un délai maximal d'attente de la réponse du serveur à 1 seconde ;

1.16.2.5.2 La classe [Utils]

On a rassemblé dans la classe [Utils] des méthodes utilitaires statiques qui peuvent être appelées de différents endroits de l'architecture de l'application :

```

1. package exemples.android.architecture;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. public class Utils {
7.
8.     // liste de messages d'une exception - version 1
9.     static public List<String> getMessagesFromException(Throwable ex) {
10.         // on crée une liste avec les msg d'erreur de la pile d'exceptions
11.         List<String> messages = new ArrayList<>();
12.         Throwable th = ex;
13.         while (th != null) {
14.             messages.add(th.getMessage());
15.             th = th.getCause();
16.         }
17.         return messages;
18.     }
19.
20.     // liste de messages d'une exception
21.     static public String getMessagesForAlert(Throwable th) {
22.         // on construit le texte à afficher
23.         StringBuilder texte = new StringBuilder();
```

```

24.     List<String> messages = getMessagesFromException(th);
25.     int n = messages.size();
26.     for (String message : messages) {
27.         texte.append(String.format("%s : %s\n", n, message));
28.         n--;
29.     }
30.     // résultat
31.     return texte.toString();
32. }
33.
34. }
```

- lignes 9-18 : crée la liste des messages d'erreur contenu dans un *Throwable* ;
- lignes 21-32 : s'appuie sur la méthode précédente pour construire à partir de la liste de messages obtenue, le texte à afficher dans un message d'alerte Android ;
- lignes 27-28 : les messages sont numérotés. Le n° le plus petit (1) correspond à l'exception initiale et le n° le plus élevé à l'exception la plus récente dans la pile des exceptions ;

1.16.2.5.3 La classe abstraite [AbstractFragment]

La classe [AbstractFragment] a deux vocations :

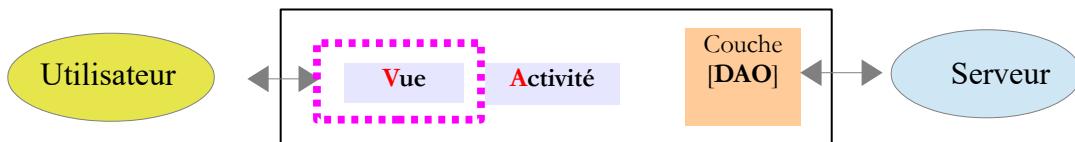
1. faire en sorte que la méthode [updateFragments] des classes filles soit toujours appelée lors de l'affichage du fragment et ce qu'une fois ;
2. factoriser l'état et les méthodes des classes filles qui peuvent l'être ;

C'est la vocation 2 qui nous fait mettre dans cette classe les opérations de gestion de l'image d'attente : tous les fragments d'une application Android asynchrone ont à gérer ce type de problématique :

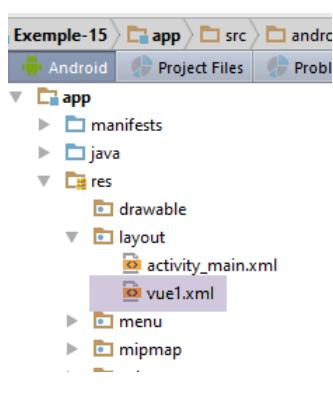
```

1.     // gestion de l'attente
2.     protected void beginWaiting() {
3.         // on met le sablier
4.         MainActivity.beginWaiting();
5.     }
6.
7.     protected void cancelWaiting() {
8.         // on enlève le sablier
9.         MainActivity.cancelWaiting();
10.    }
```

1.16.2.6 La vue



1.16.2.6.1 La vue [vue1.xml]



Par rapport à l'exemple précédent, la vue [vue1.xml] évolue de la façon suivante :

Exemple-15

Génération de N nombres aléatoires

Valeur de N : 10

Intervalle [a,b] de génération, a : 100 b : 200

URL du service web : 192.168.82.2:8080 1

Délai d'attente en ms avant chaque appel au service web : 0 2

EXÉCUTER

Liste des réponses (0) 3

Exemple-15 5

Génération de N nombres aléatoires

Valeur de N : 10

Intervalle [a,b] de génération, a : 100 b : 200

URL du service web : 192.168.82.2:8080

Délai d'attente en ms avant chaque appel au service web : 5000

ANNULER

4

Liste des réponses (0)

- en [1], l'utilisateur doit préciser l'URL du service web ainsi que le délai d'attente [2] avant chaque appel au service web ;
- en [3], les réponses sont comptées ;

- en [4], l'utilisateur peut annuler sa demande ;
- en [5], un indicateur d'attente s'affiche lorsque les nombres sont demandés. Il s'efface lorsqu'ils ont été tous reçus ou que l'opération a été annulée ;

Exemple-15

Génération de N nombres aléatoires

Valeur de N : Tapez un nombre entier >=1

Intervalle [a,b] de génération, a : b : Les bornes de l'intervalle doivent être entières et b>=a

URL du service web : Ip1.Ip2.Ip3.IP4:8080 L'Url du service doit être entrée sous la forme Ip1.Ip2.Ip3.IP4:Port/contexte

Délai d'attente en ms avant chaque appel au service web : Tapez un nombre entier >=0

EXÉCUTER

6

Liste des réponses (0)

- en [6], la validité des saisies est contrôlée ;

Le lecteur est invité à charger le fichier [vue1.xml] à partir des exemples. Pour la suite, nous donnons l'identifiant des nouveaux composants :

Exemple-15

Génération de N nombres aléatoires

Valeur de N : 1 Tapez un nombre entier >=1 2

Intervalle [a,b] de génération, a : 3 b : 4

Les bornes de l'intervalle doivent être entières et b>=a

URL du service web : Ip1.Ip2.Ip3.IP4:8080

5

7

6

L'Url du service doit être entrée sous la forme Ip1.Ip2.Ip3.IP4:Port/contexte

Délai d'attente en ms avant chaque appel au service web :

8

Tapez un nombre entier >=0

9

EXÉCUTER

10 11

Liste des réponses (0)

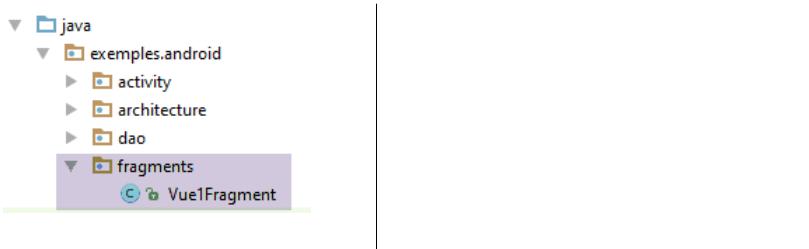
12 13

n°	Type	Id
1	EditText	edt_nbaleas
2	TextView	txt_errorNbAleas
3	EditText	edt_a
4	EditText	edt_b
5	TextView	txt_errorIntervalle
6	EditText	editTextUrlServiceWeb
7	TextView	textViewErreurUrl

8	EditText	editTextDelay
9	TextView	textViewErreurDelay
10	Button	btn_Executer
11	Button	btn_Annuler
12	TextView	txt_Reponses
13	ListView	lst_reponses

Les boutons [10-11] sont physiquement l'un sur l'autre. A un moment donné, on ne rendra visible que l'un des deux.

1.16.2.6.2 Le fragment [Vue1Fragment]



Le squelette du fragment [Vue1Fragment] est le suivant :

```

1. package exemples.android.fragments;
2.
3. import android.app.AlertDialog;
4. import android.support.annotation.*;
5. import android.support.v4.app.Fragment;
6. import android.view.View;
7. import android.widget.*;
8. import exemples.android.R;
9. import exemples.android.architecture.AbstractFragment;
10. import exemples.android.architecture.Utils;
11. import org.androidannotations.annotations.*;
12. import org.androidannotations.annotationsUiThread;
13. import org.androidannotations.api.BackgroundExecutor;
14.
15. import java.net.URI;
16. import java.net.URISyntaxException;
17. import java.util.ArrayList;
18. import java.util.List;
19.
20. @EFragment(R.layout.vue1)
21. public class Vue1Fragment extends AbstractFragment {
22.
23.     // les éléments de l'interface visuelle
24.     @ViewById(R.id.editTextUrlServiceWeb)
25.     EditText edtUrlServiceRest;
26.     @ViewById(R.id.textViewErreurUrl)
27.     TextView txtMsgErreurUrlServiceWeb;
28.     @ViewById(R.id.editTextDelay)
29.     EditText edtDelay;
30.     @ViewById(R.id.textViewErreurDelay)
31.     TextView textViewErreurDelay;
32.     @ViewById(R.id.lst_reponses)
33.     ListView listReponses;
34.     @ViewById(R.id.txt_Reponses)
35.     TextView infoReponses;
36.     @ViewById(R.id.edt_nbAleas)
37.     EditText edtNbAleas;
38.     @ViewById(R.id.edt_a)
39.     EditText edtA;
40.     @ViewById(R.id.edt_b)
41.     EditText edtB;
42.     @ViewById(R.id.txt_errorNbAleas)
43.     TextView txtErrorAleas;
44.     @ViewById(R.id.txt_errorIntervalle)
45.     TextView txtErrorIntervalle;
46.     @ViewById(R.id.btn_Executer)
47.     Button btnExecuter;
48.     @ViewById(R.id.btn_Annuler)
49.     Button btnAnnuler;
50. ...
51.     // données locales
52.     private List<String> reponses;
53.     private ArrayAdapter<String> adapterReponses;

```

```

54.
55. @AfterViews
56. void afterViews() {
57.     // mémoire
58.     afterViewsDone=true;
59.     // au départ pas de messages d'erreur
60.     txtErrorAleas.setVisibility(View.INVISIBLE);
61.     txtErrorIntervalle.setVisibility(View.INVISIBLE);
62.     txtMsgErreurUrlServiceWeb.setVisibility(View.INVISIBLE);
63.     textViewErreurDelay.setVisibility(View.INVISIBLE);
64.     // bouton [Annuler] caché
65.     btnAnnuler.setVisibility(View.INVISIBLE);
66.     btnExecuter.setVisibility(View.VISIBLE);
67.     // liste des réponses
68.     reponses = new ArrayList<>();
69. }
70. ...

```

- lignes 24-49 : les références sur les composants de la vue [vue1.xml] (ligne 20) ;
- lignes 55-69 : la méthode [@AfterViews] exécutée lorsque les références des lignes 24-49 ont été initialisées ;
- ligne 58 : à ne pas oublier - nécessaire pour le cycle de vie du fragment ;
- lignes 60-63 : les messages d'erreur sont cachés ;
- lignes 65-66 : on cache le bouton [Annuler] (ligne 65) et on affiche le bouton [Exécuter] (ligne 66). On rappelle qu'ils sont physiquement l'un sur l'autre ;
- ligne 68 : le champ de la ligne 52 va contenir la liste des chaînes de caractères à afficher par le *ListView* des réponses ;

Juste après la méthode [@AfterViews], la méthode [updateFragment] suivante va être exécutée :

```

1.     @Override
2.     protected void updateFragment() {
3.         // on crée l'adaptateur de la liste des réponses
4.         adapterReponses = new ArrayAdapter<>(activity, android.R.layout.simple_list_item_1, android.R.id.text1, reponses);
5.         listReponses.setAdapter(adapterReponses);
6.     }

```

- lignes 4-5 : on crée l'adaptateur du *ListView* des réponses. Il est mémorisé dans une variable d'instance pour être disponible aux autres méthodes de la classe ;

Le 'clic' sur le bouton [Exécuter] provoque l'exécution de la méthode suivante :

```

1.     // les saisies
2.     private int nbAleas;
3.     private int a;
4.     private int b;
5.     private String urlServiceWebJson;
6.     private int delay;
7.
8.     // données locales
9.     private int nbInfos;
10.    private List<String> reponses;
11.    private ArrayAdapter<String> adapterReponses;
12.    private boolean hasBeenCalled;
13.
14.    @Click(R.id.btn_Executer)
15.    protected void doExecute() {
16.        // on efface les réponses précédentes
17.        reponses.clear();
18.        adapterReponses.notifyDataSetChanged();
19.        hasBeenCalled = false;
20.        // on remet à 0 le compteur de réponses
21.        nbInfos = 0;
22.        infoReponses.setText(String.format("Liste des réponses (%s)", nbInfos));
23.        // on teste la validité des saisies
24.        if (!isValidPage()) {
25.            return;
26.        }
27.        // initialisation activité
28.        mainActivity.setUrlServiceWebJson(urlServiceWebJson);
29.        mainActivity.setDelay(delay);
30.        // on demande les nombres aléatoires
31.        for (int i = 0; i < nbAleas; i++) {
32.            getAlea(a, b);
33.        }
34.        // on commence l'attente
35.        beginWaiting();
36.    }
37.
38.    @Background(id = "alea")
39.    void getAlea(int a, int b) {

```

```

40.    // il faut faire le moins de choses possibles ici
41.    // en tout cas aucun affichage - ceux ci doivent se faire dans l'UiThread
42.    try {
43.        // on affiche le résultat dans l'UiThread
44.        showInfo(mainActivity.getAlea(a, b));
45.    } catch (RuntimeException e) {
46.        // on affiche l'exception dans l'UiThread
47.        showAlert(e);
48.    }
49. }

```

- lignes 17-18 : on efface la précédente liste de réponses du serveur. Pour cela, ligne 17, on vide la source de données [reponses] associée à l'adaptateur du *ListView* ;
 - ligne 19 : un booléen qui nous servira à savoir si l'utilisateur a annulé ou non sa demande ;
 - lignes 21-22 : on affiche un compteur nul pour le nombre de réponses ;
 - lignes 24-26 : on récupère les saisies des lignes [2-6] et on vérifie leur validité. Si l'une d'elles est invalide, la méthode est abandonnée (ligne 25) et pour l'utilisateur il y a retour à l'interface visuelle ;
 - lignes 28-29 : si les données saisies sont toutes valides, alors on transmet à l'activité l'URL du service web (ligne 28) ainsi que le délai d'attente avant chaque appel au service (ligne 29). Ces informations sont nécessaires à la couche [DAO] et on rappelle que c'est l'activité qui communique avec celle-ci ;
 - lignes 31-33 : les nombres aléatoires sont demandés un par un à la méthode [getAlea] de la ligne 39 ;
 - ligne 38 : la méthode [getAlea] est annotée avec l'annotation AA [@Background], ce qui fait qu'elle va être exécutée dans un autre thread (flux d'exécution, process) que celui dans lequel s'exécute l'interface visuelle. Il est en effet obligatoire d'exécuter tout appel internet dans un thread différent de celui de l'interface visuelle. Ainsi, à un moment donné, on pourra avoir plusieurs threads :
 - celui qui affiche l'interface visuelle UI (User Interface) et gère ses événements,
 - les [nbAleas] threads qui chacun demandent un nombre aléatoire au service web. Ces threads sont lancés de façon asynchrone : le thread de l'UI lance un thread [getAlea] (ligne 32) qui demande un nombre aléatoire au service web et n'attend pas sa fin. Celle-ci lui sera signalée par un événement. Ainsi, les [nbAleas] threads vont être lancés en parallèle. Il est possible de configurer l'application pour qu'elle ne lance qu'un thread à la fois. Il y a alors une file d'attente des threads à exécuter ;
- Ligne 38, le paramètre [id] donne un nom au thread généré. Ici les [nbAleas] threads portent tous le même nom [alea]. Cela va nous permettre de les annuler tous en même temps. Ce paramètre est facultatif si on ne gère pas l'annulation du thread ;
- ligne 44 : la méthode [getAlea] de l'activité est appelée. Elle le sera donc dans un thread à part de celui de l'UI. Celui-ci fera l'appel au service web et n'attendra pas la réponse. Il sera prévenu plus tard par un événement que la réponse est disponible. C'est à ce moment que ligne 44, la méthode [showInfo] sera appelée avec comme paramètre la réponse reçue ;
 - lignes 45-47 : l'exécution de la requête web peut produire une exception. On demande alors l'affichage des messages d'erreur de l'exception dans un message d'alerte ;
 - ligne 35 : on se met en attente des résultats :
 - un indicateur d'attente va être affiché ;
 - le bouton [Annuler] va remplacer le bouton [Exécuter]. Parce que les threads lancés sont asynchrones, le thread de l'UI ne les attend pas et la ligne 35 est exécutée avant leur fin. Une fois la méthode [beginWaiting] terminée, l'UI peut de nouveau répondre aux sollicitations de l'utilisateur tel que le clic sur le bouton [Annuler]. Si les threads lancés avaient été synchrones, on arriverait à la ligne 35 qu'une fois tous les thread terminés. L'annulation de ceux-ci n'aurait alors plus de sens ;

La méthode [showInfo] est la suivante :

```

1.  @UiThread
2.  protected void showInfo(int alea) {
3.      if (!hasBeenCalled) {
4.          // une info de plus
5.          nbInfos++;
6.          infoReponses.setText(String.format("Liste des réponses (%s)", nbInfos));
7.          // a-t-on terminé ?
8.          if (nbInfos == nbAleas) {
9.              // on termine l'attente
10.             cancelWaiting();
11.         }
12.         // on ajoute l'information à la liste des réponses
13.         reponses.add(0, String.valueOf(alea));
14.         // on affiche les réponses
15.         adapterReponses.notifyDataSetChanged();
16.     }
17. }

```

- la méthode [showInfo] est appelée à l'intérieur du thread [getAlea] annotée par [@Background]. Cette méthode va mettre à jour l'interface visuelle UI. Elle ne peut le faire qu'en étant exécutée à l'intérieur du thread de l'UI. C'est la signification de l'annotation [@UiThread] de la ligne 1 ;

- ligne 2 : la méthode reçoit un nombre aléatoire ;
- ligne 3 : le corps de la méthode n'est exécuté que si l'utilisateur n'a pas annulé sa demande ;
- lignes 5-6 : on incrémente le compteur de réponses et on l'affiche ;
- lignes 8-11 : si on a reçu toutes les réponses attendues, alors on termine l'attente (fin du signal d'attente, le bouton [Exécuter] remplace le bouton [Annuler]) ;
- lignes 12-15 : on ajoute le nombre aléatoire reçu à la liste des réponses affichée par le composant [ListView listReponses] et on rafraîchit celui-ci ;

La méthode [showAlert] est la suivante :

```

1.  @UiThread
2.  protected void showAlert(Throwable th) {
3.      if (!hasBeenCanceled) {
4.          // on annule tout
5.          doAnnuler();
6.          // on l'affiche
7.          new AlertDialog.Builder(activity).setTitle("Des erreurs se sont
produites").setMessage(Utils.getMessagesForAlert(th)).setNeutralButton("Fermer", null).show();
8.      }
9.  }
```

On retrouve une logique analogue à celle de la méthode [showInfo] :

- ligne 1 : l'annotation [**@UiThread**] est obligatoire ;
- ligne 2 : la méthode reçoit l'exception qui s'est produite ;
- ligne 3 : la méthode est exécutée que si l'utilisateur n'a pas annulé sa demande ;
- ligne 5 : on annule la demande de l'utilisateur comme s'il avait cliqué lui-même sur le bouton [Annuler] ;
- ligne 7 : on affiche l'alerte à l'aide de la classe Android [AlertDialog] :
 - [activity] : est l'activité de type [Activity] mémorisée dans la classe parent [AbstractFragment] ;
 - [setTitle] : fixe le titre de la fenêtre d'alerte [1] ;
 - [setMessage] : fixe le message affiché par la fenêtre d'alerte [2] ;
 - [setNeutral] : fixe le bouton qui va fermer la fenêtre d'alerte [3] ;
 - [show] : demande l'affichage de la fenêtre d'alerte ;

Des erreurs se sont produites 1

3 : org.springframework.web.client.ResourceAccessException: I/O error on GET request for "http://192.168.82.2:8080/100/200": failed to connect to /192.168.82.2 (port 8080) after 1000ms; nested exception is java.net.SocketTimeoutException: failed to connect to /192.168.82.2 (port 8080) after 1000ms
2 : I/O error on GET request for "http://192.168.82.2:8080/100/200": failed to connect to /192.168.82.2 (port 8080) after 1000ms; nested exception is java.net.SocketTimeoutException: failed to connect to /192.168.82.2 (port 8080) after 1000ms
1 : failed to connect to /192.168.82.2 (port 8080) after 1000ms

FERMER 3

Le 'clic' sur le bouton [Annuler] est géré avec la méthode suivante :

```

1.  @Click(R.id.btn_Annuler)
2.  protected void doAnnuler() {
3.      // mémoire
4.      hasBeenCanceled=true;
5.      // on annule la tâche asynchrone
6.      BackgroundExecutor.cancelAll("alea", true);
7.      // fin de l'attente
8.      cancelWaiting();
9.  }
```

- ligne 4 : on note que l'utilisateur a annulé sa demande ;
- ligne 6 : annule toutes les tâches identifiées par la chaîne [alea]. Le second paramètre [true] signifie qu'elles doivent être annulées même si elles ont déjà été lancées. L'identifiant [alea] est celui utilisé pour qualifier la méthode [getAlea] du fragment (ligne 1 ci-dessous) :

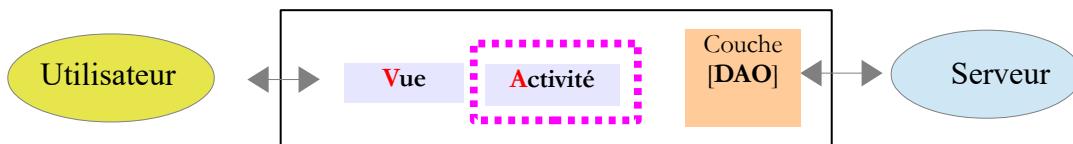
```

1.  @Background(id = "alea")
2.  void getAlea(int a, int b) {
3.      ...
4.  }

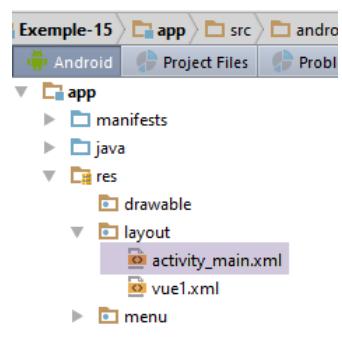
```

Note : il s'est avéré que la ligne 6 du code de la méthode [doAnnuler] marchait incorrectement. C'est pour cette raison qu'on a ajouté le booléen [hasBeenCalled]. En effet, en cas d'exception (serveur absent), on récupérait n fois la fenêtre d'alerte si on avait demandé n nombres aléatoires.

1.16.2.7 L'activité [MainActivity]



1.16.2.7.1 La vue [activity-main.xml]



Par rapport à l'exemple précédent, nous avons ajouté une image d'attente dans la vue associée à l'activité [MainActivity] :

```

1.  ...
2.  <android.support.design.widget.AppBarLayout
3.      android:id="@+id/appbar"
4.      android:layout_width="match_parent"
5.      android:layout_height="wrap_content"
6.      android:paddingTop="@dimen/appbar_padding_top"
7.      android:theme="@style/AppTheme.AppBarOverlay">
8.
9.      <android.support.v7.widget.Toolbar
10.         android:id="@+id/toolbar"
11.         android:layout_width="match_parent"
12.         android:layout_height="?attr/actionBarSize"
13.         android:background="?attr/colorPrimary"
14.         app:popupTheme="@style/AppTheme.PopupOverlay"
15.         app:layout_scrollFlags="scroll/enterAlways">
16.             <!-- image d'attente -->
17.             <ProgressBar
18.                 android:id="@+id>LoadingPanel"
19.                 android:layout_width="wrap_content"
20.                 android:layout_height="wrap_content"
21.                 android:indeterminate="true"/>
22.
23.             </android.support.v7.widget.Toolbar>
24.             <!-- image d'attente -->
25.         </android.support.design.widget.AppBarLayout>
26. ...

```

- lignes 17-21 : l'image d'attente ;

1.16.2.7.2 L'activité [MainActivity]

L'activité [MainActivity] bouge peu vis à vis de ce qu'elle était dans [Exemple-14]. Tout d'abord, on lui injecte la couche [DAO] :

```

1.  // injection dao
2.  @Bean(Dao.class)
3.  protected IDao dao;
4.  ...
5.  @AfterInject
6.  protected void afterInject() {
7.      // log
8.      if (IS_DEBUG_ENABLED) {
9.          Log.d("MainActivity", "afterInject");
10.     }
11.     // on paramètre la couche [DAO]
12.     setTimeout(TIMEOUT);
13. }
```

- lignes 2-3 : injection de la couche [DAO] par une annotation AA ;
- lignes 5-13 : code exécuté après cette injection ;
- ligne 12 : on paramètre le *timeout* de la couche [DAO]

Par ailleurs, l'activité [MainActivity] doit implémenter l'interface [IMainActivity] qui elle-même étend l'interface [IDao] :

```

1.  // implémentation IMainActivity -----
2.  @Override
3.  public void navigateToView(int position) {
4.      // on affiche la vue position
5.      if (mViewPager.getCurrentItem() != position) {
6.          // affichage fragment
7.          mViewPager.setCurrentItem(position);
8.      }
9.  }
10.
11. // gestion de l'image d'attente
12. public void cancelWaiting() {
13.     loadingPanel.setVisibility(View.INVISIBLE);
14. }
15.
16. public void beginWaiting() {
17.     loadingPanel.setVisibility(View.VISIBLE);
18. }
19.
20. // implémentation IDao -----
21.
22. @Override
23. public int getAlea(int a, int b) {
24.     // exécution
25.     return dao.getAlea(a, b);
26. }
27.
28. @Override
29. public void setDelay(int delay) {
30.     dao.setDelay(delay);
31. }
32.
33. @Override
34. public void setUrlServiceWebJson(String url) {
35.     dao.setUrlServiceWebJson(url);
36. }
37.
38. @Override
39. public void setTimeout(int timeout) {
40.     dao.setTimeout(timeout);
41. }
```

1.16.2.8 Exécution du projet

Lancez le service web (paragraphe 1.16.1.7, page 172) puis lancez le client Android :

Exemple-15

Génération de N nombres aléatoires

Valeur de N :

Intervalle $[a,b]$ de génération, a : b :

URL du service web : Ip1.Ip2.Ip3.IP4:8080 1

Délai d'attente en ms avant chaque appel au service web : —

EXÉCUTER

Pour savoir quoi mettre en [1], procédez comme suit. Ouvrez une fenêtre de commande et tapez la commande suivante :

Si vous avez installé [GenyMotion], la machine virtuelle VirtualBox a ajouté des adresses IP à votre poste (lignes 10 et 18). Ces adresses sont particulièrement pratiques car elles ne sont pas bloquées par le pare-feu de Windows. La ligne 30 donne l'adresse IP de votre poste sur un réseau local. Pour utiliser cette adresse, il faut en général inhiber le pare-feu de Windows. Si vous êtes connecté à un réseau wifi, utilisez l'adresse wifi et là aussi, inhibez le pare-feu si vous en avez un.

Testez l'application dans les cas suivants :

- 100 nombres aléatoires dans l'intervalle [1000, 2000] sans délai d'attente ;
 - 2000 nombres aléatoires dans l'intervalle [10000, 20000] sans délai d'attente et annulez l'attente avant la fin de la génération ;

- 5 nombres aléatoires dans l'intervalle [100,200] avec un délai d'attente de 5000 ms et annulez l'attente avant la fin de la génération ;

1.16.2.9 Gestion de l'annulation

Pour suivre ce qui se passe lorsque l'utilisateur demande l'annulation ou que celle-ci est demandée parce qu'une exception s'est produite, nous ajoutons la méthode suivante à l'interface [IDao] (cf paragraphe 1.16.2.4.1, page 181) :

```

1. package exemples.android.dao;
2.
3. public interface IDao {
4.
5.     ...
6.
7.     // mode debug
8.     void setDebugMode(boolean isDebugEnabled);
9. }
```

Dans la classe [Dao], nous ajoutons le code suivant :

```

1.     // mode debug
2.     private boolean isDebugEnabled;
3.     // nom de la classe
4.     private String className;
5.     ...
6.     // constructeur
7.     public Dao() {
8.         // nom de la classe
9.         className = getClass().getSimpleName();
10.    }
11.   ...
12.   // interface IDao -----
13.   @Override
14.   public int getAlea(int a, int b) {
15.       // log
16.       if (isDebugEnabled) {
17.           Log.d(String.format("%s", className), String.format("getAlea [%s, %s] en cours", a, b));
18.       }
19.       // exécution service
20.       Response<Integer> info;
21.       ...
22.   @Override
23.   public void setDebugMode(boolean isDebugEnabled) {
24.       this.isDebugEnabled = isDebugEnabled;
25.   }
```

- ligne 9 : nous notons le nom de la classe ;
- lignes 16-18 : nous écrivons un log à chaque fois que la méthode [getAlea] est appelée ;

Par ailleurs, dans le fragment [Vue1Fragment], nous ajoutons les logs suivants :

```

1.     @UiThread
2.     protected void showInfo(int alea) {
3.         // log
4.         if (isDebugEnabled) {
5.             Log.d(String.format("%s", className), String.format("showInfo(%s)", alea));
6.         }
7.         ....
8.     }
9.
10.    @UiThread
11.    protected void showAlert(Throwable th) {
12.        // log
13.        if (isDebugEnabled) {
14.            Log.d(String.format("%s", className), "Exception reçue");
15.        }
16.        ...
17.    }
18. }
19.
20. @Click(R.id.btn_Annuler)
21. protected void doAnnuler() {
22.     // log
23.     if (isDebugEnabled) {
24.         Log.d(String.format("%s", className), "Annulation demandée");
25.     }
26.     ...
27. }
```

A chaque fois que le fragment [Vue1Fragment] reçoit une information de la couche [DAO], un log est émis. Par ailleurs, lorsque la méthode [doAnnuler] est appelée, on logue l'événement.

Test 1

On demande 5 nombres alors que le serveur n'a pas été lancé. On a les logs suivants :

```
1. 06-06 08:48:51.571 15317-16201/exemples.android D/Dao_: getAlea [100, 200] en cours
2. 06-06 08:48:51.576 15317-16202/exemples.android D/Dao_: getAlea [100, 200] en cours
3. 06-06 08:48:51.585 15317-16204/exemples.android D/Dao_: getAlea [100, 200] en cours
4. 06-06 08:48:51.586 15317-16203/exemples.android D/Dao_: getAlea [100, 200] en cours
5. 06-06 08:48:51.593 15317-16205/exemples.android D/Dao_: getAlea [100, 200] en cours
6. ...
7. 06-06 08:48:53.568 15317-15317/exemples.android D/Vue1Fragment_: Exception reçue
8. 06-06 08:48:53.568 15317-15317/exemples.android D/Vue1Fragment_: Annulation demandée
9. 06-06 08:48:53.587 15317-15317/exemples.android D/Vue1Fragment_: Exception reçue
10. 06-06 08:48:53.587 15317-15317/exemples.android D/Vue1Fragment_: Exception reçue
11. 06-06 08:48:53.587 15317-15317/exemples.android D/Vue1Fragment_: Exception reçue
12. 06-06 08:48:53.587 15317-15317/exemples.android D/Vue1Fragment_: Exception reçue
```

- lignes 1-5 : la méthode [getAlea] de la classe [Dao] est appelée cinq fois. On rappelle que ce sont des appels asynchrones faits par le fragment [VueFragment] et que celui-ci n'attend pas le résultat de son appel ;
- ligne 7 : la première requête HTTP a eu lieu et le fragment [VueFragment] a reçu sa première exception ;
- ligne 8 : il demande alors l'annulation de toutes les demandes ;
- lignes 9-12 : on voit cependant qu'il reçoit les quatre exceptions suivantes. Donc les requêtes asynchrones qui étaient en attente ont toutes été exécutées ;

Test 2

Maintenant, lançons le serveur et demandons 5 nombres avec un délai de 5 secondes et cliquons sur [Annuler] avant la fin de ce délai. Les logs sont les suivants :

```
1. 06-06 09:12:38.360 4640-5054/exemples.android D/Dao_: getAlea [100, 200] en cours
2. 06-06 09:12:38.360 4640-5055/exemples.android D/Dao_: getAlea [100, 200] en cours
3. 06-06 09:12:38.361 4640-5056/exemples.android D/Dao_: getAlea [100, 200] en cours
4. 06-06 09:12:38.362 4640-5057/exemples.android D/Dao_: getAlea [100, 200] en cours
5. 06-06 09:12:38.363 4640-5058/exemples.android D/Dao_: getAlea [100, 200] en cours
6. ...
7. 06-06 09:12:39.895 4640-4640/exemples.android D/Vue1Fragment_: Annulation demandée
8. 06-06 09:29:56.313 1616-1616/exemples.android D/Vue1Fragment_: showInfo(185)
9. 06-06 09:29:56.313 1616-1616/exemples.android D/Vue1Fragment_: showInfo(185)
10. 06-06 09:29:56.313 1616-1616/exemples.android D/Vue1Fragment_: showInfo(185)
11. 06-06 09:30:00.150 1616-1616/exemples.android D/Vue1Fragment_: showInfo(157)
12. 06-06 09:30:00.151 1616-1616/exemples.android D/Vue1Fragment_: showInfo(157)
```

- lignes 1-5 : la méthode [getAlea] de la classe [Dao] est appelée cinq fois ;
- ligne 7 : l'utilisateur a demandé l'annulation des requêtes ;
- ligne 8 : on voit que [Vue1_Fragment] reçoit 5 valeurs. De nouveau, les requêtes asynchrones qui étaient en attente ont toutes été exécutées ;

C'est la raison pour laquelle nous avons du gérer un booléen [hasBeenCanceled] afin d'éviter d'afficher quelque chose alors qu'une annulation avait été demandée. Dans le code de l'annulation :

```
1. @Click(R.id.btn_Annuler)
2. protected void doAnnuler() {
3.     // log
4.     if (isDebugEnabled) {
5.         Log.d(String.format("%s", className), "Annulation demandée");
6.     }
7.     // mémoire
8.     hasBeenCanceled = true;
9.     // on annule la tâche asynchrone
10.    BackgroundExecutor.cancelAll("alea", true);
11.    // fin de l'attente
12.    cancelWaiting();
13. }
```

le code de la ligne 10 ne fait pas ce qui est attendu. Il est possible que ce soit parce que les tâches asynchrones se partagent la même méthode annotée [@Background] :

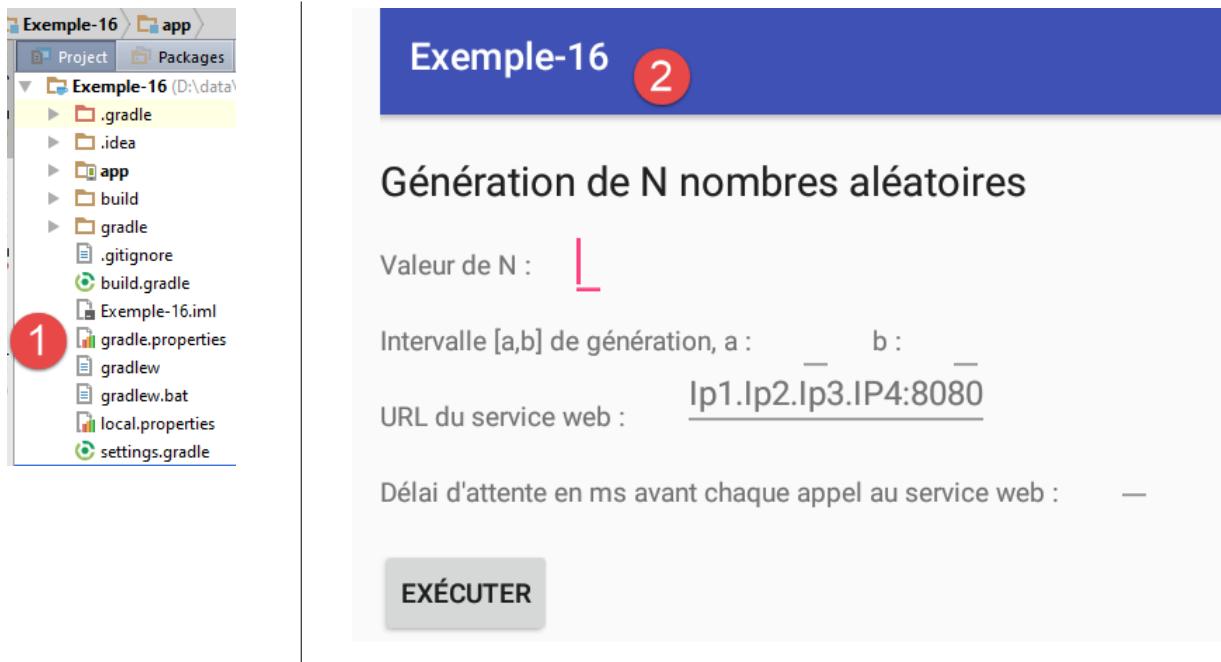
```
1. @Background(id = "alea")
2. void getAlea(int a, int b) {
3.     ...
4. }
```


1.17 Exemple-16 : gérer l'asynchronisme avec RxAndroid

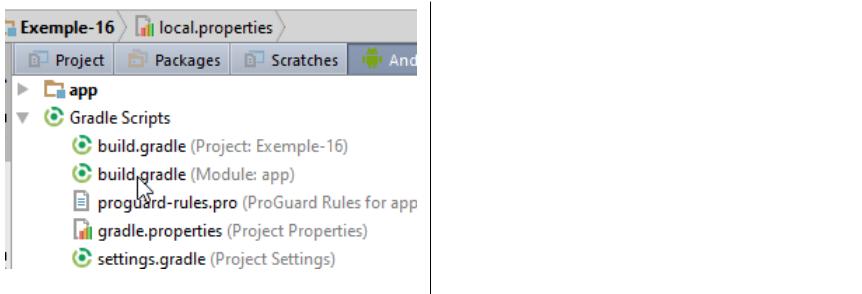
Nous nous proposons maintenant de gérer l'asynchronisme nécessaire aux applications Android avec une bibliothèque appelée RxJava [<http://reactivex.io/>] et sa version dérivée pour l'environnement Android [RxAndroid]. Pour cela, nous utiliserons le cours [[Introduction à RxJava. Application aux environnements Swing et Android](#)].

1.17.1 Création du projet

Nous dupliquons le projet [Exemple-1] dans [Exemple-16] :



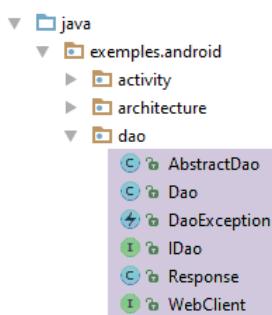
1.17.2 Configuration Gradle



Dans [build.gradle], nous ajoutons la dépendance sur la bibliothèque [RxAndroid] :

```
1. dependencies {
2.     ...
3.     compile 'io.reactivex:rxandroid:1.2.0'
4. }
```

1.17.3 La couche [DAO]



1.17.4 L'interface [IDao]

L'interface [IDao] devient la suivante :

```
1. package exemples.android;
2.
3. import rx.Observable;
4.
5. public interface IDao {
6.
7.     // nombre aléatoire
8.     Observable<Integer> getAlea(int a, int b);
9.
10.    // URL du service web
11.    void setUrlServiceWebJson(String url);
12.
13.    // délai d'attente (ms) max de la réponse du serveur
14.    void setTimeout(int timeout);
15.
16.    // délai d'attente en millisecondes du client avant requête
17.    void setDelay(int delay);
18.
19.    // mode debug
20.    void setDebugMode(boolean isDebugEnabled);
21. }
```

- ligne 8 : la méthode [getAlea] rend désormais un type [Observable] de la bibliothèque RxJava (ligne 3). Le principe est le suivant :

Un flux d'éléments de type Observable<T> est observé par un ou plusieurs souscripteurs (abonnés, observateurs, consommateurs) de type Subscriber<T>. La bibliothèque RxJava permet que le flux Observable<T> s'exécute dans un thread T1 et son observateur Subscriber<T> dans un thread T2 sans que le développeur n'ait à se soucier de gérer le cycle de vie de ces threads et de problèmes naturellement difficiles, tels que le partage de données entre threads et la synchronisation de ceux-ci pour exécuter une tâche globale. Elle facilite donc la programmation asynchrone.

1.17.5 La classe [AbstractDao]

Nous ferons dériver la classe [Dao] de la classe [AbstractDao] suivante :

```
1. package exemples.android;
2.
3. import com.fasterxml.jackson.core.JsonProcessingException;
4. import com.fasterxml.jackson.databind.ObjectMapper;
5. import rx.Observable;
6. import rx.Subscriber;
7.
8. public abstract class AbstractDao {
9.
10.    // mappeur JSON
11.    private ObjectMapper mapper = new ObjectMapper();
12.
13.    // méthodes protégées -----
14.    // interface générique
15.    protected interface IRequest<T> {
16.        Response<T> getResponse();
17.    }
```

```

18.
19.    // requête générique
20.    protected <T> Observable<T> getResponse(final IRequest<T> request) {
21.        // exécution service
22.        return rx.Observable.create(new rx.Observable.OnSubscribe<T>()) {
23.            @Override
24.            public void call(Subscriber<? super T> subscriber) {
25.                DaoException ex = null;
26.                // exécution service
27.                try {
28.                    // on fait la requête et on fait suivre la réponse à l'abonné
29.                    Response<T> response = request.getResponse();
30.                    // erreur ?
31.                    int status = response.getStatus();
32.                    if (status != 0) {
33.                        // on note l'exception
34.                        ex = new DaoException(mapper.writeValueAsString(response.getMessages()), status);
35.                    } else {
36.                        // on émet la réponse
37.                        subscriber.onNext(response.getBody());
38.                        // on signale la fin de l'observable
39.                        subscriber.onCompleted();
40.                    }
41.                } catch (JsonProcessingException | RuntimeException e) {
42.                    // on note l'exception
43.                    ex = new DaoException(e, 100);
44.                }
45.                // exception ?
46.                if (ex != null) {
47.                    // on émet l'exception
48.                    subscriber.onError(ex);
49.                }
50.            }
51.        });
52.    }
53. }
54. }
```

- la classe [AbstractDao] a comme principal élément une méthode générique [getResponse] pour obtenir du serveur un type [Response<T>] où T est le type du résultat désiré par le client HTTP (ici Integer) ;
- ligne 20 : l'unique paramètre de la méthode générique [getResponse] est une instance de l'interface générique [IRequest<T>] des lignes 15-17. Elle n'a qu'une méthode [getResponse] et c'est cette méthode qui fournit la réponse [Response<T>] désirée ;
- grâce aux deux éléments précédents, la classe [AbstractDao] peut servir de classe parent à toute couche [Dao] cliente d'un serveur envoyant des types [Response<T>] ;
- ligne 20 : la méthode générique [getResponse] rend un type [Observable<T>] qui représente le résultat réellement attendu par le client HTTP (ici un type Observable<Integer>) ;
- lignes 22-51 : la méthode statique [rx.Observable.create] crée un type [Observable] ;
- ligne 22 : l'unique paramètre de cette méthode est une instance du type [rx.Observable.OnSubscribe<T>], une interface qui possède les méthodes suivantes :
 - [onNext(T element)] : permet d'émettre vers un observateur un élément de type T ;
 - [onError(Throwable th)] : permet d'émettre vers un observateur une exception ;
 - [onCompleted] : permet d'indiquer à un observateur la fin des émissions ;

Un type [Observable<T>] obéit à certaines contraintes :

- il émet ses éléments avec la méthode [onNext(T element)] ;
- la méthode [onCompleted] doit être appelée une unique fois dès qu'il n'y a plus d'éléments à émettre vers l'observateur ;
- la méthode [onCompleted] n'est pas appelée si la méthode [onError(Throwable th)] l'a été ;

Dans notre exemple :

- l'observateur sera le fragment [Vue1Fragment]. C'est lui qui consomme les éléments émis par le [Observable<T>] (élément ou exception) ;
- le type [Observable<T>] créé n'émettra qu'un unique élément (ligne 37) ;
- ligne 29 : fait une requête HTTP **synchrone** au serveur et obtient le type [Response<T>]. Cette requête HTTP est assurée par le type [IRequest] passé en paramètre à la méthode générique [getResponse] ;
- ligne 31 : on récupère le *status* de la réponse ;
- lignes 32-34 : si ce *status* est celui d'une erreur, on prépare une exception ;
- lignes 36-39 : si ce *status* n'est pas celui d'une erreur, alors on émet la réponse attendue réellement par le client (ligne 37) et on indique à l'observateur qu'il n'y aura plus d'autres émissions (ligne 39) ;
- lignes 41-44 : si la requête HTTP se termine par une exception, on note celle-ci ;
- lignes 46-49 : si l'exception [ex] est différente de *null*, alors on l'émet vers l'observateur. Il n'y a là pas lieu d'appeler la méthode [onCompleted] pour indiquer à l'observateur qu'il n'y aura plus d'émissions d'éléments. C'est implicite ;

On retiendra de ces explications que :

- la méthode générique [$<T>$ Observable< T > getResponse(final IRequest< T > request)] rend un type [Observable< T >] qui n'émet qu'un élément de type T ou bien une exception ;
- que cette méthode admet pour unique paramètre un type [IRequest< T >] dont l'unique méthode [getResponse()] réalise l'accès HTTP qui renvoie le type [Response< T >] ;

1.17.6 La classe [Dao]

La classe [Dao] évolue de la façon suivante :

```
1. @EBean
2. public class Dao extends AbstractDao implements IDao {
3.
4.     // client du service REST
5.     @RestService
6.     protected WebClient webClient;
7.
8.     // délai d'attente avant exécution requête
9.     private int delay;
10.    // mode debug
11.    private boolean isDebugEnabled;
12.    // nom de la classe
13.    private String className;
14.
15.    // constructeur
16.    public Dao() {
17.        // nomde la classe
18.        className = getClass().getSimpleName();
19.    }
20.
21.
22.    // interface IDao -----
23.    @Override
24.    public Observable<Integer> getAlea(final int a, final int b) {
25.        // log
26.        if (isDebugEnabled) {
27.            Log.d(String.format("%s", className), String.format("getAlea [%s, %s] en cours", a, b));
28.        }
29.        // exécution client web
30.        return getResponse(new IRequest<Integer>() {
31.            @Override
32.            public Response<Integer> getResponse() {
33.                // attente
34.                waitSomeTime(delay);
35.                // appel HTTP
36.                return webClient.getAlea(a, b);
37.            }
38.        });
39.    }
40.    ...
41.
```

- ligne 2 : la classe [Dao] étend la classe [AbstractDao] ;
- ligne 24 : la méthode [getAlea] rend désormais un type [Observable<Integer>] ;
- ligne 30 : appel de la méthode générique [getResponse] de la classe parent. On lui passe un paramètre de type [IRequest<Integer>] ;
- lignes 32-37 : implémentation de l'interface [IRequest<Integer>] ;
- ligne 36 : on fait la requête HTTP via l'interface AA [webClient] comme il avait été fait précédemment. On sait qu'on va récupérer un type [Response<Integer>] qui est bien le type que doit rendre la méthode [IRequest<Integer>.getReponse()] ;
- ligne 36 : on utilise ici une propriété appelée *closure* : la capacité d'encapsuler dans une instance des valeurs extérieures à celle-ci lorsqu'elle est créée, ici les valeurs de [a, b] de la ligne 24. C'est ce qui permet à la méthode [IRequest<Integer>.getReponse()] de ne pas avoir de paramètres. Ceux-ci ont été gravés dans le corps de la méthode. Et là où normalement on changerait les paramètres de la méthode (a,b) -> (x,y), ici on crée une nouvelle instance de [IRequest<Integer>] encapsulant les valeurs de x et y ;

1.17.7 La classe [MainActivity]

La classe [MainActivity] qui implémente l'interface [IDao] évolue comme suit :

```
1.    // implémentation IDao -----
2.
3.    @Override
4.    public Observable<Integer> getAlea(int a, int b) {
5.        // exécution
6.        return dao.getAlea(a, b);
7.
```

1.17.8 La classe [Vue1Fragment]

La classe [Vue1Fragment] évolue de la façon suivante :

```

1.  @Click(R.id.btn_Executer)
2.  protected void doExecuter() {
3.      // on efface les réponses précédentes
4.      reponses.clear();
5.      adapterReponses.notifyDataSetChanged();
6.      hasBeenCanceled = false;
7.      // on remet à 0 le compteur de réponses
8.      nbInfos = 0;
9.      infoReponses.setText(String.format("Liste des réponses (%s)", nbInfos));
10.     // on teste la validité des saisies
11.     if (!isValid()) {
12.         return;
13.     }
14.     // initialisation activité
15.     mainActivity.setUrlServiceWebJson(urlServiceWebJson);
16.     mainActivity.setDelay(delay);
17.     // on demande les nombres aléatoires
18.     getAleasInBackground(a, b);
19.     // on commence l'attente
20.     beginWaiting();
21. }
```

- ligne 18 : on demande les nombres aléatoires à la méthode [getAleasInBackground] appelée ainsi parce que les nombres vont être demandés dans un thread différent de celui de l'Ui ;

```

1.  private int nbReponses = 0;
2.  // les abonnements aux observables
3.  private List<Subscription> abonnements;
4.
5.  // annotation [Background] inutile
6.  void getAleasInBackground(int a, int b) {
7.      // au départ pas de réponses et pas d'abonnements
8.      nbReponses = 0;
9.      abonnements.clear();
10.     // on prépare l'observable
11.     Observable<Integer> response = Observable.empty();
12.     // on fusionne les résultats des différents appels HTTP
13.     // ils sont exécutés sur un thread d'E/S
14.     for (int i = 0; i < nbAleas; i++) {
15.         response = response.mergeWith(mainActivity.getAlea(a, b).subscribeOn(Schedulers.io()));
16.     }
17.     // l'observable cumulé sera observé sur le thread de l'UI
18.     response = response.observeOn(AndroidSchedulers.mainThread());
19.     try {
20.         // on exécute l'observable
21.         abonnements.add(response.subscribe(new Action1<Integer>() {
22.             @Override
23.             public void call(Integer alea) {
24.                 // on ajoute l'information à la liste des réponses
25.                 showInfo(alea);
26.             }
27.             new Action1<Throwable>() {
28.                 @Override
29.                 public void call(Throwable th) {
30.                     // messages
31.                     showAlert(th);
32.                     // fin attente
33.                     doAnnuler();
34.                 }
35.             }, new Action0() {
36.                 @Override
37.                 public void call() {
38.                     // fin attente
39.                     cancelWaiting();
40.                 }
41.             });
42.         } catch (RuntimeException e) {
43.             // on affiche l'exception dans l'UiThread
44.             showAlert(e);
45.         }
46.     }
```

- ligne 3 : un observable a des abonnés. Le lien entre un abonné et le processus qu'il observe est appelé un abonnement (Subscription). Ici nous n'aurons qu'un processus observé et qu'un abonné. Nous n'aurons donc qu'un abonnement. Pour le principe, nous faisons comme si nous pouvions avoir plusieurs processus observés par différents observateurs, ce qui ferait plusieurs abonnements ;

- lignes 11-18 : on configure le processus observé (observable). Il faut comprendre que ce n'est que de la configuration : le processus n'est pas exécuté ;
- ligne 11 : on part d'un observable vide, un observable qui n'émet rien ;
- lignes 14-16 : à cet observable vide, on ajoute les [nbAleas] observables qui seront les [nbAleas] requêtes HTTP qui ramèneront [nbAleas] nombres aléatoires ;
- ligne 15 : comme précédemment, le nombre aléatoire n° i est demandé à la classe [MainActivity]. Il faut vraiment comprendre qu'ici, aucune requête HTTP n'est encore exécuté. La méthode [mainActivity.getAlea(a, b)] est exécutée et rend un type [Observable<Integer>]. C'est un processus qui sera observé lorsqu'il sera lancé ;
- ligne 15 : la méthode [subscribeOn(Schedulers.io())] demande à ce que le processus soit exécuté (lorsqu'il le sera) sur un thread d'E/S. La bibliothèque RxJava offre différents types de thread. Celui d'E/S est adapté aux appels HTTP ;
- ligne 15 : l'observable n° i est fusionné à l'observable initial de la ligne 11 : des [nbAleas] Observables émettant chacun un élément, on crée un observable qui lui émettra [nbAleas] éléments. C'est lui qui sera observé. Cet observable émet la notification [onCompleted] lorsque tous les observables qui le composent auront émis leur propre notification [onCompleted]. Cela nous évitera d'avoir à compter les réponses, comme nous l'avions fait dans la version précédente, pour savoir si on a reçu tous les nombres attendus ;
- ligne 18 : lorsqu'on arrive là, on a configuré un observable qui est la composition de [nbAleas] observables s'exécutant chacun sur un thread d'E/S ;
- ligne 18 : la méthode [observeOn(AndroidSchedulers.mainThread())] sert à dire sur quel thread l'observation des valeurs émises par l'observable doit se faire. Ici le thread [AndroidSchedulers.mainThread()] appartient à la bibliothèque RxAndroid et non RxJava. Il désigne le thread de l'Ui appelé *event loop*. Ce point est important : dans une application Android, la modification d'un composant de l'Ui ne peut se faire que dans le thread de l'Ui, sinon on a une exception ;
- lignes 19-45 : maintenant que le processus à observer a été configuré, on l'exécute ;
- ligne 21 : c'est l'opération [Observable.subscribe] qui lance l'exécution du processus observé. Cette opération va lancer les [nbAleas] processus asynchrones configurés précédemment. Les résultats de ceux-ci seront automatiquement mis à disposition de l'observateur sur le thread de l'Ui ;
- nous nous rappelons que l'observable émet trois types d'événements :
 - [onNext] : lorsqu'il émet un élément ;
 - [onError] : lorsqu'il a rencontré une exception ;
 - [onCompleted] : lorsqu'il signale qu'il ne va plus émettre ;

La méthode [Observable.subscribe] a pour paramètres, trois objets [Action1<Integer>, Action1<Throwable>, Action0] dont les méthodes [call] servent à traiter chacun de ces trois événements ;

- lignes 21-27 : le 1er paramètre de type [Action1<Integer>] sert à traiter l'événement [onNext]. Sa méthode [call] reçoit l'élément qui a été émis par l'observable (ligne 23) ;
- ligne 25 : on réutilise la méthode [showInfo] de l'exemple précédent ;
- lignes 27-35 : le second paramètre de type [Action1<Throwable>] sert à traiter l'événement [onError]. Sa méthode [call] reçoit l'exception qui a été émis par l'observable (ligne 29) ;
- ligne 31 : on réutilise la méthode [showAlert] de l'exemple précédent ;
- ligne 33 : on lance la procédure d'annulation de la demande de l'utilisateur. Cela va consister à annuler tous les observables qui sont en cours d'exécution ;
- lignes 35-41 : le troisième paramètre de type [Action0] sert à traiter l'événement [onCompleted]. Sa méthode [call] ne reçoit aucun paramètre ;
- ligne 39 : on annule l'attente ;

La méthode [showInfo] évolue de la façon suivante :

```

1. // annotation [UiThread] inutile
2. protected void showInfo(int alea) {
3.     // log
4.     if (isDebugEnabled) {
5.         Log.d(String.format("%s", className), String.format("showInfo(%s)", alea));
6.     }
7.     if (!hasBeenCalled) {
8.         // une info de plus
9.         nbInfos++;
10.        infoResponses.setText(String.format("Liste des réponses (%s)", nbInfos));
11.        // on ajoute l'information à la liste des réponses
12.        responses.add(0, String.valueOf(alea));
13.        // on affiche les réponses
14.        adapterResponses.notifyDataSetChanged();
15.    }
16. }
```

La méthode présente deux changements :

- ligne 1 : on a enlevé l'annotation AA [UiThread] ;

- on ne compte plus les réponses pour savoir si on doit arrêter ou non l'attente. C'est désormais l'événement [onCompleted] de l'observable qui nous donne cette information ;

La méthode [showAlert] évolue de la façon suivante :

```

1.  // annotation [UiThread] inutile
2.  protected void showAlert(Throwable th) {
3.      // log
4.      if (isDebugEnabled) {
5.          Log.d(String.format("%s", className), "Exception reçue");
6.      }
7.      if (!hasBeenCalled) {
8.          // on annule tout
9.          doAnnuler();
10.         // on l'affiche
11.         new AlertDialog.Builder(activity).setTitle("Des erreurs se sont
produites").setMessage(Utils.getMessagesForAlert(th)).setNeutralButton("Fermer", null).show();
12.     }
13. }
```

- le seul changement est ligne 1 : on a enlevé l'annotation AA [`@UiThread`] ;

Enfin, la méthode [doAnnuler] évolue comme suit :

```

1.  @Click(R.id.btn_Annuler)
2.  protected void doAnnuler() {
3.      // log
4.      if (isDebugEnabled) {
5.          Log.d(String.format("%s", className), "Annulation demandée");
6.      }
7.      // mémoire
8.      hasBeenCalled = true;
9.      // on annule les tâches asynchrones
10.     if (abonnements != null) {
11.         for (Subscription abonnement : abonnements) {
12.             abonnement.unsubscribe();
13.         }
14.     }
15.     // fin de l'attente
16.     cancelWaiting();
17. }
```

- ligne 12 : annule un abonnement et donc l'observation du processus associé ;

1.17.9 Exécution

Lancez le service web (paragraphe 1.16.1.7, page 172), lancez le client Android et refaites les tests que vous avez faits avec l'exemple précédent (paragraphe 1.16.2.8, page 196).

1.17.10 Gestion de l'annulation

On refait les mêmes tests que pour l'exemple précédent (paragraphe 1.16.2.9, page 198).

Test 1

On demande 5 nombres alors que le serveur n'a pas été lancé. On a les logs suivants :

```

1.  06-07 05:48:09.790 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
2.  06-07 05:48:09.791 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
3.  06-07 05:48:09.791 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
4.  06-07 05:48:09.791 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
5.  06-07 05:48:09.791 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
6.  06-07 05:48:11.789 28272-28272/exemples.android D/Vue1Fragment_: Exception reçue
7.  06-07 05:48:11.789 28272-28272/exemples.android D/Vue1Fragment_: Annulation demandée
```

Après la ligne 7, il n'y a plus de logs ce qui montre que l'observateur (Vue1Fragment) ne reçoit plus de notifications du processus observé.

Test 2

Maintenant, lançons le serveur et demandons 5 nombres avec un délai de 5 secondes et cliquons sur [Annuler] avant la fin de ce délai. Les logs sont les suivants :

```
1.  06-07 05:52:22.675 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
```

```

2. 06-07 05:52:22.675 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
3. 06-07 05:52:22.675 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
4. 06-07 05:52:22.675 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
5. 06-07 05:52:22.675 28272-28272/exemples.android D/Dao_: getAlea [100, 200] en cours
6. 06-07 05:52:23.485 28272-28272/exemples.android D/Vue1Fragment_: Annulation demandée

```

Après la ligne 6, il n'y a plus de logs ce qui montre que l'observateur (Vue1Fragment) ne reçoit plus de notifications du processus observé.

On a là le comportement attendu d'une annulation. On peut donc dans le code de [Vue1Fragment] enlever le booléen [hasBeenCalled] que nous avions introduit dans l'exemple précédent parce que l'annulation ne faisait pas ce qu'on attendait d'elle.

Le fait que l'observateur ne reçoive plus de notification après l'annulation de l'observable, ne veut pas dire que les requêtes HTTP sont elles-mêmes annulées. Pour le voir, nous modifions la classe [Dao] de la façon suivante :

```

1.  @Override
2.  public Observable<Integer> getAlea(final int a, final int b) {
3.      // log
4.      if (isDebugEnabled) {
5.          Log.d(String.format("%s", className), String.format("getAlea [%s, %s] en cours", a, b));
6.      }
7.      // exécution client web
8.      return getResponse(new IRequest<Integer>() {
9.          @Override
10.         public Response<Integer> getResponse() {
11.             // attente
12.             waitSomeTime(delay);
13.             // appel HTTP
14.             Response<Integer> response= webClient.getAlea(a, b);
15.             if (isDebugEnabled) {
16.                 try {
17.                     Log.d(String.format("%s", className), String.format("response [%s]", new
18.                         ObjectMapper().writeValueAsString(response)));
19.                 } catch (JsonProcessingException e) {
20.                     Log.d(String.format("%s", className), "erreur désrialisation JSON");
21.                 }
22.             }
23.         }
24.     });
25. }

```

- lignes 15-21 : nous loguons le résultat de la requête HTTP de la ligne 14 ;

Les logs pour le test n° 2 sont alors les suivants :

```

1. 06-07 06:03:20.778 27085-27085/exemples.android D/Dao_: getAlea [100, 200] en cours
2. 06-07 06:03:20.784 27085-27085/exemples.android D/Dao_: getAlea [100, 200] en cours
3. 06-07 06:03:20.785 27085-27085/exemples.android D/Dao_: getAlea [100, 200] en cours
4. 06-07 06:03:20.785 27085-27085/exemples.android D/Dao_: getAlea [100, 200] en cours
5. 06-07 06:03:20.785 27085-27085/exemples.android D/Dao_: getAlea [100, 200] en cours
6. 06-07 06:03:21.493 27085-27085/exemples.android D/Vue1Fragment_: Annulation demandée
7. 06-07 06:03:21.636 27085-27440/exemples.android D/Dao_: response [{"body":176,"messages":null,"status":0}]
8. 06-07 06:03:21.636 27085-27442/exemples.android D/Dao_: response [{"body":145,"messages":null,"status":0}]
9. 06-07 06:03:21.636 27085-27439/exemples.android D/Dao_: response [{"body":197,"messages":null,"status":0}]
10. 06-07 06:03:21.636 27085-27438/exemples.android D/Dao_: response [{"body":136,"messages":null,"status":0}]
11. 06-07 06:03:21.636 27085-27441/exemples.android D/Dao_: response [{"body":136,"messages":null,"status":0}]

```

- lignes 1-5 : les 5 demandes ont été faites ;
- ligne 6 : l'utilisateur a annulé ;
- lignes 7-11 : on reçoit bien les réponses des cinq requêtes HTTP. Seulement, à cause de l'annulation de l'observable, ces éléments ne sont pas transmis à l'observateur ;

1.17.11 Conclusion

Dans la suite de ce document, les applications client / serveur seront réalisées avec la bibliothèque RxAndroid plutôt qu'avec la bibliothèque AA pour les raisons suivantes :

1. RxAndroid peut s'utiliser dans une application Android n'utilisant pas AA ;
2. RxAndroid fait plus que faciliter les opérations asynchrones. Il offre de très nombreuses méthodes pour créer un nouvel observable à partir d'un autre. Ces méthodes n'ont pas d'équivalent AA ;
3. dès qu'on veut dériver une classe annotée par AA, telle qu'un fragment, on rencontre de sérieux problèmes. On est alors amené à abandonner AA et à utiliser la solution 1 pour la programmation asynchrone ;

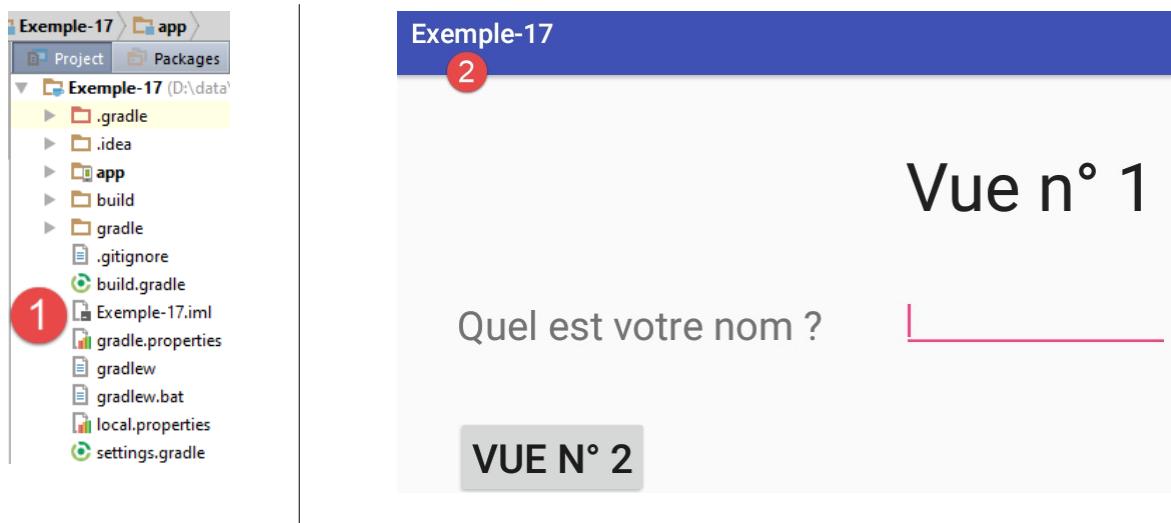
Le lecteur intéressé par approfondir les possibilités de la bibliothèque RxAndroid pourra lire le document [[Introduction à RxJava](#), [Application aux environnements Swing et Android](#)]. On y utilise RxAndroid sans la bibliothèque AA.

1.18 Exemple-17 : composants de saisie de données

Nous allons écrire un nouveau projet pour présenter quelques composants usuels dans les formulaires de saisie de données.

1.18.1 Crédation du projet

Nous dupliquons le projet [Exemple-13] dans [Exemple-17] :

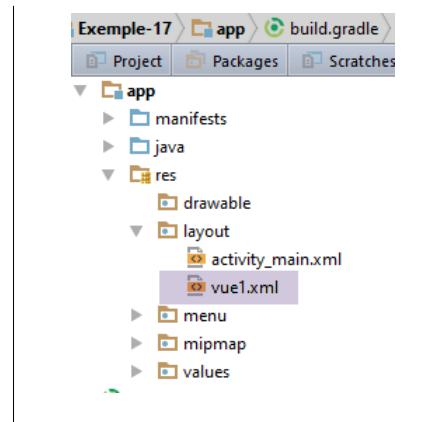


Le nouveau projet n'aura qu'une vue [vue1.xml]. Aussi supprimons-nous la vue [vue2.xml] et son fragment associé [Vue2Fragment] [2]. Nous prenons en compte cette modification dans le gestionnaire de fragments de [Mainactivity] :

```
1. // notre gestionnaire de fragments à redéfinir pour chaque application
2. // doit définir les méthodes suivantes : getItem, getCount, getPageTitle
3. public class SectionsPagerAdapter extends FragmentPagerAdapter {
4.
5.     // les fragments
6.     private final Fragment[] fragments = {new Vue1Fragment()};
7.     ...
8. }
```

Réexécutez le projet. Il doit faire apparaître la vue n° 1 comme précédemment. Nous allons travailler à partir de ce projet.

1.18.2 La vue XML du formulaire



La vue produite par le fichier [vue1.xml] est la suivante :

Exemple-17

Vue n° 1

Cases à cocher 1

Boutons Radio 1 2 3

Seek Bar

Champ de saisie multilignes

Booléen

Date

Heure

Champ de saisie

11 39 AM

12 : 40 PM

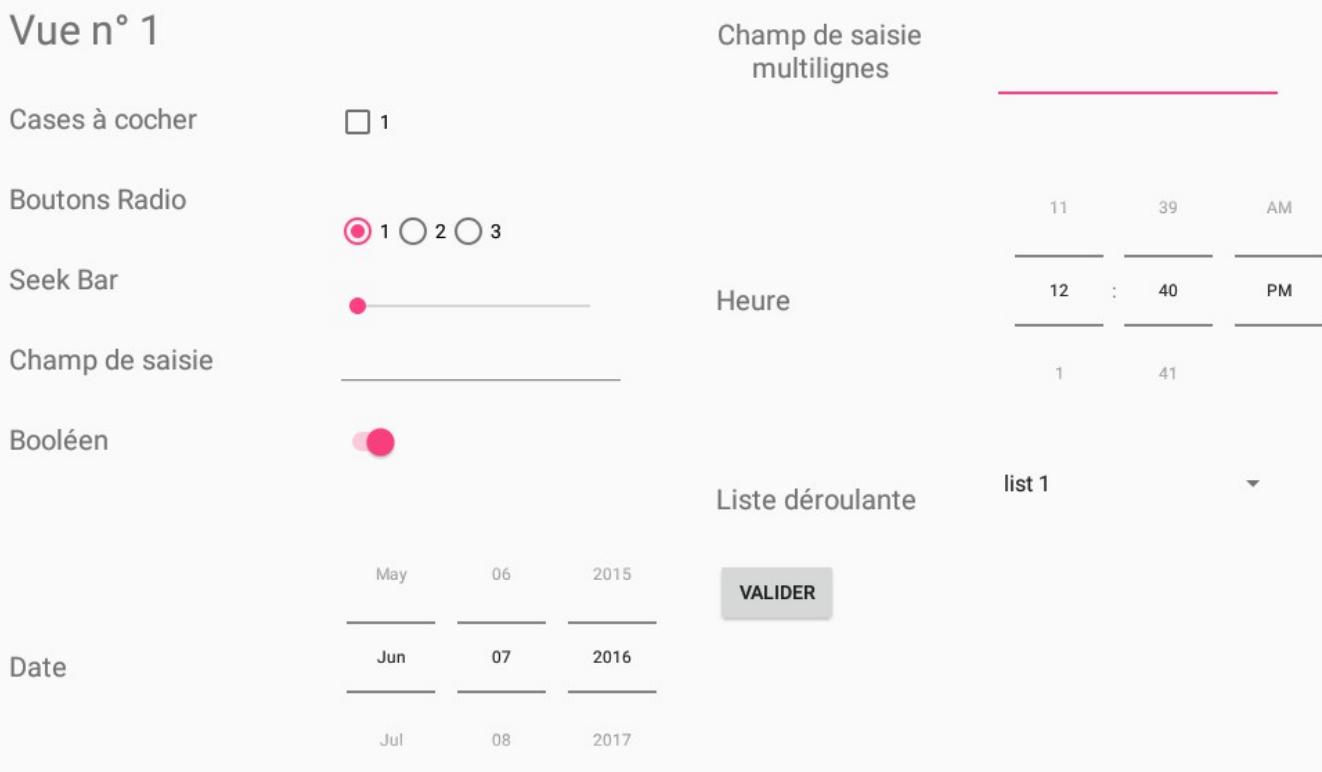
1 41

Liste déroulante list 1 ▾

May 06 2015 VALIDER

Jun 07 2016

Jul 08 2017



Le texte XML de la vue est le suivant :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:layout_marginBottom="30dp">
6.
7.     <RelativeLayout
8.         android:layout_width="match_parent"
9.         android:layout_height="wrap_content">
10.
11.        <TextView
12.            android:id="@+id/textViewFormulaireTitre"
13.            android:layout_width="wrap_content"
14.            android:layout_height="wrap_content"
15.            android:layout_alignParentLeft="true"
16.            android:layout_alignParentTop="true"
17.            android:layout_marginLeft="10dp"
18.            android:layout_marginTop="30dp"
19.            android:text="@string/titre_vue1"
20.            android:textSize="30sp"/>
21.
22.        <Button
23.            android:id="@+id/formulaireButtonValider"
24.            android:layout_width="wrap_content"
25.            android:layout_height="wrap_content"
26.            android:layout_alignLeft="@+id/TextViewFormulaireCombo"
27.            android:layout_below="@+id/TextViewFormulaireCombo"
28.            android:layout_marginTop="30dp"
29.            android:text="@string/formulaire_valider"/>
30.
31.        <TextView
32.            android:id="@+id/textViewFormulaireCheckBox"
33.            android:layout_width="wrap_content"
34.            android:layout_height="wrap_content"
35.            android:layout_alignLeft="@+id/textViewFormulaireTitre"
36.            android:layout_below="@+id/textViewFormulaireTitre"
37.            android:layout_marginTop="30dp"
38.            android:text="@string/formulaire_checkbox"
39.            android:textSize="20sp"/>
```

```

40.
41.    <TextView
42.        android:id="@+id/textViewFormulaireRadioButton"
43.        android:layout_width="wrap_content"
44.        android:layout_height="wrap_content"
45.        android:layout_alignLeft="@+id/textViewFormulaireCheckBox"
46.        android:layout_below="@+id/textViewFormulaireCheckBox"
47.        android:layout_marginTop="30dp"
48.        android:text="@string/formulaire_radioButton"
49.        android:textSize="20sp"/>
50.
51.    <TextView
52.        android:id="@+id/textViewFormulaireSeekBar"
53.        android:layout_width="wrap_content"
54.        android:layout_height="wrap_content"
55.        android:layout_alignLeft="@+id/textViewFormulaireRadioButton"
56.        android:layout_below="@+id/textViewFormulaireRadioButton"
57.        android:layout_marginTop="30dp"
58.        android:text="@string/formulaire_seekBar"
59.        android:textSize="20sp"/>
60.
61.    <TextView
62.        android:id="@+id/textViewFormulaireEdtText"
63.        android:layout_width="wrap_content"
64.        android:layout_height="wrap_content"
65.        android:layout_alignLeft="@+id/textViewFormulaireSeekBar"
66.        android:layout_below="@+id/textViewFormulaireSeekBar"
67.        android:layout_marginTop="30dp"
68.        android:text="@string/formulaire_saisie"
69.        android:textSize="20sp"/>
70.
71.    <TextView
72.        android:id="@+id/textViewFormulaireBool"
73.        android:layout_width="wrap_content"
74.        android:layout_height="wrap_content"
75.        android:layout_alignLeft="@+id/textViewFormulaireEdtText"
76.        android:layout_below="@+id/textViewFormulaireEdtText"
77.        android:layout_marginTop="30dp"
78.        android:text="@string/formulaire_bool"
79.        android:textSize="20sp"/>
80.
81.    <TextView
82.        android:id="@+id/textViewFormulaireDate"
83.        android:layout_width="wrap_content"
84.        android:layout_height="200dp"
85.        android:layout_alignLeft="@+id/textViewFormulaireBool"
86.        android:layout_below="@+id/textViewFormulaireBool"
87.        android:layout_marginTop="50dp"
88.        android:gravity="center"
89.        android:text="@string/formulaire_date"
90.        android:textSize="20sp"/>
91.
92.    <TextView
93.        android:id="@+id/textViewFormulaireMultilignes"
94.        android:layout_width="150dp"
95.        android:layout_height="100dp"
96.        android:gravity="center"
97.        android:layout_alignBaseline="@+id/textViewFormulaireTitre"
98.        android:layout_alignParentTop="true"
99.        android:layout_marginLeft="400dp"
100.       android:layout_toRightOf="@+id/textViewFormulaireTitre"
101.       android:text="@string/formulaire_multilignes"
102.       android:textSize="20sp"/>
103.
104.    <TextView
105.        android:id="@+id/textViewFormulaireTime"
106.        android:layout_width="wrap_content"
107.        android:layout_height="200dp"
108.        android:gravity="center"
109.        android:layout_alignLeft="@+id/textViewFormulaireMultilignes"
110.        android:layout_below="@+id/textViewFormulaireMultilignes"
111.        android:layout_marginTop="30dp"
112.        android:text="@string/formulaire_time"
113.        android:textSize="20sp"/>
114.
115.    <TextView
116.        android:id="@+id/TextViewFormulaireCombo"
117.        android:layout_width="wrap_content"
118.        android:layout_height="wrap_content"
119.        android:layout_alignLeft="@+id/textViewFormulaireTime"
120.        android:layout_below="@+id/textViewFormulaireTime"
121.        android:layout_marginTop="30dp"
122.        android:text="@string/formulaire_combo"
123.        android:textSize="20sp"/>
124.
125.    <CheckBox
126.        android:id="@+id/formulaireCheckBox1"

```

```

127.     android:layout_width="wrap_content"
128.     android:layout_height="wrap_content"
129.     android:layout_alignBaseline="@+id/textViewFormulaireCheckBox"
130.     android:layout_marginLeft="100dp"
131.     android:layout_toRightOf="@+id/textViewFormulaireCheckBox"
132.     android:text="@string/formulaire_checkbox1"/>
133.
134. <RadioGroup
135.     android:id="@+id/formulaireRadioGroup"
136.     android:layout_width="wrap_content"
137.     android:layout_height="wrap_content"
138.     android:layout_alignBaseline="@+id/textViewFormulaireRadioButton"
139.     android:layout_alignLeft="@+id/formulaireCheckBox1"
140.     android:orientation="horizontal">
141.
142.     <RadioButton
143.         android:id="@+id/formulaireRadioButton1"
144.         android:layout_width="wrap_content"
145.         android:layout_height="wrap_content"
146.         android:text="@string/formulaire_radiobutton1"/>
147.
148.     <RadioButton
149.         android:id="@+id/formulaireRadioButton2"
150.         android:layout_width="wrap_content"
151.         android:layout_height="wrap_content"
152.         android:text="@string/formulaire_radionbutton2"/>
153.
154.     <RadioButton
155.         android:id="@+id/formulaireRadionButton3"
156.         android:layout_width="wrap_content"
157.         android:layout_height="wrap_content"
158.         android:text="@string/formulaire_radiobutton3"/>
159. </RadioGroup>
160.
161. <SeekBar
162.     android:id="@+id/formulaireSeekBar"
163.     android:layout_width="200dp"
164.     android:layout_height="wrap_content"
165.     android:layout_alignBaseline="@+id/textViewFormulaireSeekBar"
166.     android:layout_alignLeft="@+id/formulaireCheckBox1"/>
167.
168. <EditText
169.     android:id="@+id/formulaireEditText1"
170.     android:layout_width="wrap_content"
171.     android:layout_height="wrap_content"
172.     android:layout_alignBaseline="@+id/textViewFormulaireEdtText"
173.     android:layout_alignLeft="@+id/formulaireCheckBox1"
174.     android:ems="10"
175.     android:inputType="text">
176. </EditText>
177.
178. <Switch
179.     android:id="@+id/formulaireSwitch1"
180.     android:layout_width="wrap_content"
181.     android:layout_height="wrap_content"
182.     android:layout_alignBaseline="@+id/textViewFormulaireBool"
183.     android:layout_alignLeft="@+id/formulaireCheckBox1"
184.     android:text="@string/formulaire_switch"
185.     android:textOff="Non"
186.     android:textOn="Oui"/>
187.
188. <TimePicker
189.     android:id="@+id/formulaireTimePicker1"
190.     android:layout_width="wrap_content"
191.     android:layout_height="wrap_content"
192.     android:layout_alignBottom="@+id/textViewFormulaireTime"
193.     android:layout_alignLeft="@+id/formulaireEditTextMultiLignes"
194.     android:timePickerMode="spinner"
195. />
196.
197. <EditText
198.     android:id="@+id/formulaireEditTextMultiLignes"
199.     android:layout_width="wrap_content"
200.     android:layout_height="100dp"
201.     android:layout_alignBaseline="@+id/textViewFormulaireMultilignes"
202.     android:layout_alignBottom="@+id/textViewFormulaireMultilignes"
203.     android:layout_marginLeft="50dp"
204.     android:layout_toRightOf="@+id/textViewFormulaireMultilignes"
205.     android:ems="10"
206.     android:inputType="textMultiLine">
207. </EditText>
208.
209. <Spinner
210.     android:id="@+id/formulaireDropDownList"
211.     android:layout_width="200dp"
212.     android:layout_height="50dp"
213.     android:layout_alignBottom="@+id/TextViewFormulaireCombo"

```

```

214.     android:layout_alignLeft="@+id/formulaireEditTextMultiLignes">
215.   </Spinner>
216.
217.   <DatePicker
218.     android:id="@+id/formulaireDatePicker1"
219.     android:layout_width="wrap_content"
220.     android:layout_height="wrap_content"
221.     android:layout_alignBottom="@+id/textViewFormulaireDate"
222.     android:layout_alignLeft="@+id/formulaireCheckBox1"
223.     android:datePickerMode="spinner"
224.     android:calendarViewShown="false">
225.   </DatePicker>
226.
227.   <TextView
228.     android:id="@+id/textViewSeekBarValue"
229.     android:layout_width="30dp"
230.     android:layout_height="wrap_content"
231.     android:layout_alignBaseline="@+id/textViewFormulaireSeekBar"
232.     android:layout_marginLeft="30dp"
233.     android:layout_toRightOf="@+id/formulaireSeekBar"
234.     android:text=""/>
235.   </RelativeLayout>

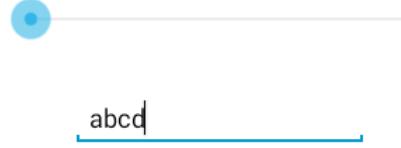
```

Les principaux composants du formulaire sont les suivants :

- ligne 2 : un layout [ScrollView] vertical. Il permet de présenter un formulaire plus grand que l'écran de la tablette. On obtient la totalité du formulaire par défilement ;
- lignes 125-132 : une case à cocher
- lignes 134-159 : un groupe de trois boutons radio
- lignes 161-166 : une barre de recherche
- lignes 16-176 : une boîte de saisie
- lignes 178-186 : un switch oui / non
- lignes 188-195 : une boîte de saisie de l'heure

1

1 2 3



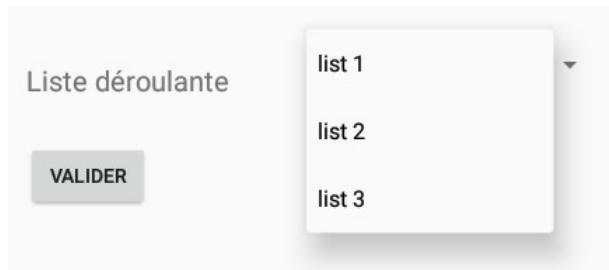
abcd

Oui

9	18		
10	:	19	AM
11	20	PM	

ligne1
ligne2

- lignes 209-215 : une liste déroulante



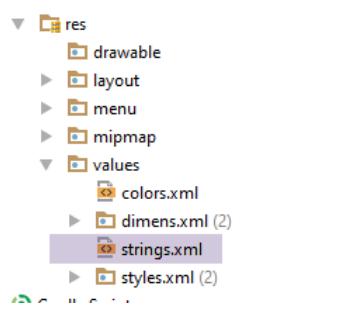
- lignes 217-225 : une boîte de saisie d'une date



- tous les autres composants sont des [TextView] qui affichent des textes.

1.18.3 Les chaînes de caractères du formulaire

Les chaînes de caractères du formulaire sont définies dans le fichier [res / values / strings.xml] suivant :

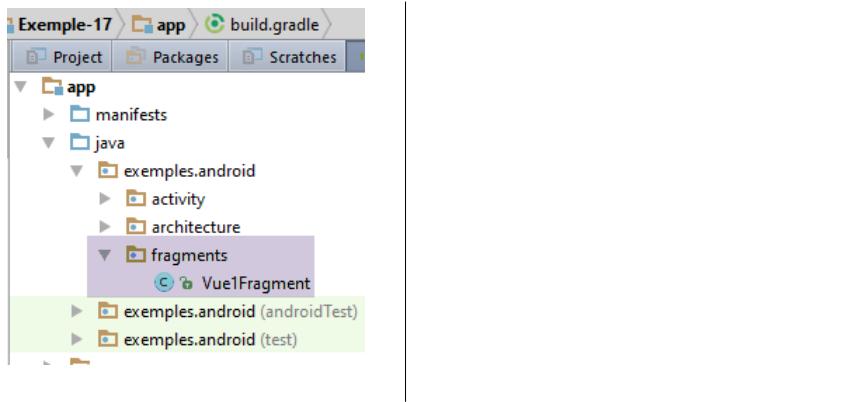


```

1. <resources>
2.   <string name="app_name">Exemple-17</string>
3.   <string name="action_settings">Settings</string>
4.   <string name="section_format">Hello World from section: %1$d</string>
5.   <!-- vue 1 -->
6.   <string name="titre_vue1">Vue n° 1</string>
7.   <string name="formulaire_checkbox">Cases à cocher</string>
8.   <string name="formulaire_radioButton">Boutons Radio</string>
9.   <string name="formulaire_seekBar">Seek Bar</string>
10.  <string name="formulaire_saisie">Champ de saisie</string>
11.  <string name="formulaire_bool">Booléen</string>
12.  <string name="formulaire_date">Date</string>
13.  <string name="formulaire_time">Heure</string>
14.  <string name="formulaire_multilignes">Champ de saisie multilignes</string>
15.  <string name="formulaire_listview">Liste</string>
16.  <string name="formulaire_combo">Liste déroulante</string>
17.  <string name="formulaire_checkbox1">1</string>
18.  <string name="formulaire_checkbox2">2</string>
19.  <string name="formulaire_radiobutton1">1</string>
20.  <string name="formulaire_radionbutton2">2</string>
21.  <string name="formulaire_radiobutton3">3</string>
22.  <string name="formulaire_switch"></string>
23.  <string name="formulaire_valider">Valider</string>
24. </resources>

```

1.18.4 Le fragment du formulaire



La classe [Vue1Fragment] est la suivante :

```

1. package exemples.android.fragments;
2.
3. import android.annotation.SuppressLint;
4. import android.app.AlertDialog;
5. import android.widget.*;
6. import android.widget.SeekBar.OnSeekBarChangeListener;
7. import exemples.android.R;
8. import exemples.android.architecture.AbstractFragment;
9. import org.androidannotations.annotations.AfterViews;
10. import org.androidannotations.annotations.Click;
11. import org.androidannotations.annotations.EFragment;
12. import org.androidannotations.annotations.ViewById;
13.
14. import java.util.ArrayList;
15. import java.util.List;
16.
17. // un fragment est une vue affichée par un conteneur de fragments
18. @EFragment(R.layout.vue1)
19. public class Vue1Fragment extends AbstractFragment {
20.
21.     // les champs de la vue affichée par le fragment
22.     @ViewById(R.id.formulaireDropDownList)
23.     Spinner dropDownList;
24.     @ViewById(R.id.formulaireButtonValider)
25.     Button buttonValider;
26.     @ViewById(R.id.formulaireCheckBox1)
27.     CheckBox checkBox1;
28.     @ViewById(R.id.formulaireRadioGroup)
29.     RadioGroup radioGroup;
30.     @ViewById(R.id.formulaireSeekBar)
31.     SeekBar seekBar;
32.     @ViewById(R.id.formulaireEditText1)
33.     EditText saisie;
34.     @ViewById(R.id.formulaireSwitch1)
35.     Switch switch1;
36.     @ViewById(R.id.formulaireDatePicker1)
37.     DatePicker datePicker1;
38.     @ViewById(R.id.formulaireTimePicker1)
39.     TimePicker timePicker1;
40.     @ViewById(R.id.formulaireEditTextMultiLignes)
41.     EditText multiLignes;
42.     @ViewById(R.id.formulaireRadioButton1)
43.     RadioButton radioButton1;
44.     @ViewById(R.id.formulaireRadioButton2)
45.     RadioButton radioButton2;
46.     @ViewById(R.id.formulaireRadionButton3)
47.     RadioButton radioButton3;
48.     @ViewById(R.id.textViewSeekBarValue)
49.     TextView seekBarValue;
50.
51.     // liste déroulante
52.     private List<String> list;
53.     private ArrayAdapter<String> dataAdapter;
54.
55.     @AfterViews
56.     void afterViews() {
57.         // on coche le premier bouton
58.         radioButton1.setChecked(true);
59.         // le calendrier
60.         datePicker1.setCalendarViewShown(false);
61.         // le seekBar
62.         seekBar.setMax(100);
63.         seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {

```

```

64.
65.     public void onStopTrackingTouch(SeekBar seekBar) {
66.         }
67.
68.     public void onStartTrackingTouch(SeekBar seekBar) {
69.         }
70.
71.     public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
72.         seekBarValue.setText(String.valueOf(progress));
73.     }
74. };
75. // la liste déroulante
76. list = new ArrayList<>();
77. list.add("list 1");
78. list.add("list 2");
79. list.add("list 3");
80. }
81.
82.
83. @SuppressLint("DefaultLocale")
84. @Click(R.id.formulaireButtonValider)
85. protected void doValider() {
86.     ...
87. }
88. @Override
89. protected void updateFragment() {
90.     // initialisation adaptateur de la liste déroulante
91.     dataAdapter = new ArrayAdapter<>(activity, android.R.layout.simple_spinner_item, list);
92.     dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
93.     dropDownList.setAdapter(dataAdapter);
94. }
95. }
```

- lignes 22-49 : on récupère les références de tous les composants du formulaire XML [vue1] (ligne 18) ;
- ligne 58 : la méthode [setChecked] permet de cocher un bouton radio ou une case à cocher ;
- ligne 60 : par défaut le composant [DatePicker] affiche et une boîte de saisie de la date et un calendrier. La ligne 60 élimine le calendrier ;
- ligne 62 : [SeekBar].setMax() permet de fixer la valeur maximale de la barre de réglage. La valeur minimale est 0 ;
- lignes 63-74 : on gère les événements de la barre de réglage. On veut, à chaque changement opéré par l'utilisateur, afficher la valeur de la règle dans le [TextView] de la ligne 49 ;
- ligne 71 : le paramètre [progress] représente la valeur de la règle ;
- lignes 76-79 : une liste de [String] qu'on va associer à la liste déroulante ;
- ligne 89 : la méthode [updateFragment] du fragment. Lorsqu'elle est exécutée, la variable [activity] de la classe parent a été initialisée ;
- ligne 91 : la source de données [list] est associée à l'adaptateur de la liste déroulante ;
- lignes 92-93 : l'adaptateur [dataAdapter] est associé à la liste déroulante [dropDownList] ;
- ligne 84 : on associe la méthode [doValider] au clic sur le bouton [Valider] ;

La méthode [doValider] a pour but d'afficher les valeurs saisies par l'utilisateur. Son code est le suivant :

```

1.     @Click(R.id.formulaireButtonValider)
2.     protected void doValider() {
3.         // liste des messages à afficher
4.         List<String> messages = new ArrayList<>();
5.         // case à cocher
6.         boolean isChecked = checkBox1.isChecked();
7.         messages.add(String.format("CheckBox1 [checked=%s]", isChecked));
8.         // les boutons radio
9.         int id = radioGroup.getCheckedRadioButtonId();
10.        String radioGroupText = id == -1 ? "" : ((RadioButton) activity.findViewById(id)).getText().toString();
11.        messages.add(String.format("RadioGroup [checked=%s]", radioGroupText));
12.        // le SeekBar
13.        int progress = seekBar.getProgress();
14.        messages.add(String.format("SeekBar [value=%d]", progress));
15.        // le champ de saisie
16.        String texte = String.valueOf(saisie.getText());
17.        messages.add(String.format("Saisie simple [value=%s]", texte));
18.        // le switch
19.        boolean état = switch1.isChecked();
20.        messages.add(String.format("Switch [value=%s]", état));
21.        // la date
22.        int an = datePicker1.getYear();
23.        int mois = datePicker1.getMonth() + 1;
24.        int jour = datePicker1.getDayOfMonth();
25.        messages.add(String.format("Date [%d, %d, %d]", jour, mois, an));
26.        // le texte multi-lignes
27.        String lignes = String.valueOf(multiLignes.getText());
28.        messages.add(String.format("Saisie multi-lignes [value=%s]", lignes));
29.        // l'heure
30.        int heure = timePicker1.getHour();
```

```

31.     int minutes = timePicker1.getMinute();
32.     messages.add(String.format("Heure [%d, %d]", heure, minutes));
33.     // liste déroulante
34.     int position = dropDownList.getSelectedItemPosition();
35.     String selectedItem = String.valueOf(dropDownList.getSelectedItem());
36.     messages.add(String.format("DropDownList [position=%d, item=%s]", position, selectedItem));
37.     // affichage
38.     doAfficher(messages);
39. }

```

- ligne 4 : les valeurs saisies vont être cumulées dans une liste de messages ;
- ligne 6 : la méthode `[CheckBox].isCkecked()` permet de savoir si une case est cochée ou non ;
- ligne 9 : la méthode `[RadioGroup].getCheckedButtonId()` permet d'obtenir l'id du bouton radio qui a été coché ou -1 si aucun n'a été coché ;
- ligne 10 : le code `[activity.findViewById(id)]` permet de retrouver le bouton radio coché et d'avoir ainsi son libellé ;
- ligne 13 : la méthode `[SeekBar].getProgress()` permet d'avoir la valeur d'une barre de réglage ;
- ligne 19 : la méthode `[Switch].isChecked()` permet de savoir si un switch est *On* (true) ou *Off* (false) ;
- ligne 22 : la méthode `[DatePicker].getYear()` permet d'avoir l'année choisie avec un objet `[DatePicker]` ;
- ligne 23 : la méthode `[DatePicker].getMonth()` permet d'avoir le mois choisi avec un objet `[DatePicker]` dans l'intervalle `[0,11]` ;
- ligne 24 : la méthode `[DatePicker].getDayOfMonth()` permet d'avoir le jour du mois choisi avec un objet `[DatePicker]` dans l'intervalle `[1,31]` ;
- ligne 30 : la méthode `[TimePicker].getHour()` permet d'avoir l'heure choisie avec un objet `[TimePicker]` ;
- ligne 31 : la méthode `[TimePicker].getMinute()` permet d'avoir les minutes choisies avec un objet `[TimePicker]` ;
- ligne 34 : la méthode `[Spinner].getSelectedItemPosition()` permet d'avoir la position de l'élément sélectionné dans une liste déroulante ;
- ligne 35 : la méthode `[Spinner].getSelectedItem()` permet d'avoir l'objet sélectionné dans une liste déroulante ;

La méthode `[doAfficher]` qui affiche la liste des valeurs saisies est la suivante :

```

1.  private void doAfficher(List<String> messages) {
2.      // on construit le texte à afficher
3.      StringBuilder texte = new StringBuilder();
4.      for (String message : messages) {
5.          texte.append(String.format("%s\n", message));
6.      }
7.      // on l'affiche
8.      new AlertDialog.Builder(activité).setTitle("Valeurs saisies").setMessage(texte).setNeutralButton("Fermer",
9.          null).show();
}

```

- ligne 1 : la méthode reçoit une liste de messages à afficher ;
- lignes 3-6 : un objet `[StringBuilder]` est construit à partir de ces messages. Pour concaténer des chaînes, le type `[StringBuilder]` est plus efficace que le type `[String]` ;
- ligne 8 : une boîte de dialogue affiche le texte de la ligne 3 :

Valeurs saisies

CheckBox1 [checked=true]
 RadioGroup [checked=1]
 SeekBar [value=62]
 Saisie simple [value=]
 Switch [value=true]
 Date [7, 6, 2016]
 Saisie multi-lignes [value=ligne1
 ligne2]
 Heure [12, 40]
 DropDownList [position=2, item=list 3]

FERMER

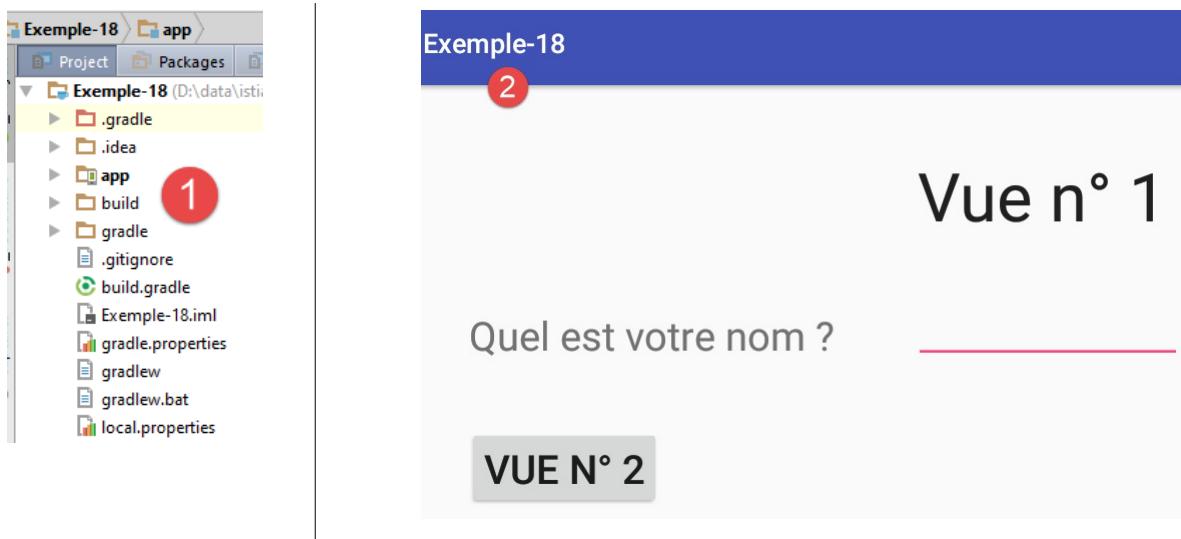
1.18.5 Exécution du projet

Exécutez le projet et testez les différents composants de saisie.

1.19 Exemple-18 : utilisation d'un patron de vues

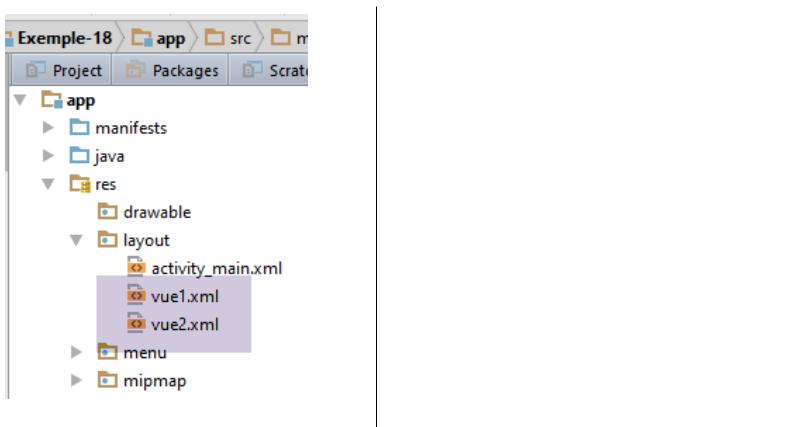
1.19.1 Crédation du projet

Nous créons un nouveau projet [Exemple-18] par recopie du projet [Exemple-13].

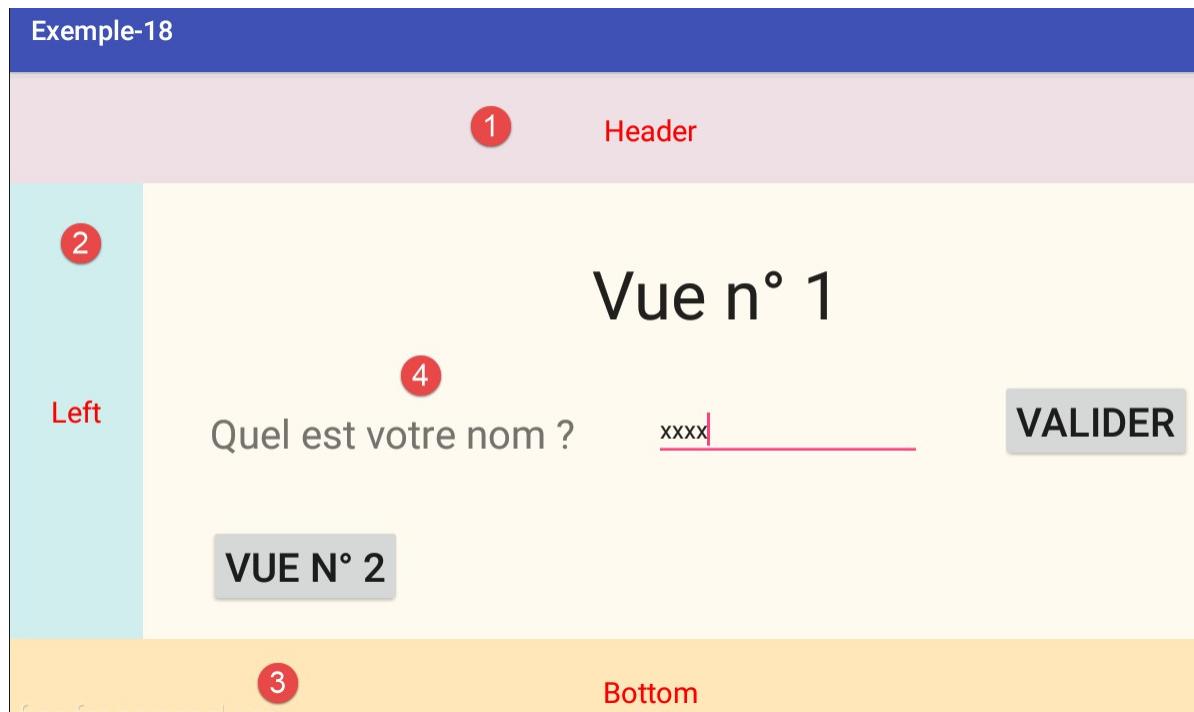


1.19.2 Le patron des vues

Nous voulons reprendre les deux vues du projet et les inclure dans un patron :



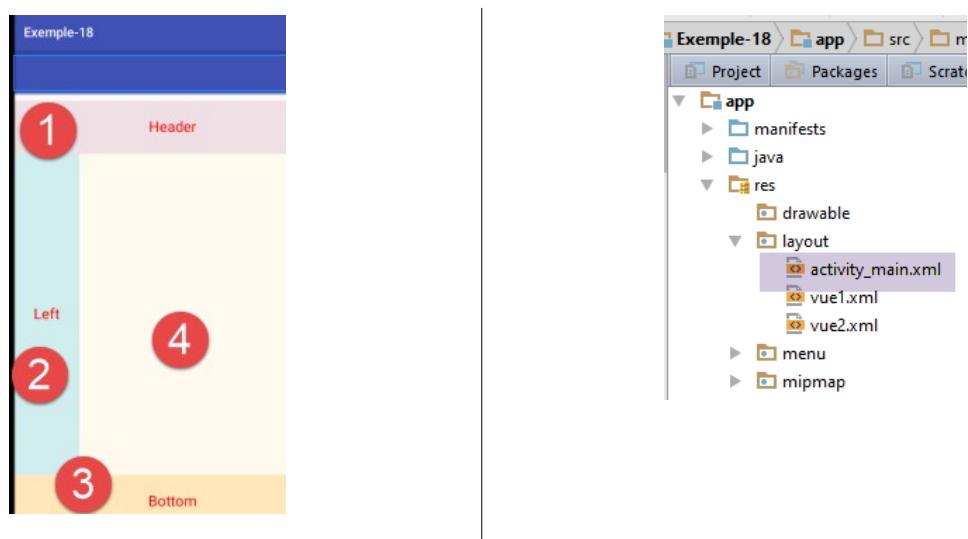
Exemple-18



Chacune des deux vues sera structurée de la même façon :

- en [1], un entête ;
- en [2], une colonne de gauche qui pourrait contenir des liens ;
- en [3], un bas de page ;
- en [4], un contenu.

Ceci est obtenu en modifiant la vue de base [activity_main.xml] de l'activité ;



Le code XML de la vue [main] est le suivant :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.                                         xmlns:tools="http://schemas.android.com/tools"
4.                                         xmlns:app="http://schemas.android.com/apk/res-auto"
5.                                         android:id="@+id/main_content"
6.                                         android:layout_width="match_parent"
7.                                         android:layout_height="match_parent"
8.                                         android:fitsSystemWindows="true"
9.                                         tools:context=".activity.MainActivity">
```

```

10.
11. <android.support.design.widget.AppBarLayout
12.     android:id="@+id/appbar"
13.     android:layout_width="match_parent"
14.     android:layout_height="wrap_content"
15.     android:paddingTop="@dimen/appbar_padding_top"
16.     android:theme="@style/AppTheme.AppBarOverlay">
17.
18.     <android.support.v7.widget.Toolbar
19.         android:id="@+id/toolbar"
20.         android:layout_width="match_parent"
21.         android:layout_height="?attr/actionBarSize"
22.         android:background="?attr/colorPrimary"
23.         app:popupTheme="@style/AppTheme.PopupOverlay"
24.         app:layout_scrollFlags="scroll|enterAlways">
25.
26.     </android.support.v7.widget.Toolbar>
27.
28. </android.support.design.widget.AppBarLayout>
29.
30. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
31.                 xmlns:tools="http://schemas.android.com/tools"
32.                 android:layout_width="match_parent"
33.                 android:layout_height="match_parent"
34.                 android:gravity="center"
35.                 android:layout_marginTop="75dp"
36.                 android:orientation="vertical">
37.
38.     <LinearLayout
39.         android:id="@+id/header"
40.         android:layout_width="match_parent"
41.         android:layout_height="100dp"
42.         android:layout_weight="0.1"
43.         android:background="@color/lavenderblushh2">
44.
45.         <TextView
46.             android:id="@+id/textViewHeader"
47.             android:layout_width="match_parent"
48.             android:layout_height="wrap_content"
49.             android:layout_gravity="center"
50.             android:gravity="center_horizontal"
51.             android:text="@string/txt_header"
52.             android:textAppearance="?android:attr/textAppearanceLarge"
53.             android:textColor="@color/red"/>
54.     </LinearLayout>
55.
56.     <LinearLayout
57.         android:layout_width="match_parent"
58.         android:layout_height="fill_parent"
59.         android:layout_weight="0.8"
60.         android:orientation="horizontal">
61.
62.         <LinearLayout
63.             android:id="@+id/left"
64.             android:layout_width="100dp"
65.             android:layout_height="match_parent"
66.             android:background="@color/lightcyan2">
67.
68.             <TextView
69.                 android:id="@+id/txt_left"
70.                 android:layout_width="fill_parent"
71.                 android:layout_height="fill_parent"
72.                 android:gravity="center_vertical|center_horizontal"
73.                 android:text="@string/txt_left"
74.                 android:textAppearance="?android:attr/textAppearanceLarge"
75.                 android:textColor="@color/red"/>
76.         </LinearLayout>
77.
78.         <examples.android.architecture.MyPagerAdapter
79.             android:id="@+id/container"
80.             android:layout_width="match_parent"
81.             android:layout_height="match_parent"
82.             android:background="@color/floral_white"
83.             app:layout_behavior="@string/appbar_scrolling_view_behavior"/>
84.     </LinearLayout>
85.
86.     <LinearLayout
87.         android:id="@+id/bottom"
88.         android:layout_width="match_parent"
89.         android:layout_height="100dp"
90.         android:layout_weight="0.1"
91.         android:background="@color/wheat1">
92.
93.         <TextView
94.             android:id="@+id/textViewBottom"
95.             android:layout_width="fill_parent"
96.             android:layout_height="fill_parent">

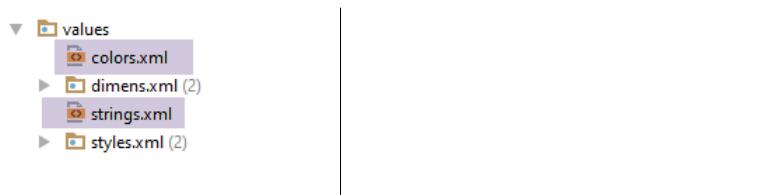
```

```

97.     android:gravity="center_vertical|center_horizontal"
98.     android:text="@string/txt_bottom"
99.     android:textAppearance="?android:attr/textAppearanceLarge"
100.    android:textColor="@color/red"/>
101.   
```

- l'en-tête [1] est obtenu avec les lignes 38-54 ;
- la bande gauche [2] est obtenue avec les lignes 56-84 ;
- le bas de page [3] est obtenu avec les lignes 86-101 ;
- le contenu [4] est obtenu avec les lignes 78-84 ;

La vue XML [main] utilise des informations trouvées dans les fichiers [res / values / colors.xml] et [res / values / strings.xml] :



Le fichier [colors.xml] est le suivant :

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.      <color name="red">#FF0000</color>
5.      <color name="blue">#0000FF</color>
6.      <color name="wheat">#FFEFD5</color>
7.      <color name="floral_white">#FFFAF0</color>
8.      <color name="Lavenderblushh2">#EEE0E5</color>
9.      <color name="lightcyan2">#D1EEEE</color>
10.     <color name="wheat1">#FFE7BA</color>
11.
12. </resources>

```

et le fichier [strings.xml] le suivant :

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.      <string name="app_name">exemple-12</string>
4.      <string name="action_settings">Settings</string>
5.      <string name="titre_vue1">Vue n° 1</string>
6.      <string name="textView_nom">Quel est votre nom :</string>
7.      <string name="btn_Validator">Validez</string>
8.      <string name="btn_vue2">Vue n° 2</string>
9.      <string name="titre_vue2">Vue n° 2</string>
10.     <string name="btn_vue1">Vue n° 1</string>
11.     <string name="textView_bonjour">"Bonjour "</string>
12.     <string name="txt_header">Header</string>
13.     <string name="txt_left">Left</string>
14.     <string name="txt_bottom">Bottom</string>
15.
16. </resources>

```

Créez un contexte d'exécution pour ce projet et exécutez-le.

1.20 Exemple-19 : le composant [ListView]

Le composant [ListView] permet de répéter une vue particulière pour chaque élément d'une liste. La vue répétée peut être d'une complexité quelconque, d'une simple chaîne de caractères à une vue permettant de saisir des informations pour chaque élément de la liste. Nous allons créer le [ListView] suivant :

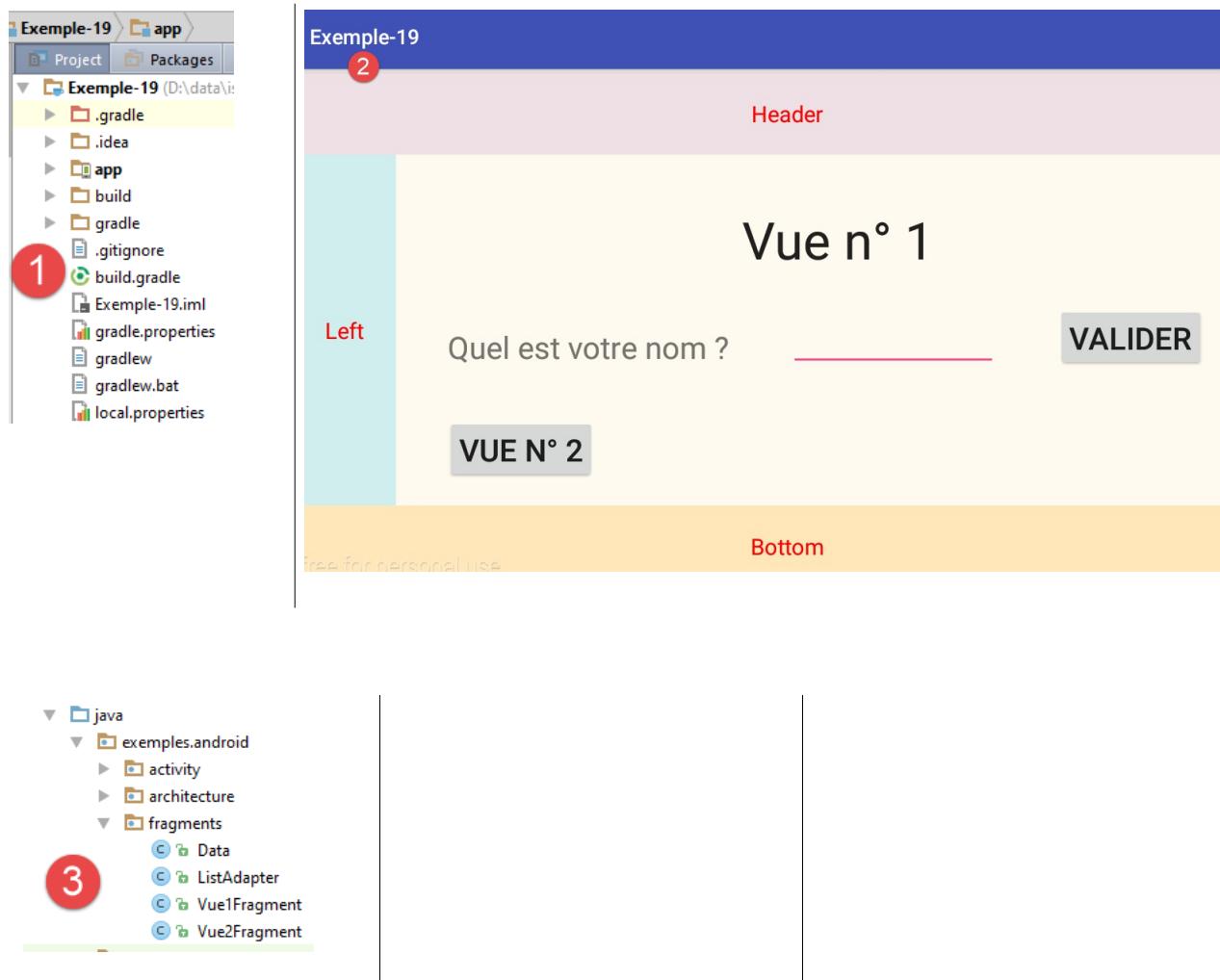
Texte n° 0	<input type="checkbox"/>	Retirer
Texte n° 1	<input type="checkbox"/>	Retirer
Texte n° 2	<input type="checkbox"/>	Retirer
Texte n° 3	<input type="checkbox"/>	Retirer
Texte n° 4	<input type="checkbox"/>	Retirer

Chaque vue de la liste a trois composants :

- un [TextView] d'information ;
- un [CheckBox] ;
- un [TextView] cliquable ;

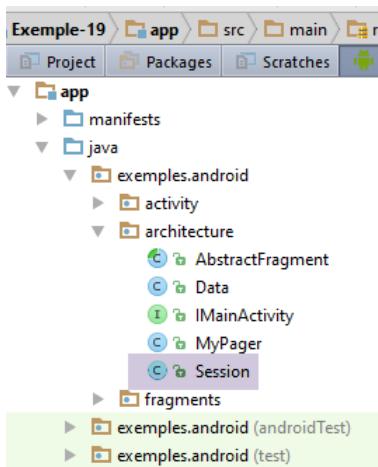
1.20.1 Crédation du projet

Nous créons un nouveau projet [Exemple-19] par recopie du projet [Exemple-18].



Nous allons faire évoluer le projet comme indiqué en [3].

1.20.2 La session



La session mémorise les données partagées entre activité et fragments :

```
1. package exemples.android.architecture;
2.
3. import org.androidannotations.annotations.EBean;
4.
5. import java.util.ArrayList;
6. import java.util.List;
7.
8. @EBean(scope = EBean.Scope.Singleton)
9. public class Session {
10.     // une liste de données
11.     private List<Data> liste=new ArrayList<>();
12.
13.     // getters et setters
14.     ...
15. }
```

- ligne 11 : la liste de données exploitée par les deux vues ;

La classe [Data] est la suivante :

```
1. package exemples.android.architecture;
2.
3. public class Data {
4.
5.     // données
6.     private String texte;
7.     private boolean isChecked;
8.
9.     // constructeur
10.    public Data(String texte, boolean isCkecked) {
11.        this.texte = texte;
12.        this.isChecked = isCkecked;
13.    }
14.
15.    // getters et setters
16.    ...
17. }
```

- ligne 6 : le texte qui va alimenter le premier [TextView] de [list_data] ;
- ligne 7 : le booléen qui va servir à cocher ou non le [checkBox] de [list_data] ;

1.20.3 L'activité [MainActivity]

Le code de la méthode [@AfterInject] devient le suivant :

```
1.     // injection session
2.     @Bean(Session.class)
3.     protected Session session;
4.     ...
5.     @AfterInject
6.     protected void afterInject() {
```

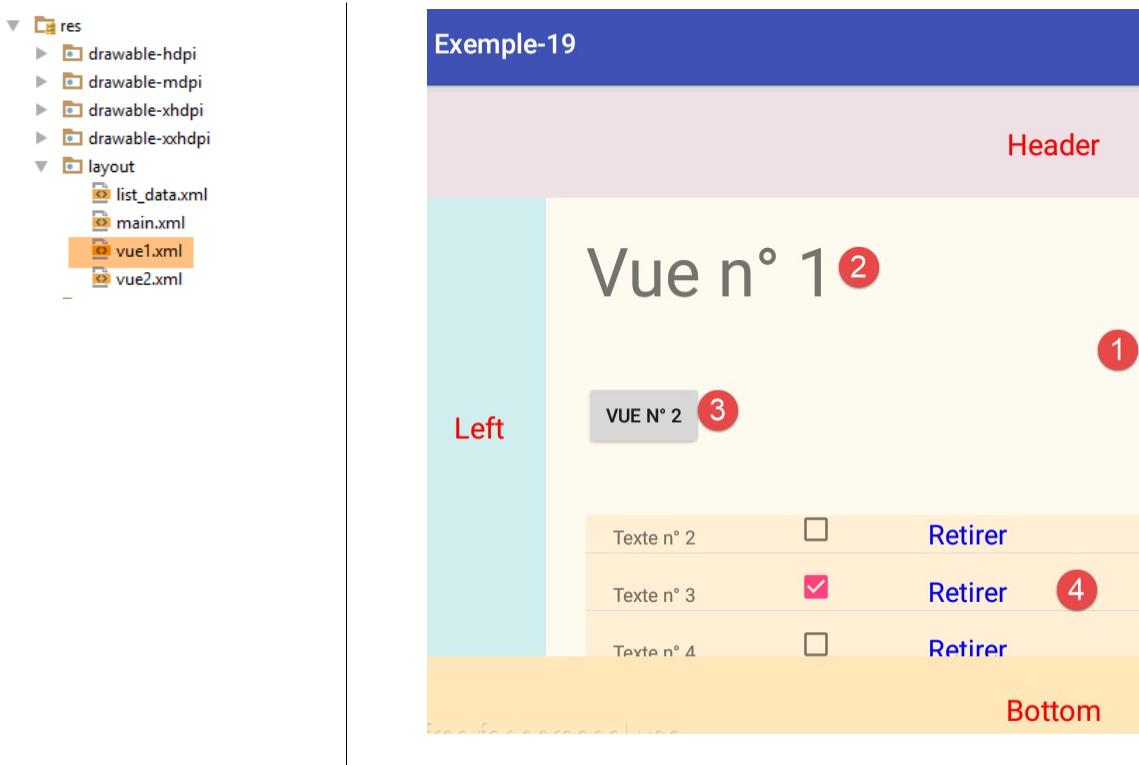
```

7.      // log
8.      if (IS_DEBUG_ENABLED) {
9.          Log.d("MainActivity", "afterInject");
10.     }
11.    // on crée une liste de données
12.    List<Data> liste = session.getListe();
13.    for (int i = 0; i < 20; i++) {
14.        liste.add(new Data("Texte n° " + i, false));
15.    }
16. }

```

- lignes 12-15 : initialisation de la liste des données présente en session ;

1.20.4 La vue [Vue1] initiale



La vue XML [vue1.xml] affiche la zone [1] ci-dessus. Son code est le suivant :

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
4.                  android:layout_width="match_parent"
5.                  android:layout_height="match_parent" >
6.
7.      <TextView
8.          android:id="@+id/textView_titre"
9.          android:layout_width="wrap_content"
10.         android:layout_height="wrap_content"
11.         android:layout_alignParentLeft="true"
12.         android:layout_alignParentTop="true"
13.         android:layout_marginLeft="30dp"
14.         android:layout_marginTop="20dp"
15.         android:text="@string/titre_vue1"
16.         android:textSize="50sp" />
17.
18.      <Button
19.          android:id="@+id/button_vue2"
20.          android:layout_width="wrap_content"
21.          android:layout_height="wrap_content"
22.          android:layout_alignLeft="@+id/textView_titre"
23.          android:layout_below="@+id/textView_titre"
24.          android:layout_marginTop="50dp"
25.          android:text="@string/btn_vue2" />
26.
27.      <ListView
28.          android:id="@+id/listView1"

```

```

29.     android:layout_width="600dp"
30.     android:layout_height="200dp"
31.     android:layout_alignParentLeft="true"
32.     android:layout_below="@+id/button_vue2"
33.     android:layout_marginLeft="30dp"
34.     android:layout_marginTop="50dp" >
35.   </ListView>
36.
37. </RelativeLayout>

```

- lignes 7-16 : le composant [TextView] [2] ;
- lignes 27-35 : le composant [ListView] [4] ;
- lignes 18-25 : le composant [Button] [3] ;

1.20.5 La vue répétée par le [ListView]



La vue répétée par le [ListView] est la vue [list_data] suivante :

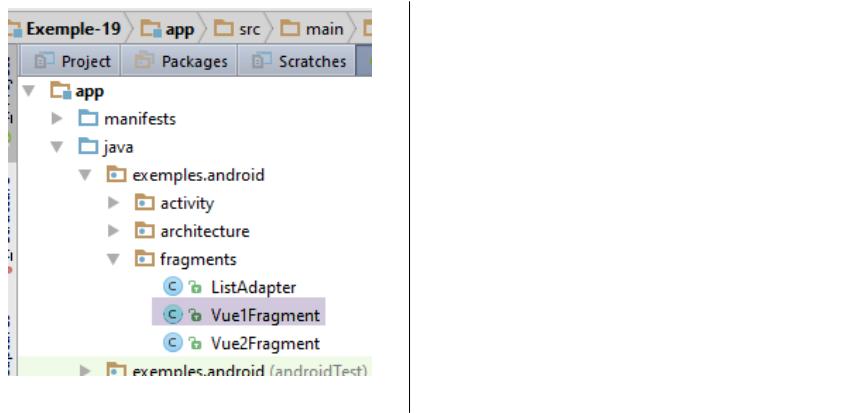
```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.    android:id="@+id/RelativeLayout1"
4.    android:layout_width="match_parent"
5.    android:layout_height="match_parent"
6.    android:background="@color/wheat" >
7.
8.    <TextView
9.      android:id="@+id/txt.Libellé"
10.     android:layout_width="100dp"
11.     android:layout_height="wrap_content"
12.     android:layout_marginLeft="20dp"
13.     android:layout_marginTop="20dp"
14.     android:text="@string/txt_dummy" />
15.
16.    <CheckBox
17.      android:id="@+id/checkBox1"
18.      android:layout_width="wrap_content"
19.      android:layout_height="wrap_content"
20.      android:layout_alignBottom="@+id/txt.Libellé"
21.      android:layout_marginLeft="37dp"
22.      android:layout_toRightOf="@+id/txt.Libellé"
23.      android:text="@string/txt_dummy" />
24.
25.    <TextView
26.      android:id="@+id/textViewRetirer"
27.      android:layout_width="wrap_content"
28.      android:layout_height="wrap_content"
29.      android:layout_alignBaseline="@+id/txt.Libellé"
30.      android:layout_alignBottom="@+id/txt.Libellé"
31.      android:layout_marginLeft="68dp"
32.      android:layout_toRightOf="@+id/checkBox1"
33.      android:text="@string/txt_retirer"
34.      android:textColor="@color/blue"
35.      android:textSize="20sp" />
36.
37.  </RelativeLayout>

```

- lignes 8-14 : le composant [TextView] [1] ;
- lignes 16-23 : le composant [CheckBox] [2] ;
- lignes 25-35 : le composant [TextView] [3] ;

1.20.6 Le fragment [Vue1Fragment]



Le fragment [Vue1Fragment] gère la vue XML [vue1]. Son code est le suivant :

```
1. package exemples.android.fragments;
2.
3. import android.view.View;
4. import android.widget.ListView;
5. import exemples.android.R;
6. import exemples.android.architecture.AbstractFragment;
7. import exemples.android.architecture.Data;
8. import org.androidannotations.annotations.AfterViews;
9. import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.ViewById;
12.
13. import java.util.List;
14.
15. @EFragment(R.layout.vue1)
16. public class Vue1Fragment extends AbstractFragment {
17.
18.     // les champs de la vue affichée par le fragment
19.     @ViewById(R.id.listView1)
20.     protected ListView listView;
21.     // l'adaptateur de liste
22.     private ListAdapter adapter;
23.     // init done
24.     private boolean initDone = false;
25.
26.     @AfterViews
27.     void afterViews() {
28.         // mémoire
29.         afterViewsDone = true;
30.     }
31.
32.     @Click(R.id.button_vue2)
33.     void navigateToView2() {
34.         // on navigue vers la vue 2
35.         MainActivity.navigateView(1);
36.     }
37.
38.     public void doRetirer(int position) {
39.         ...
40.     }
41.
42.     @Override
43.     protected void updateFragment() {
44.         if (!initDone) {
45.             // on associe des données au [ListView]
46.             adapter = new ListAdapter(activity, R.layout.list_data, session.getListe(), this);
47.             initDone = true;
48.         }
49.         // cas où le fragment a été (ré)généré - dans ce cas il faut relier de nouveau le ListView à son adaptateur
50.         listView.setAdapter(adapter);
51.         // cas où d'autres fragments ont changé la source de données - dans ce cas il faut rafraîchir le ListView
52.         adapter.notifyDataSetChanged();
53.     }
54. }
```

- ligne 15 : la vue XML [vue1] est associée au fragment ;
- lignes 26-30 : la méthode [@AfterViews] ne fait rien. Elle est cependant nécessaire pour mettre la variable [afterViewsDone] à *true* car celle-ci est utilisée par la classe parent [Abstractfragment] ;

- lignes 42-53 : la méthode [updateFragment] qui est appelée à chaque fois que le fragment va être visible. La méthode a été écrite ici comme si le fragment pouvait sortir de l'adjacence du fragment affiché et donc réinitialiser son cycle de vie. Ce n'est pas le cas ici, mais ce serait le cas si l'application venait à avoir 3 fragments avec une adjacence de 1 ;
- ligne 44 : l'adaptateur du ListView n'a besoin d'être initialisé qu'une fois ;
- ligne 46 : on associe à ce [ListView] un adaptateur de type [ListAdapter]. Nous allons construire cette classe. Elle dérive de la classe [ArrayAdapter] que nous avons déjà eu l'occasion d'utiliser pour associer des données à un [ListView]. Nous passons diverses informations au constructeur de [ListAdapter] :
 - une référence sur l'activité courante,
 - l'identifiant de la vue qui sera instanciée pour chaque élément de la liste,
 - une source de données pour alimenter la liste,
 - une référence sur le fragment. Celle-ci sera utilisée pour gérer le clic sur un lien [Retirer] du [ListView] par la méthode [doRetirer] de la ligne 38 ;
- ligne 50 : l'adaptateur est associé au ListView. Par la même occasion, la source de données [listes] est associé au ListView. Cette opération sera ici faite à chaque fois que la vue 1 est affichée. Elle n'aurait besoin d'être faite en réalité que lorsque la méthode [@AfterViews] a été exécutée. Ici l'instruction est exécutée trop souvent. On sent le besoin d'un booléen qui nous dirait que la méthode [@AfterViews] vient juste d'être exécutée et que donc le ListView doit être de nouveau associé à son adaptateur ;
- ligne 52 : on rafraîchit le ListView. Dans cet exemple, cela ne sert à rien car seule la vue n° 1 peut modifier la source de données du ListView. On se place dans un cas plus général où la vue n° 2 pourrait elle aussi changer la source de données du ListView. On rencontrera de tels exemples plus loin dans ce document. Dans ce cas, lorsqu'on passe de la vue n° 2 à la vue n° 1, le ListView de la vue n° 1 doit être rafraîchi ;

1.20.7 L'adaptateur [ListAdapter] du [ListView]



La classe [ListAdapter]

- configure la source de données du [ListView] ;
- gère l'affichage des différents éléments du [ListView] ;
- gère les événements de ces éléments ;

Son code est le suivant :

```

1. package exemples.android.fragments;
2.
3. import java.util.List;
4. ...
5. public class ListAdapter extends ArrayAdapter<Data> {
6.
7.     // le contexte d'exécution
8.     private Context context;
9.     // l'id du layout d'affichage d'une ligne de la liste
10.    private int layoutResourceId;
11.    // les données de la liste
12.    private List<Data> data;
13.    // le fragment qui affiche le [ListView]
14.    private Vue1Fragment fragment;
15.    // l'adaptateur
16.    final ListAdapter adapter = this;
17.
18.    // constructeur
19.    public ListAdapter(Context context, int layoutResourceId, List<Data> data, Vue1Fragment fragment) {
20.        super(context, layoutResourceId, data);
21.        // on mémorise les infos
22.        this.context = context;

```

```

23.     this.layoutResourceId = layoutResourceId;
24.     this.data = data;
25.     this.fragment = fragment;
26. }
27.
28. @Override
29. public View getView(final int position, View convertView, ViewGroup parent) {
30. ...
31. }
32. }
```

- ligne 5 : la classe [ListAdapter] étend la classe [ArrayAdapter] ;
- ligne 19 : le constructeur ;
- ligne 20 : ne pas oublier d'appeler le constructeur de la classe parent [ArrayAdapter] avec les trois premiers paramètres ;
- lignes 22-25 : on mémorise les informations du constructeur ;
- ligne 29 : la méthode [getView] va être appelée de façon répétée par le [ListView] pour générer la vue de l'élément n° [position]. Le résultat [View] rendu est une référence sur la vue créée.

Le code de la méthode [getView] est le suivant :

```

1.  @Override
2.  public View getView(final int position, View convertView, ViewGroup parent) {
3.      // on crée la ligne
4.      View row = ((Activity) context).getLayoutInflater().inflate(layoutResourceId, parent, false);
5.      // le texte
6.      TextView textView = (TextView) row.findViewById(R.id.txt.Libellé);
7.      textView.setText(data.get(position).getTexte());
8.      // la case à cocher
9.      CheckBox checkBox = (CheckBox) row.findViewById(R.id.checkBox1);
10.     checkBox.setChecked(data.get(position).isChecked());
11.     // le lien [Retirer]
12.     TextView txtRetirer = (TextView) row.findViewById(R.id.textViewRetirer);
13.     txtRetirer.setOnClickListener(new OnClickListener() {
14.
15.         public void onClick(View v) {
16.             fragment.doRetirer(position);
17.         }
18.     });
19.     // on gère le clic sur la case à cocher
20.     checkBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
21.
22.         public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
23.             data.get(position).setChecked(isChecked);
24.         }
25.     });
26.     // on rend la ligne
27.     return row;
28. }
```

- ligne 2 : la méthode reçoit trois paramètres. Nous n'allons utiliser que le premier ;
- ligne 4 : on crée la vue de l'élément n° [position]. C'est la vue [list_data] dont l'id a été passé comme deuxième paramètre au constructeur. Ensuite on procède comme d'habitude. On récupère les références des composants de la vue qu'on vient d'instancier ;
- ligne 6 : on récupère la référence du [TextView] n° 1 ;
- ligne 7 : on lui assigne un texte provenant de la source de données qui a été passée comme troisième paramètre au constructeur ;
- ligne 9 : on récupère la référence du [CheckBox] n° 2 ;
- ligne 10 : on le coche ou non avec une valeur provenant de la source de données du [ListView] ;
- ligne 12 : on récupère la référence du [TextView] n° 3 ;
- lignes 13-18 : on gère le clic sur le lien [Retirer] ;
- ligne 16 : c'est la méthode [Vue1Fragment]. doRetirer qui va gérer ce clic. Il paraît en effet plus logique de faire gérer cet événement par le fragment qui affiche le [ListView]. Il a une vue d'ensemble que n'a pas la classe [ListAdapter]. La référence du fragment [Vue1Fragment] avait été passée comme quatrième paramètre au constructeur de la classe ;
- lignes 20-25 : on gère le clic sur la case à cocher. L'action faite sur elle est répercutée sur la donnée qu'elle affiche. Ceci pour la raison suivante. Le [ListView] est une liste qui n'affiche qu'une partie de ces éléments. Ainsi un élément de la liste est-il parfois caché, parfois affiché. Lorsque l'élément n° i doit être affiché, la méthode [getView] de la ligne 2 ci-dessus est appellée pour la position n° i. La ligne 10 va recalculer l'état de la case à cocher à partir de la donnée à laquelle elle est liée. Il faut donc que celle-ci mémorise l'état de la case à cocher au fil du temps ;

1.20.8 Retirer un élément de la liste

Le clic sur le lien [Retirer] est géré dans le fragment [Vue1Fragment] par la méthode [doRetirer] suivante :

```
1.  public void doRetirer(int position) {
```

```

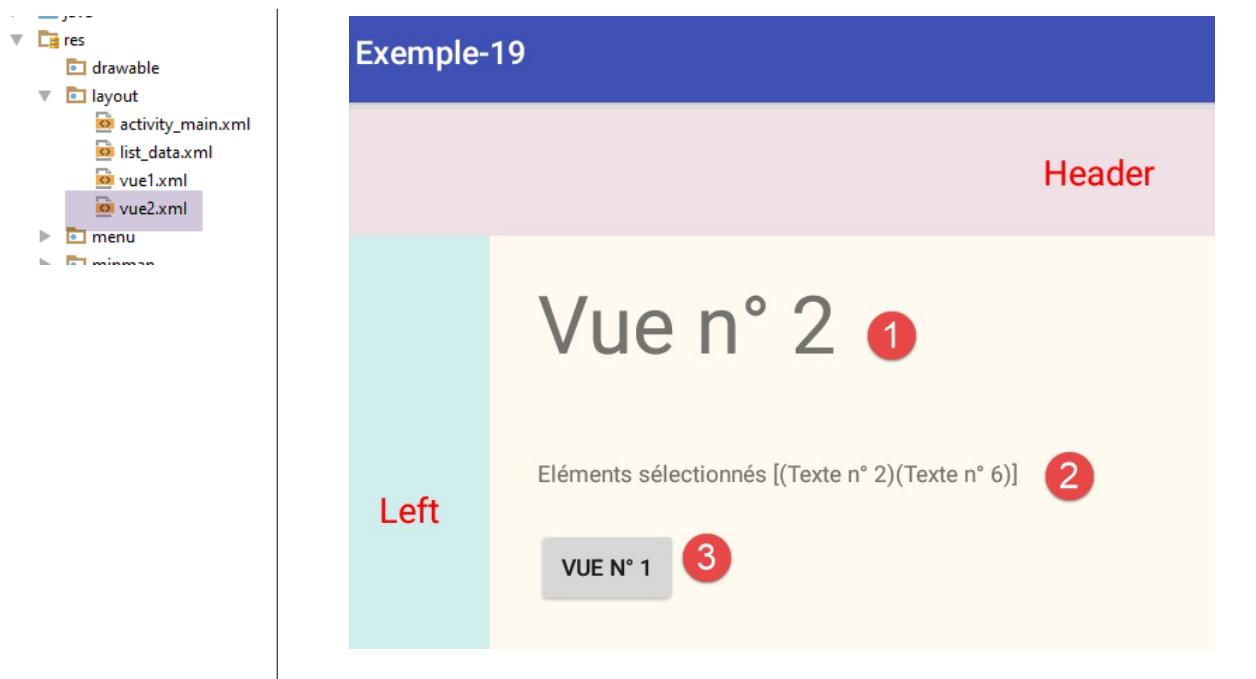
2.     // on enlève l'élément n° [position] dans la liste
3.     List<Data> liste = mainActivity.getListe();
4.     liste.remove(position);
5.     // on note la position du scroll pour y revenir
6.     // lire
7.     // [http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-returning-to-a-listview]
8.     // position du 1er élément visible complètement ou non
9.     int firstPosition = listView.getFirstVisiblePosition();
10.    // offset Y de cet élément par rapport au haut du ListView
11.    // mesure la hauteur de la partie éventuellement cachée
12.    View v = listView.getChildAt(0);
13.    int top = (v == null) ? 0 : v.getTop();
14.    // on rafraîchit le [ListView]
15.    adapter.notifyDataSetChanged();
16.    // on se positionne au bon endroit du ListView
17.    listView.setSelectionFromTop(firstPosition, top);
18. }

```

- ligne 1 : on reçoit la position dans le [ListView] du lien [Retirer] qui a été cliqué ;
- ligne 3 : on récupère la liste de données ;
- ligne 4 : on retire l'élément de n° [position] ;
- ligne 15 : on rafraîchit le [ListView]. Sans cela, visuellement rien ne change.
- lignes 5-13, 17 : une gymnastique assez complexe. Sans elle, il se passe la chose suivante :
 - le [ListView] affiche les lignes 15-18 de la liste de données,
 - on supprime la ligne 16,
 - la ligne 15 ci-dessus le réinitialise totalement et le [ListView] affiche alors les lignes 0-3 de la liste de données ;

Avec les lignes ci-dessus, la suppression se fait et le [ListView] reste positionné sur la ligne qui suit la ligne supprimée.

1.20.9 La vue XML [Vue2]



Le code XML de la vue est le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent" >
5.
6.     <TextView
7.         android:id="@+id/textView_titre"
8.         android:layout_width="wrap_content"
9.         android:layout_height="wrap_content"
10.        android:layout_alignParentLeft="true"
11.        android:layout_alignParentTop="true"
12.        android:layout_marginLeft="30dp"
13.        android:layout_marginTop="20dp" 

```

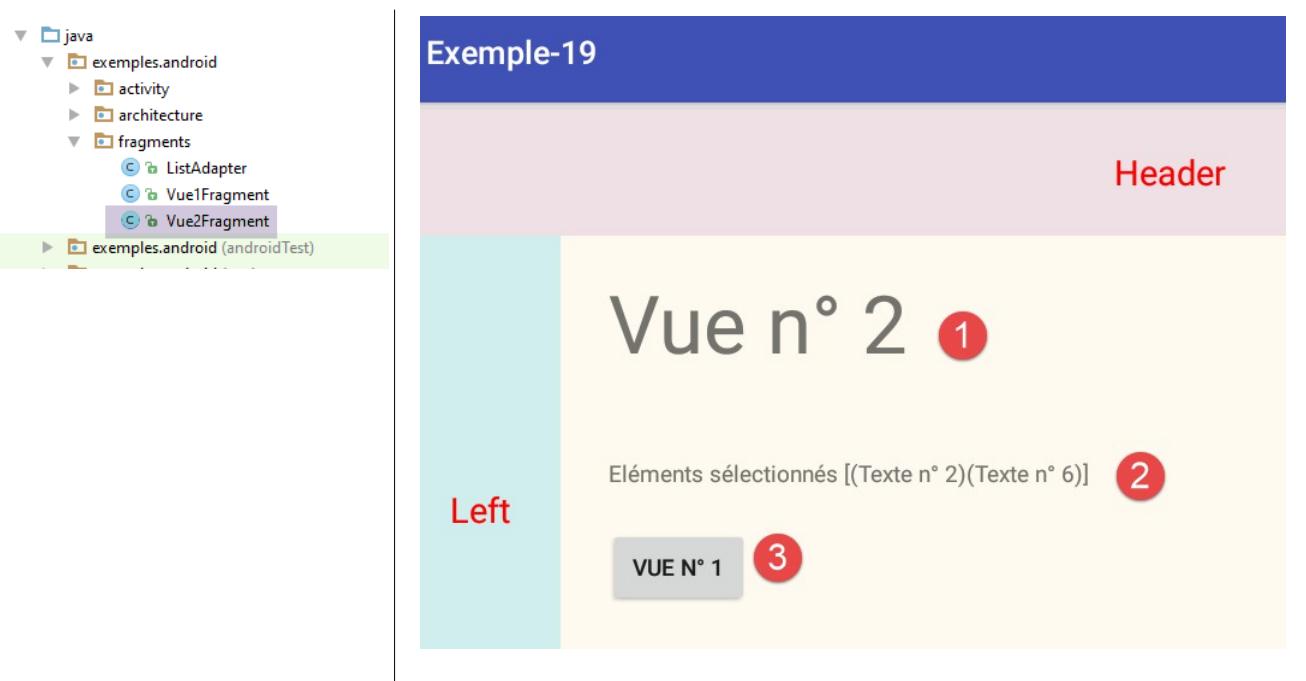
```

14.        android:text="@string/titre_vue2"
15.        android:textSize="50sp" />
16.
17.    <Button
18.        android:id="@+id/button_vue1"
19.        android:layout_width="wrap_content"
20.        android:layout_height="wrap_content"
21.        android:layout_below="@+id/textViewResultats"
22.        android:layout_marginTop="25dp"
23.        android:layout_alignLeft="@+id/textView_titre"
24.        android:text="@string/btn_vue1" />
25.
26.    <TextView
27.        android:id="@+id/textViewResultats"
28.        android:layout_width="wrap_content"
29.        android:layout_height="wrap_content"
30.        android:layout_below="@+id/textView_titre"
31.        android:layout_marginTop="50dp"
32.        android:layout_alignLeft="@+id/textView_titre"
33.        android:text="" />
34.
35. </RelativeLayout>

```

- lignes 6-15 : le composant [TextView] n° 1 ;
- lignes 26-33 : le composant [TextView] n° 2 ;
- lignes 17-24 : le composant [Button] n° 3 ;

1.20.10 Le fragment [Vue2Fragment]



Le fragment [Vue2Fragment] gère la vue XML [vue2]. Son code est le suivant :

```

1. package exemplles.android.fragments;
2.
3. import android.widget.TextView;
4. import exemplles.android.R;
5. import exemplles.android.architecture.AbstractFragment;
6. import exemplles.android.architecture.Data;
7. import org.androidannotations.annotations.AfterViews;
8. import org.androidannotations.annotations.Click;
9. import org.androidannotations.annotations.EFragment;
10. import org.androidannotations.annotations.ViewById;
11.
12. @EFragment(R.layout.vue2)
13. public class Vue2Fragment extends AbstractFragment {
14.
15.     // les champs de la vue
16.     @ViewById(R.id.textViewResultats)
17.     TextView txtResultats;

```

```

18.
19.     @AfterViews
20.     void initFragment(){
21.         // mémoire
22.         afterViewsDone=true;
23.     }
24.
25.     @Click(R.id.button_vue1)
26.     void navigateToView1() {
27.         // on navigue vers la vue 1
28.         mainActivity.navigateToView(0);
29.     }
30.
31.     @Override
32.     protected void updateFragment() {
33.         // on affiche les éléments de la liste qui ont été sélectionnés dans la vue 1
34.         StringBuider texte = new StringBuider("Eléments sélectionnés [");
35.         for (Data data : mainActivity.getListe()) {
36.             if (data.isChecked()) {
37.                 texte.append(String.format("(%s)", data.getTexte()));
38.             }
39.         }
40.         texte.append("]");
41.         txtResultats.setText(texte);
42.     }
43. }

```

Le code important est dans la méthode [updateFragment] de la ligne 32 :

- ligne 34 : on calcule le texte à afficher dans le [TextView] n° 2 ;
- lignes 35-39 : on parcourt la liste des données affichée par le [ListView]. Elle est stockée dans l'activité ;
- ligne 36 : si la donnée n° i a été cochée, on ajoute le libellé associé dans un type [StringBuider] ;
- ligne 41 : le [TextView] affiche le texte calculé ;

1.20.11 Exécution

Créez une configuration d'exécution pour ce projet et exécutez-la.

1.20.12 Amélioration

Dans l'exemple précédent nous avons utilisé une source de données List<Data> où la classe [Data] était la suivante :

```

1. package exemples.android.fragments;
2.
3. public class Data {
4.
5.     // données
6.     private String texte;
7.     private boolean isChecked;
8.
9.     // constructeur
10.    public Data(String texte, boolean isCkecked) {
11.        this.texte = texte;
12.        this.isChecked = isCkecked;
13.    }
14. ...
15. }
16. }

```

Ligne 7, on avait utilisé un booléen pour gérer la case à cocher des éléments du [ListView]. Souvent, le [ListView] doit afficher des données qu'on peut sélectionner en cochant une case sans que pour autant l'élément de la source de données ait un champ booléen correspondant à cette case. On peut alors procéder de la façon suivante :

La classe [Data] devient la suivante :

```

1. package exemples.android.fragments;
2.
3. public class Data {
4.
5.     // données
6.     private String texte;
7.
8.     // constructeur
9.     public Data(String texte) {
10.         this.texte = texte;
11.     }
12.
13.     // getters et setters
14. ...

```

```
15. }
```

On crée une classe [CheckedData] dérivée de la précédente :

```
1. package exemples.android.fragments;
2.
3. public class CheckedData extends Data {
4.
5.     // élément coché
6.     private boolean isChecked;
7.
8.     // constructeur
9.     public CheckedData(String text, boolean isChecked) {
10.         // parent
11.         super(text);
12.         // local
13.         this.isChecked = isChecked;
14.     }
15.
16.     // getters et setters
17. ...
18. }
```

Il suffit ensuite de remplacer partout dans le code (MainActivity, ListAdapter, Vue1Fragment, Vue2Fragment), le type [Data] par le type [CheckedData]. Par exemple dans [MainActivity] :

```
1. @AfterInject
2. protected void afterInject() {
3.     // log
4.     if (IS_DEBUG_ENABLED) {
5.         Log.d("MainActivity", "afterInject");
6.     }
7.     // on crée une liste de données
8.     List<CheckedData> liste = session.getListe();
9.     for (int i = 0; i < 20; i++) {
10.         liste.add(new CheckedData("Texte n° " + i, false));
11.     }
12. }
```

Le projet de cette version vous est fourni sous le nom [Exemple-19B].

1.21 Exemple-20 : utiliser un menu

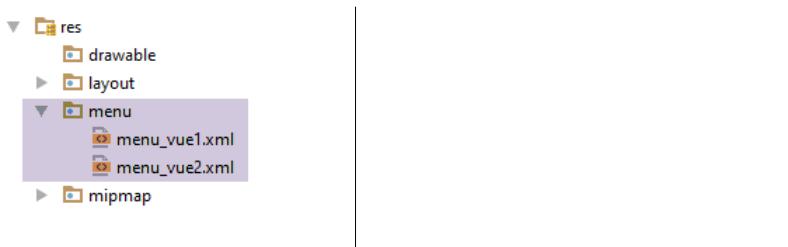
1.21.1 Cration du projet

Nous dupliquons le projet [Exemple-19B] dans le projet [Exemple-20] :



Nous allons supprimer les boutons des vues 1 et 2 pour les remplacer par des options de menu [1-2].

1.21.2 La definition XML des menus



Le fichier [res / menu / menu_vue1] definit le menu de la vue n° 1 :

```
1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:app="http://schemas.android.com/apk/res-auto"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     tools:context=".activity.MainActivity">
5.     <item
6.         android:id="@+id/menuOptions"
7.         app:showAsAction="ifRoom"
8.         android:title="@string/menuOptions">
9.         <menu>
10.            <item
11.                android:id="@+id/actionCacherMontrerTout"
```

```

12.        android:title="@string/actionCacherMontrerTout"/>
13.    <item
14.        android:id="@+id/actionCacherMontrerActions"
15.        android:title="@string/actionCacherMontrerActions"/>
16.    <item
17.        android:id="@+id/actionCacherMontrerActionsValider"
18.        android:title="@string/actionCacherMontrerActionsValider"/>
19.    </menu>
20.  </item>
21. <item
22.    android:id="@+id/menuActions"
23.    app:showAsAction="ifRoom"
24.    android:title="@string/menuActions">
25.    <menu>
26.      <item
27.          android:id="@+id/actionValider"
28.          android:title="@string/actionValider"/>
29.    </menu>
30.  </item>
31. <item
32.    android:id="@+id/menuNavigation"
33.    app:showAsAction="ifRoom"
34.    android:title="@string/menuNavigation">
35.    <menu>
36.      <item
37.          android:id="@+id/navigationVue2"
38.          android:title="@string/navigationVue2"/>
39.    </menu>
40.  </item>
41. </menu>

```

Les éléments du menu sont définis par les informations suivantes :

- **android:id** : l'identifiant de l'élément ;
- **android:title** : le libellé de l'élément ;
- **app:showsAsAction** : indique si l'élément de menu peut être placé dans la barre d'actions de l'activité. [ifRoom] indique que l'élément doit être placé dans la barre d'actions s'il y a de la place pour lui ;
- une option de menu peut elle-même être un sous-menu (balise <menu>, lignes 25, 29) ;

Le fichier [res / menu / menu_vue2] définit le menu de la vue n° 2 :

```

1.  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.    xmlns:app="http://schemas.android.com/apk/res-auto"
3.    xmlns:tools="http://schemas.android.com/tools"
4.    tools:context=".activity.MainActivity">
5.    <item
6.      android:id="@+id/menuNavigation"
7.      app:showAsAction="ifRoom"
8.      android:title="@string/menuNavigation">
9.      <menu>
10.        <item
11.            android:id="@+id/navigationVue1"
12.            android:title="@string/navigationVue1"/>
13.      </menu>
14.    </item>
15.  </menu>

```

1.21.3 La gestion du menu dans la classe abstraite [AbstractFragment]

Nous allons factoriser la gestion du menu dans la classe parent [AbstractFragment] des deux vues :

```

1. package exemples.android.architecture;
2.
3. import android.app.Activity;
4. import android.support.v4.app.Fragment;
5. import android.util.Log;
6. import android.view.Menu;
7. import android.view.MenuInflater;
8. import android.view.MenuItem;
9.
10. import java.util.ArrayList;
11. import java.util.List;
12.
13. public abstract class AbstractFragment extends Fragment {
14.
15.     // données accessibles aux classes filles
16.     final protected boolean isDebugEnabled = IMainActivity.IS_DEBUG_ENABLED;
17.     protected String className;
18.
19.     // activité
20.     protected IMainActivity mainActivity;
21.     protected Activity activity;

```

```

22.
23.    // session
24.    protected Session session;
25.
26.    // menu
27.    private Menu menu;
28.    private int[] menuOptions;
29.    private boolean initDone;
30.
31.    // constructeur
32.    public AbstractFragment() {
33.        // init
34.        className = getClass().getSimpleName();
35.        // log
36.        if (isDebugEnabled) {
37.            Log.d("AbstractFragment", String.format("constructor %s", className));
38.        }
39.    }
40.
41.    @Override
42.    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
43.        // mémoire
44.        this.menu = menu;
45.        // log
46.        if (isDebugEnabled) {
47.            Log.d(className, String.format("création menu en cours"));
48.        }
49.        // on récupère les # options du menu si ce n'a pas déjà été fait
50.        if (!initDone) {
51.            // on récupère les # options du menu
52.            List<Integer> menuOptionsIds = new ArrayList<>();
53.            menuOptions = getMenuOptions(menu, menuOptionsIds);
54.            // activité
55.            this.activity = getActivity();
56.            this.mainActivity = (IMainActivity) activity;
57.            this.session = this.mainActivity.getSession();
58.            // mémoire
59.            initDone = true;
60.        }
61.
62.        // on demande au fragment fille de se mettre
63.        updateFragment();
64.    }
65.
66.    private int[] getMenuOptions(Menu menu, List<Integer> menuOptionsIds) {
67.        ...
68.    }
69.
70.    // affichage options de menu -----
71.    protected void setAllMenuOptions(boolean isVisible) {
72.        ....
73.    }
74.
75.    protected void setMenuOptions(MenuItemState[] menuItemStates) {
76.        ...
77.    }
78.
79.    // update classe fille
80.    protected abstract void updateFragment();
81. }

```

- ligne 42 : les logs montrent que la méthode [onCreateOptionsMenu] est appelée à chaque fois que le fragment est affiché. Elle est appelée très tard, après que la méthode [updateFragment] ait été appelée. Cela suggère qu'elle pourrait être utilisée pour mettre à jour le fragment. C'est ce que nous allons faire ici (ligne 63) ;
- ligne 42 : la méthode a deux paramètres :
 - [menu] : qui est un menu vide ;
 - [inflater] : un outil qui permet de créer le menu à partir de sa description initiale. Nous n'utiliserons pas cette possibilité ici car nous utiliserons une annotation AA qui le fera pour nous ;
- ligne 44 : nous mémorisons le menu. Nous en aurons besoin ultérieurement ;
- lignes 52-53 : nous stockons dans le tableau de la ligne 28 les identifiants de tous les éléments du menu ;
- lignes 55-57 : les logs montrent que lorsque la méthode [onCreateOptionsMenu] est appelée, la méthode [Fragment.getActivity()] rend l'activité associée au fragment ;
- ligne 55 : nous mémorisons l'activité comme instance de la classe Android [Activity] ;
- ligne 56 : nous mémorisons l'activité comme instance de l'interface [IMainActivity] ;
- ligne 57 : nous mémorisons la session ;
- ligne 59 : nous notons que l'initialisation de la classe a été faite pour ne pas avoir à la refaire (ligne 50) ;
- ligne 63 : on demande au fragment fille de se mettre à jour. C'est possible parce que le fragment est à la fois visible, associé à sa vue et à son menu ;

La méthode [getMenuOptions] qui permet d'avoir les identifiants des éléments d'un menu est la suivante :

```
1. private int[] getMenuOptions(Menu menu, List<Integer> menuOptionsIds) {
2.     // on parcourt tous les items du menu
3.     for (int i = 0; i < menu.size(); i++) {
4.         // item n° i
5.         MenuItem menuItem = menu.getItem(i);
6.         menuOptionsIds.add(menuItem.getItemId());
7.         // si item n° i est un sous-menu, alors on recommence
8.         if (menuItem.hasSubMenu()) {
9.             // récursivité
10.            getMenuOptions(menuItem.getSubMenu(), menuOptionsIds);
11.        }
12.    }
13.    // on transfère la liste d'Integer dans un tableau de int
14.    int[] menuOptions = new int[menuOptionsIds.size()];
15.    for (int i = 0; i < menuOptions.length; i++) {
16.        menuOptions[i] = menuOptionsIds.get(i);
17.    }
18.    // résultat
19.    return menuOptions;
20. }
```

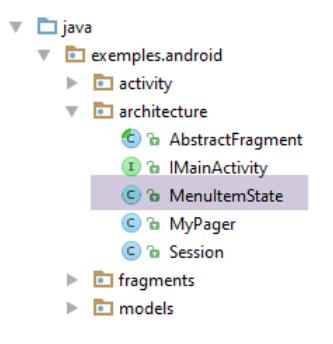
La méthode [setAllMenuOptions] permet de cacher / montrer l'ensemble des options du menu ;

```
1. protected void setAllMenuOptions(boolean isVisible) {
2.     // on met à jour toutes les options du menu
3.     for (int menuItemId : menuOptions) {
4.         menu.findItem(menuItemId).setVisible(isVisible);
5.     }
6. }
```

La méthode [setMenuOptions] permet de cacher / montrer certaines des options du menu ;

```
1. protected void setMenuOptions(MenuItemState[] menuItemStates) {
2.     // on met à jour certaines options du menu
3.     for (MenuItemState menuItemState : menuItemStates) {
4.         menu.findItem(menuItemState.getMenuItemId()).setVisible(menuItemState.isVisible());
5.     }
6. }
```

La classe [MenuItemState] est la suivante :



```
1. package exemplles.android.architecture;
2.
3. public class MenuItemState {
4.
5.     // identifiant de l'option de menu
6.     private int menuItemId;
7.     // visibilité de l'option
8.     private boolean isVisible;
9.
10.    // constructeurs
11.    public MenuItemState() {
12.
13.    }
14.
15.    public MenuItemState(int menuItemId, boolean isVisible) {
16.        this.menuItemId = menuItemId;
17.        this.isVisible = isVisible;
18.    }
19.
20.    // getters et setters
21.    ...
```

1.21.4 La gestion du menu dans le fragment [Vue1Fragment]

La classe [Vue1Fragment] devient la suivante :

```

1.  @EFragment(R.layout.vue1)
2.  @OptionsMenu(R.menu.menu_vue1)
3.  public class Vue1Fragment extends AbstractFragment {
4.
5.  ...
6.
7.  @OptionsItem(R.id.navigationVue2)
8.  void navigateToView2() {
9.      // on navigue vers la vue 2
10.     MainActivity.navigateToView(1);
11. }
12.
13. @OptionsItem(R.id.actionValider)
14. void valider() {
15.     // on affiche un message
16.     Toast.makeText(activity, "Valider", Toast.LENGTH_SHORT).show();
17. }
18.
19. private boolean actionCacherMontrerTout = true;
20. @OptionsItem(R.id.actionCacherMontrerTout)
21. void cacherMontrerTout() {
22.     // on change d'état
23.     actionCacherMontrerTout = !actionCacherMontrerTout;
24.     setMenuOptions(new MenuItemState[]{new MenuItemState(R.id.menuNavigation, actionCacherMontrerTout), new
MenuItemState(R.id.menuActions, actionCacherMontrerTout)});
25. }
26.
27. private boolean actionCacherMontrerActions = true;
28. @OptionsItem(R.id.actionCacherMontrerActions)
29. void actionCacherMontrerActions() {
30.     // on change d'état
31.     actionCacherMontrerActions = !actionCacherMontrerActions;
32.     setMenuOptions(new MenuItemState[]{new MenuItemState(R.id.menuActions, actionCacherMontrerActions)});
33. }
34.
35. private boolean actionCacherMontrerActionsValider = true;
36. @OptionsItem(R.id.actionCacherMontrerActionsValider)
37. void actionCacherMontrerActionsValider() {
38.     // on change d'état
39.     actionCacherMontrerActionsValider = !actionCacherMontrerActionsValider;
40.     setMenuOptions(new MenuItemState[]{new MenuItemState(R.id.menuActions, true), new MenuItemState(R.id.actionValider,
actionCacherMontrerActionsValider)});
41. }
42. ...
43.
44. @Override
45. protected void updateFragment() {
46.     ...
47.     // on met à jour le menu
48.     //setMenuOptions(...)
49. }
50. }
```

- ligne 2 : le menu [res / menu / menu_vue1.xml] est associé au fragment ;
- ligne 48 : lorsque la méthode [updateFragment] s'exécute, le menu peut être lui aussi mis à jour pour refléter le nouvel état du fragment ;
- ligne 7 : l'annotation [`@OptionsItem(R.id.navigationVue2)`] annote la méthode qui doit être exécutée lors d'un clic sur l'option de menu [Navigation / Vue 2] ;
- lignes 19-25 : pour cacher une branche du menu, il suffit de cacher l'option racine de celle-ci ;
- ligne 24 : on montre / cache les options racine [menuNavigation, menuActions] ;
- ligne 40 : pour montrer une option d'une branche de menu, il faut non seulement montrer celle-ci mais également toutes les options qu'on rencontre en remontant de l'option feuille vers la racine du menu ;

1.21.5 La gestion du menu dans le fragment [Vue2Fragment]

On retrouve un code similaire dans le fragment de la vue n° 2 :

```

1. package exemples.android.fragments;
2.
3. import android.widget.TextView;
4. import exemples.android.R;
5. import exemples.android.architecture.AbstractFragment;
```

```

6. import exemples.android.models.CheckedData;
7. import org.androidannotations.annotations.*;
8.
9. @EFragment(R.layout.vue2)
10. @OptionsMenu(R.menu.menu_vue2)
11. public class Vue2Fragment extends AbstractFragment {
12.
13.     // les champs de la vue
14.     @ViewById(R.id.textViewResultats)
15.     TextView txtResultats;
16.
17.     @OptionsItem(R.id.navigationVue1)
18.     void navigateToView1() {
19.         // on navigue vers la vue 1
20.         MainActivity.navigateToIntView(0);
21.     }
22.
23.     @Override
24.     protected void updateFragment() {
25.         // on affiche les éléments de la liste qui ont été sélectionnés dans la vue 1
26.         StringBuider texte = new StringBuider("Eléments sélectionnés [");
27.         for (CheckedData data : session.getListe()) {
28.             if (data.isChecked()) {
29.                 texte.append(String.format("(%s)", data.getTexte()));
30.             }
31.         }
32.         texte.append("]");
33.         txtResultats.setText(texte);
34.         // on met à jour le menu
35.         // setMenuOptions(...)
36.     }
37. }
```

- ligne 35 : on affiche l'option [Navigation / Vue 1] ;
- lignes 17-20 : lors du clic sur l'option [Navigation / Vue1], on appelle la méthode [navigateToIntView] ;

1.21.6 Exécution

Créez un contexte d'exécution pour ce projet et exécutez-le.

1.22 Exemple-21 : refactoring de la classe abstraite [AbstractFragment]

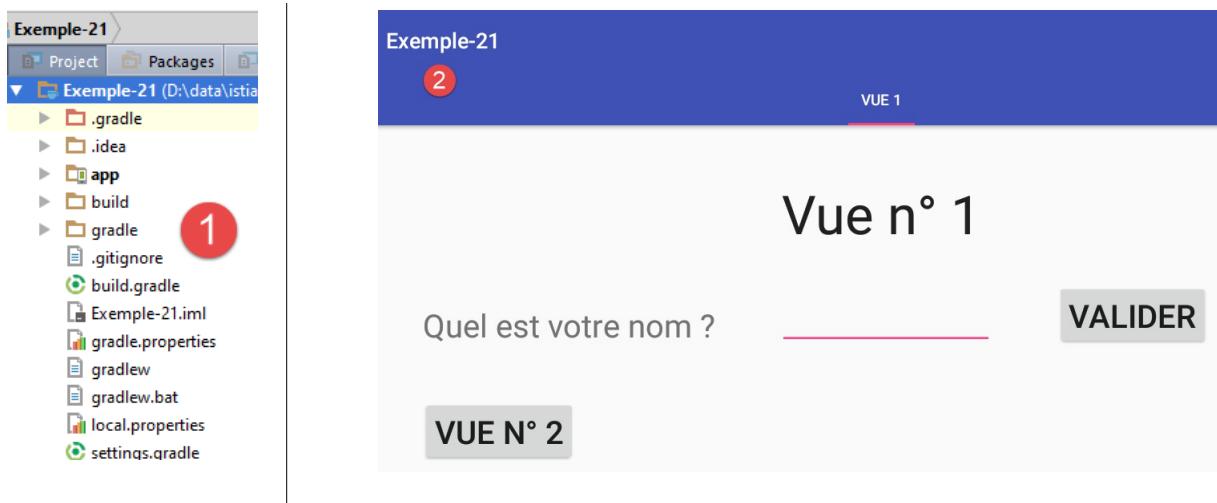
L'exemple précédent nous a montré que lorsque le fragment a un menu, sa méthode [onCreateOptionsMenu] est un bon endroit pour demander au fragment de se mettre à jour :

- elle est appelée exactement une fois lorsque le fragment va s'afficher ;
- lorsqu'elle est appelée, les associations du fragment avec son activité, sa vue et son menu sont faites ;

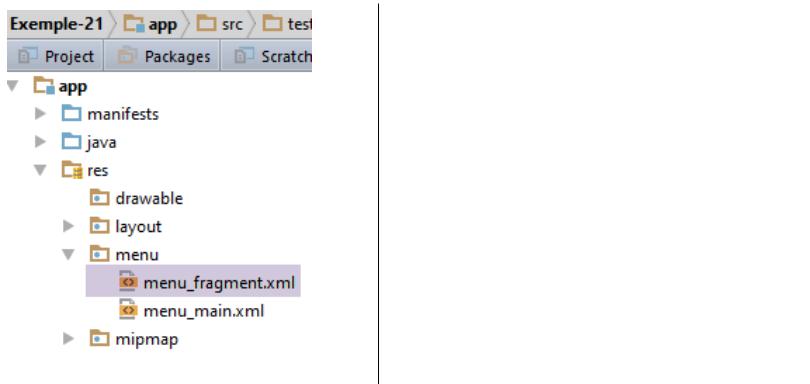
Pour le montrer, on reprend l'exemple 12 qui a la propriété d'avoir beaucoup de fragments dont on peut modifier l'adjacence. Dans cet exemple, les fragments n'avaient pas de menu. On va leur associer un menu vide.

1.22.1 Crédation du projet

Nous dupliquons le projet [Exemple-12] dans le projet [Exemple-21] :



1.22.2 Le menu des fragments



Le menu ajouté par les fragments sera vide :

```
1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.   xmlns:app="http://schemas.android.com/apk/res-auto"
3.   xmlns:tools="http://schemas.android.com/tools"
4.   tools:context="exemples.android.MainActivity">
5. </menu>
```

Ce qu'il faut comprendre ici c'est que l'activité a déjà son propre menu [menu_main] :

```
1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.   xmlns:app="http://schemas.android.com/apk/res-auto"
```

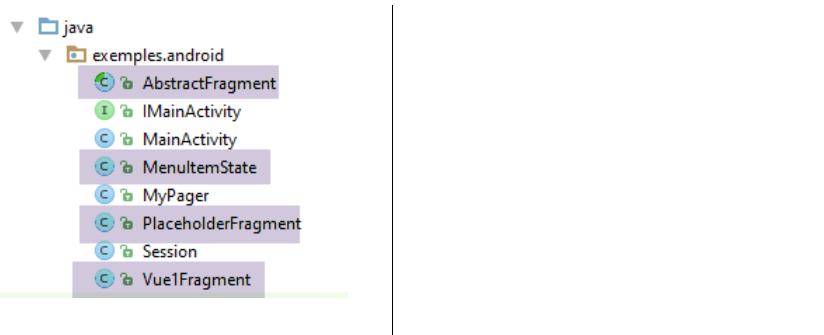
```

3.      xmlns:tools="http://schemas.android.com/tools"
4.      tools:context="exemples.android.MainActivity"
5.      <item android:id="@+id/action_settings"
6.          android:title="@string/action_settings"
7.          android:orderInCategory="100"
8.          app:showAsAction="never"/>
9.      <item android:id="@+id/fragment1"
10.         android:title="@string/fragment1"
11.         android:orderInCategory="100"
12.         app:showAsAction="never"/>
13.      <item android:id="@+id/fragment2"
14.         android:title="@string/fragment2"
15.         android:orderInCategory="100"
16.         app:showAsAction="never"/>
17.      <item android:id="@+id/fragment3"
18.         android:title="@string/fragment3"
19.         android:orderInCategory="100"
20.         app:showAsAction="never"/>
21.      <item android:id="@+id/fragment4"
22.         android:title="@string/fragment4"
23.         android:orderInCategory="100"
24.         app:showAsAction="never"/>
25.  </menu>

```

Lorsqu'une activité a déjà un menu, le menu associé aux fragments s'ajoute à celui de l'activité : on a donc les options de deux menus. Ici, le menu des fragments sera vide. Donc on ne verra que le menu de l'activité.

1.22.3 Les fragments



Nous reprenons la classe abstraite [AbstractFragment] de l'exemple précédent (cf paragraphe 1.21.3, page 235).

Nous associons le menu [menu_fragment] aux deux fragments :

```

1.  @EFragment(R.layout.fragment_main)
2.  @OptionsMenu(R.menu.menu_fragment)
3.  public class PlaceholderFragment extends AbstractFragment {

1.  @EFragment(R.layout.vue1)
2.  @OptionsMenu(R.menu.menu_fragment)
3.  public class Vue1Fragment extends AbstractFragment {

```

Dans les deux fragments [PlaceholderFragment] et [Vue1Fragment], nous nous débarrassons de ce qui fait référence à l'ancienne classe abstraite [AbstractFragment].

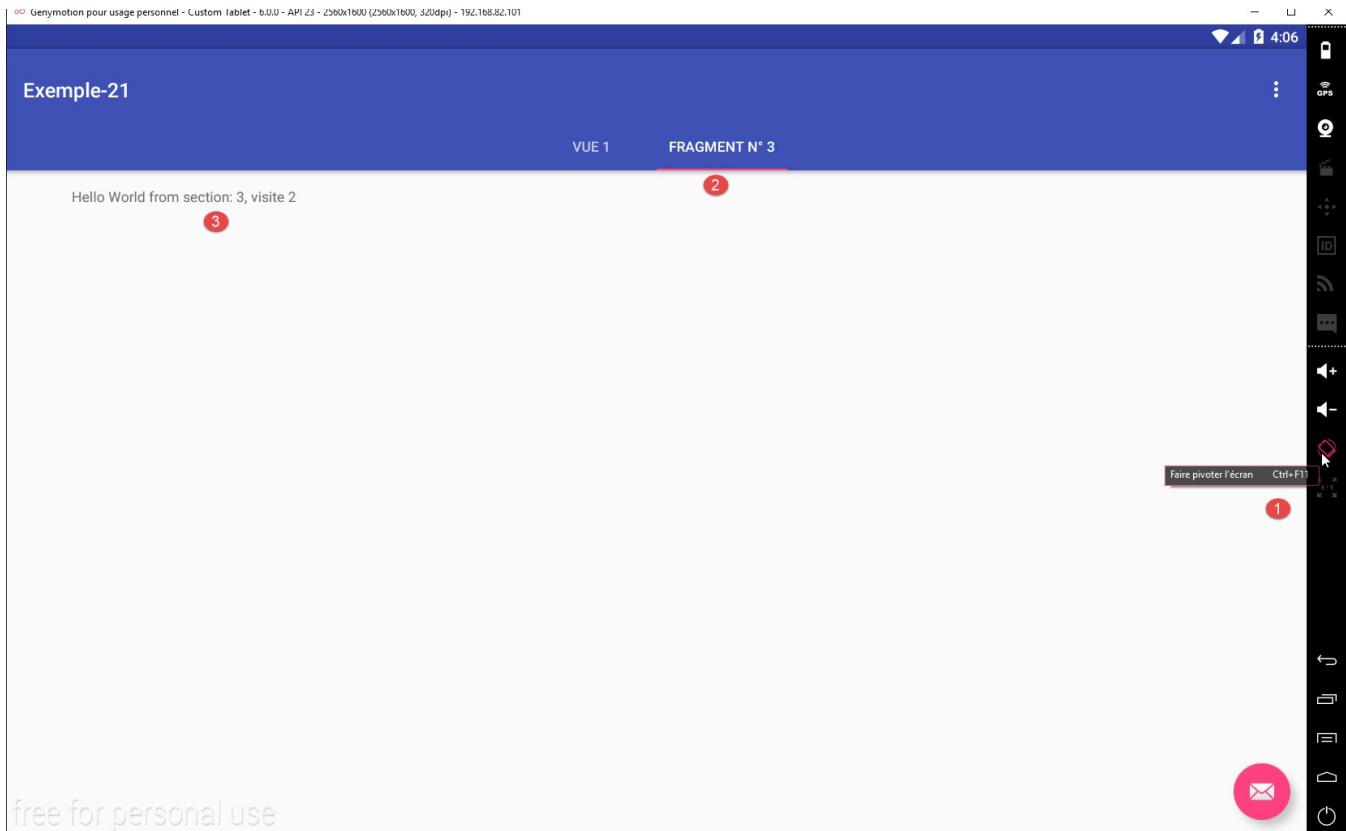
1.22.4 Exécution

Exécutez l'application et constatez qu'elle fonctionne. Suivez les logs pour voir quand la méthode [onCreateOptionsMenu] de la classe [AbstractFragment] est exécutée. C'est désormais elle qui appelle la méthode [updateFragment] des fragments filles.

1.23 Exemple-22 : sauvegarde / restauration de l'état de l'activité et des fragments

1.23.1 Le problème

Nous abordons ici le problème de la rotation du périphérique Android (portrait <--> paysage). Pour l'illustrer, nous reprenons l'exemple 21 précédent :



Si nous faisons tourner le périphérique [1], nous obtenons la nouvelle vue suivante :



On voit que :

- en [1], l'onglet [Fragment n° 3] a disparu ;
- en [2], le texte affiché est bien celui du fragment n° 3 mais le compteur de visites est incorrect ;

Lors de cette rotation, les logs sont les suivants :

```

1. 07-13 04:08:27.188 1677-1677/exemples.android D/MainActivity: constructor
2. 07-13 04:08:27.189 1677-1677/exemples.android D/MainActivity: afterInject
3. 07-13 04:08:27.190 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
4. 07-13 04:08:27.190 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
5. 07-13 04:08:27.190 1677-1677/exemples.android D/AbstractFragment: constructor Vue1Fragment_
6. 07-13 04:08:27.190 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
7. 07-13 04:08:27.190 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
8. 07-13 04:08:27.194 1677-1677/exemples.android D/MainActivity: afterViews
9. 07-13 04:08:27.195 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
10. 07-13 04:08:27.195 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
11. 07-13 04:08:27.195 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
12. 07-13 04:08:27.195 1677-1677/exemples.android D/AbstractFragment: constructor PlaceholderFragment_
13. 07-13 04:08:27.195 1677-1677/exemples.android D/AbstractFragment: constructor Vue1Fragment_
14. 07-13 04:08:27.203 1677-1677/exemples.android D/PlaceholderFragment: afterViews 4 - PlaceholderFragment_ - numVisit=0,
initDone=false, getActivity()==null:false
15. 07-13 04:08:27.204 1677-1677/exemples.android D/PlaceholderFragment: afterViews 3 - PlaceholderFragment_ - numVisit=0,
initDone=false, getActivity()==null:false
16. 07-13 04:08:27.208 1677-1677/exemples.android D/Vue1Fragment: afterViews Vue1Fragment_ - numVisit=0

```

```

17. 07-13 04:08:27.208 1677-1677/exemples.android D/PlaceholderFragment: afterViews 2 - PlaceholderFragment_ - numVisit=0,
    initDone=false, getActivity()==null:false
18. 07-13 04:08:27.209 1677-1677/exemples.android D/PlaceholderFragment: afterViews 1 - PlaceholderFragment_ - numVisit=0,
    initDone=false, getActivity()==null:false
19. 07-13 04:08:27.351 1677-1677/exemples.android D/menu: création menu en cours
20. 07-13 04:08:27.351 1677-1677/exemples.android D/PlaceholderFragment_: création menu en cours
21. 07-13 04:08:27.351 1677-1677/exemples.android D/PlaceholderFragment: update 3 - PlaceholderFragment_ - numVisit=0,
    initDone=true, getActivity()==null:false

```

- ligne 1 : on voit que l'activité est totalement reconstruite ;
- lignes 3-7 : il en est de même pour les cinq fragments gérés par l'activité ;
- ligne 21 : le fragment n° 3 va être affiché. On voit qu'avant incrémentation, le n° de la visite est 0 ;

On peut alors expliquer le résultat obtenu après rotation de la façon suivante :

- la classe [MainActivity] crée au départ une barre d'onglet avec un seul onglet, libellé [Vue 1]. C'est l'onglet qu'on voie ;
- après la rotation du périphérique, le gestionnaire de pages [mViewPager] réaffiche le même fragment, donc ici le fragment n° 3. Il faut se souvenir ici que onglets et fragments sont des notions différentes et ont un cycle de vie différent. La méthode [updateFragment] du fragment n° 3 va s'exécuter :

```

1.   public void updateFragment() {
2.     // log
3.     if (isDebugEnabled) {
4.       Log.d("PlaceholderFragment", String.format("update %s - %s - %s", getArguments().getInt(ARG_SECTION_NUMBER),
5.         className, getLocalInfos()));
6.     }
7.     // incrément n° de visite
8.     numVisit = session.getNumVisit();
9.     numVisit++;
10.    session.setNumVisit(numVisit);
11.    // texte modifié
12.    textViewInfo.setText(String.format("%s, visite %s", text, numVisit));
13.  }

```

- ligne 7 : le dernier n° de la visite est lu dans la session. Or celle-ci, comme tout le reste, a été reconstruite et le n° de visite a été remis à zéro. Ce qui explique le résultat affiché dans le fragment n° 3 ;

1.23.2 Les méthodes de sauvegarde / restauration de l'activité et des fragments

1.23.2.1 Solution 1 : sauvegarde manuelle

Lors de la rotation du périphérique, deux méthodes de l'activité sont appelées :

```

1. // gestion sauvegarde / restauration de l'activité -----
2. @Override
3. protected void onSaveInstanceState(Bundle outState) {
4.   // parent
5.   super.onSaveInstanceState(outState);
6.   // sauvegarde état de l'activité
7.   // ....
8. }
9.
10. @Override
11. protected void onCreate(Bundle savedInstanceState) {
12.   // parent
13.   super.onCreate(savedInstanceState);
14.   // restauration de l'activité
15.   // ...
16. }

```

- lignes 2-8 : la méthode [onSaveInstanceState] est appelée par le système lors de la rotation. C'est l'endroit où la sauvegarde de l'activité peut se faire. Si on ne fait rien, rien n'est sauvegardé. La sauvegarde de l'état de l'activité doit se faire dans le paramètre [Bundle outState] passé à la méthode. La classe [Bundle] ressemble à un dictionnaire. Elle possède des méthodes [putString, putInt, putLong, putBoolean, putChar, ...] à deux paramètres : **void putT(String key, T value)** ;
- lignes 10-16 : la méthode [onCreate] est appelée lors de la création de l'activité. Si l'état de celle-ci a été sauvegardé, cette sauvegarde lui est passée dans le paramètre [Bundle savedInstanceState]. Pour récupérer les valeurs sauvegardées, on dispose de méthodes telles que [getString, getInt, getLong, getBoolean, getChar, ...] à un paramètre : **T getT(String key)** ;

Les fragments disposent de ces deux mêmes méthodes pour sauvegarder leur état.

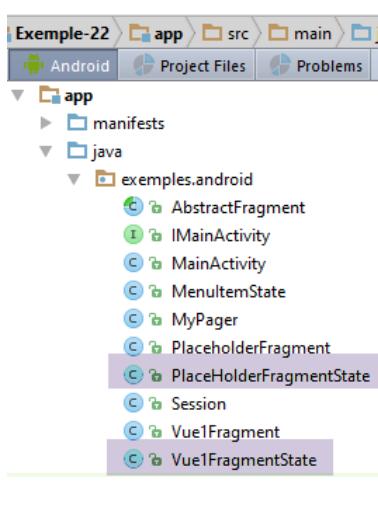
Nous allons utiliser ces informations pour sauvegarder et restaurer l'état de l'exemple 21. Pour cela nous dupliquons le projet [Exemple-21] dans [Exemple-22].

1.23.2.2 Solution 2 : sauvegarde automatique

La documentation Android indique que lors de la rotation du périphérique, on peut éviter la destruction d'un fragment en utilisant l'instruction : `[Fragment].setRetainInstance(true)`. Plusieurs articles de [StackOverflow] préconisent d'utiliser cette instruction uniquement pour les fragments sans interface visuelle [<http://stackoverflow.com/questions/11182180/understanding-fragments-setretaininstanceboolean>, <http://stackoverflow.com/questions/12640316/further-understanding-setretaininstancetrue>, <http://stackoverflow.com/questions/21203948/setretaininstancetrue-in-oncreate-fragment-in-android>]. J'ai testé cette instruction sur deux exemples : **Exemple-17** (paragraphe 1.18, page 210 - une application à un fragment qui affiche un formulaire) et **Exemple-21** (paragraphe 1.22, page 240), une application à cinq fragments). Dans les deux cas, cette seule instruction appliquée à tous les fragments de l'application s'est révélée insuffisante pour restaurer correctement la vue affichée lors de la rotation du périphérique. Plutôt que de construire deux modèles, l'un basé sur `[setRetainInstance(true)]` et un autre basé sur `[setRetainInstance(false)]` qui est la valeur par défaut, j'ai décidé de suivre les préconisations de [StackOverflow] et de garder la valeur *false* par défaut de la méthode `[setRetainInstance(boolean)]`. L'instruction : `[Fragment].setRetainInstance(true)` n'a jamais été utilisée dans la suite de ce document.

1.23.3 La méthode de sauvegarde / restauration du projet [Exemple-22]

Le projet [Exemple-22] évolue comme suit :



On y voit apparaître deux nouvelles classes :

- [PlaceHolderFragmentState] qui mémorisera l'état d'un fragment de type [PlaceholderFragment] ;
- [Vue1FragmentManager] qui mémorisera l'état du fragment de type [Vue1Fragment] ;

Ces classes sont les suivantes :

```
1. package exemples.android;
2.
3. public class Vue1FragmentManager {
4.     // état Vue1Fragment
5.     private boolean hasBeenVisited=false;
6.     // getters et setters
7.     ...
8. }
```

- ligne 5 : le booléen [hasBeenVisited] est vrai si le fragment [Vue1Fragment] a été visité (affiché) au moins une fois. Ce champ a été créé pour l'exemple car le fragment [Vue1Fragment] n'a rien à sauvegarder ;

La classe [PlaceHolderFragmentState] est la suivante :

```
1. package exemples.android;
2.
3. public class PlaceHolderFragmentState {
4.     // état visité ou pas
5.     private boolean hasBeenVisited;
6.     // texte affiché
7.     private String text;
8.
9.     // getters et setters
10.    ...
```

11. }

- ligne 5 : on retrouve le booléen [hasBeenVisited] ;
- ligne 7 : le texte affiché par le fragment lorsqu'il doit être sauvegardé. On a vu que ce texte avait été perdu lors de la rotation ;

L'état des fragments va être stocké dans la session et c'est l'activité qui aura la charge de sauvegarder / restaurer cette session. La session évolue de la façon suivante :

```
1. package exemples.android;
2.
3. import com.fasterxml.jackson.annotation.JsonIgnore;
4. import org.androidannotations.annotations.EBean;
5.
6. @EBean(scope = EBean.Scope.Singleton)
7. public class Session {
8.     // nombre de fragments visités
9.     private int numVisit;
10.    // n° fragment de type [PlaceholderFragment] affiché dans second onglet
11.    private int numFragment = -1;
12.    // n° onglet sélectionné
13.    private int selectedTab = 0;
14.    // n° vue courante
15.    private int currentView;
16.
17.    // sauvegardes fragments -----
18.    private Vue1FragmentState vue1FragmentState;
19.    private PlaceHolderFragmentState[] placeHolderFragmentStates = new PlaceHolderFragmentState[IMainActivity.FRAGMENTS_COUNT
- 1];
20.
21.    // constructeur
22.    public Session() {
23.        for (int i = 0; i < placeHolderFragmentStates.length; i++) {
24.            placeHolderFragmentStates[i] = new PlaceHolderFragmentState();
25.        }
26.        vue1FragmentState = new Vue1FragmentState();
27.    }
28.    // getters et setters
29. ...
30. }
```

- ligne 18 : l'état du fragment [Vue1Fragment] ;
- ligne 19 : l'état des fragments de type [PlaceHolderFragment] ;
- lignes 22-27 : dans le constructeur de la session, on initialise les champs des lignes 18 et 19 ;
- lignes 12-15 : deux nouveaux champs apparaissent :
 - ligne 13 : le n° du dernier onglet sélectionné ;
 - ligne 15 : le n° du dernier fragment affiché ;

L'activité sauvegarde / restaure la session de la façon suivante :

```
1.    // gestion sauvegarde / restauration de l'activité -----
2.    @Override
3.    protected void onSaveInstanceState(Bundle outState) {
4.        // parent
5.        super.onSaveInstanceState(outState);
6.        // sauvegarde session
7.        try {
8.            outState.putString("session", jsonMapper.writeValueAsString(session));
9.        } catch (JsonProcessingException e) {
10.            e.printStackTrace();
11.        }
12.        // log
13.        if (IS_DEBUG_ENABLED) {
14.            try {
15.                Log.d(className, String.format("onSaveInstanceState session=%s", jsonMapper.writeValueAsString(session)));
16.            } catch (JsonProcessingException e) {
17.                e.printStackTrace();
18.            }
19.        }
20.    }
21.
22.    @Override
23.    protected void onCreate(Bundle savedInstanceState) {
24.        // parent
25.        super.onCreate(savedInstanceState);
26.        if (savedInstanceState != null) {
27.            // récupération session
28.            try {
29.                session = jsonMapper.readValue(savedInstanceState.getString("session"), new TypeReference<Session>() {
30.            });
31.        }
```

```

31.     } catch (IOException e) {
32.         e.printStackTrace();
33.     }
34.     // log
35.     if (IS_DEBUG_ENABLED) {
36.         try {
37.             Log.d(className, String.format("onCreate session=%s", jsonMapper.writeValueAsString(session)));
38.         } catch (JsonProcessingException e) {
39.             e.printStackTrace();
40.         }
41.     }
42. }
43. }
```

- ligne 8 : on sauvegarde la session sous la forme de sa chaîne JSON ;
- ligne 29 : on restaure la session à partir de sa chaîne JSON ;

Pour gérer la sauvegarde / restauration des fragments, la classe abstraite [AbstractFragment] évolue de la façon suivante :

```

1.  // gestion sauvegarde / restauration -----
2.  @Override
3.  public void setUserVisibleHint(boolean isVisibleToUser) {
4.      // parent
5.      super.setUserVisibleHint(isVisibleToUser);
6.      // sauvegarde ?
7.      if (this.isVisibleToUser && !isVisibleToUser && !saveFragmentDone) {
8.          // le fragment va être caché - on le sauvegarde
9.          saveFragment();
10.         saveFragmentDone = true;
11.     }
12.     // mémoire
13.     this.isVisibleToUser = isVisibleToUser;
14. }
15.
16. @Override
17. public void onActivityCreated(Bundle savedInstanceState) {
18.     // parent
19.     super.onActivityCreated(savedInstanceState);
20.     // log
21.     if (isDebugEnabled) {
22.         Log.d(className, "onActivityCreated");
23.     }
24.     // le fragment doit être restauré
25.     fragmentHasToBeInitialized = true;
26. }
27.
28.
29. @Override
30. public void onSaveInstanceState(Bundle outState) {
31.     // log
32.     if (isDebugEnabled) {
33.         Log.d(className, "onSaveInstanceState");
34.     }
35.     // parent
36.     super.onSaveInstanceState(outState);
37.     // sauvegarde du fragment seulement s'il est visible
38.     if (isVisibleToUser && !saveFragmentDone) {
39.         saveFragment();
40.         saveFragmentDone = true;
41.     }
42. }
43.
44. // classes filles
45. protected abstract void updateFragment();
46.
47. protected abstract void saveFragment();
```

- on décide de sauvegarder l'état des fragments dans la session à deux moments :
 - lignes 2-14 : lorsque le fragment passe de visible à caché ;
 - lignes 29-42 : lorsque le système indique qu'il faut faire une sauvegarde du fragment et que celui-ci est visible (ligne 38) ;

Ce mécanisme évite de faire des sauvegardes plus souvent que nécessaire. En effet, comme on a sauvegardé l'état du fragment i lorsqu'il est passé de visible à caché, lorsque le fragment j est affiché et qu'on fait une rotation, il est inutile de sauvegarder de nouveau le fragment i. S'il n'a pas été réaffiché depuis sa dernière sauvegarde, alors son état n'a pas changé. Seul l'état du fragment j doit être sauvegardé. Cette mécanique a également un autre avantage : il n'y a pas que lors d'une rotation du périphérique qu'on a besoin de sauvegarder l'état d'un fragment. Il y a aussi le cas de la navigation pure entre fragments par exemple dans un système à onglets. On veut alors retrouver un fragment dans l'état où on l'a laissé la dernière fois qu'il était affiché. Cet état peut avoir en partie disparu si ce fragment est sorti à un moment donné de

l'adjacence des fragments affichés. Le fragment n'est alors pas reconstruit dans sa totalité mais sa vue associée elle l'est. La sauvegarde qu'on a faite lorsque le fragment est devenu caché va servir à retrouver le dernier état de cette vue ;

- lignes 10, 40 : pour éviter de faire deux sauvegardes successives, on utilise le booléen [saveFragmentDone] pour indiquer qu'une sauvegarde a été faite ;
- lignes 9, 39 : on demande au fragment fille de sauvegarder son état. La méthode [saveFragment] est abstraite (ligne 47). C'est donc aux classes filles de l'implémenter ;
- lignes 16-26 : la méthode [onActivityCreated] est utilisée pour positionner le booléen [fragmentHasToBeInitialized] à vrai. En effet, le fragment fille doit savoir qu'elle doit réinitialiser entièrement l'état du fragment à partir d'un état qu'elle trouvera dans la session ;

Toujours dans la classe [AbstractFragment], la méthode [onCreateOptionsMenu] évolue comme suit :

```
1. // mise à jour du fragment
2. @Override
3. public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
4.     // mémoire
5.     this.menu = menu;
6.     // log
7.     if (isDebugEnabled) {
8.         Log.d(className, String.format("création menu en cours"));
9.     }
10.    ...
11.    // on demande au fragment fille de se mettre à jour
12.    updateFragment();
13.    // sauvegarde à faire
14.    saveFragmentDone = false;
15. }
```

- ligne 14 : on a vu que le booléen [saveFragmentDone] était passé à *vrai* lorsqu'une sauvegarde avait été faite. Il faut à un moment donné qu'il revienne à *faux*. Lorsque la méthode [updateFragment] (ligne 12) du fragment fille est exécuté, celui-ci va devenir visible. Or c'est lorsqu'il est visible qu'un fragment est sauvegardé ;

1.23.4 Sauvegarde du fragment [Vue1Fragment]

La sauvegarde des fragments se fait dans la méthode [saveFragment] appelée par la classe mère [AbstractFragment] :

```
1. // sauvegarde état fragment
2. @Override
3. public void saveFragment() {
4.     // log
5.     if (isDebugEnabled) {
6.         Log.d(className, String.format("saveFragment 1 %s - %s", className, getLocalInfos()));
7.     }
8.     // sauvegarde en session de l'état du fragment
9.     Vue1FragmentState state = new Vue1FragmentState();
10.    state.setHasBeenVisited(true);
11.    session.setVue1FragmentState(state);
12.    // log
13.    if (isDebugEnabled) {
14.        try {
15.            Log.d(className, String.format("saveFragment 2 state=%s", jsonMapper.writeValueAsString(state)));
16.        } catch (JsonProcessingException e) {
17.            e.printStackTrace();
18.        }
19.    }
20. }
```

- lignes 9-11 : sauvegarde de l'état du fragment en session. Lorsque la méthode [saveFragment] est appelée, le fragment est visible. Il faut donc mettre le booléen [hasBeenVisited] à *vrai* (ligne 10) ;

1.23.5 Sauvegarde du fragment [PlaceHolderFragment]

La sauvegarde des fragments se fait dans la méthode [saveFragment] appelée par la classe mère [AbstractFragment] :

```
1. @Override
2. public void saveFragment() {
3.     // on sauvegarde l'état du fragment en session
4.     PlaceHolderFragmentState state = new PlaceHolderFragmentState();
5.     state.setText(textViewInfo.getText().toString());
6.     state.setHasBeenVisited(true);
7.     session.getPlaceHolderFragmentStates()[getArguments().getInt(ARG_SECTION_NUMBER) - 1] = state;
8.     // log
9.     if (isDebugEnabled) {
10.        try {
11.            Log.d(className, String.format("saveFragment state=%s", jsonMapper.writeValueAsString(state)));
12.        }
```

```

12.     } catch (JsonProcessingException e) {
13.         e.printStackTrace();
14.     }
15. }
16.

```

- lignes 4-7 : sauvegarde en session de l'état du fragment ;
- ligne 5 : le texte actuellement affiché par le TextView textViewInfo est sauvegardé ;
- ligne 6 : le booléen [hasBeenVisited] du fragment est passé à *vrai* ;
- ligne 7 : l'état du fragment est mis en session dans le tableau [placeHolderFragmentStates]. Le n° de l'élément à initialiser est le n° de section du fragment moins un ;

1.23.6 Restauration du fragment [Vue1Fragment]

La restauration des fragments se fait dans la méthode [updateFragment] :

```

1.  @Override
2.  protected void updateFragment() {
3.      // log
4.      if (isDebugEnabled) {
5.          Log.d(className, String.format("updateFragment 1 %s - %s", className, getLocalInfos()));
6.      }
7.      // restauration ?
8.      if (fragmentHasToBeInitialized) {
9.          // restauration état
10.         hasBeenVisited = session.getVue1FragmentState().isHasBeenVisited();
11.         fragmentHasToBeInitialized = false;
12.     }
13.     // log
14.     if (isDebugEnabled) {
15.         Log.d(className, String.format("updateFragment 2 %s - %s", className, getLocalInfos()));
16.     }
17.     // navigation ?
18.     boolean navigation = session.getCurrentView() != IMainActivity.FRAGMENTS_COUNT - 1;
19.     if (navigation) {
20.         // incrément n° de visite
21.         numVisit = session.getNumVisit();
22.         numVisit++;
23.         session.setNumVisit(numVisit);
24.         // on affiche le n° de la visite
25.         Toast.makeText(activity, String.format("Visite n° %s", numVisit), Toast.LENGTH_SHORT).show();
26.     }
27.     // changement n° vue courante
28.     session.setCurrentView(IMainActivity.FRAGMENTS_COUNT - 1);
29. }

```

- lignes 8-12 : restauration de l'état du fragment. Le booléen [fragmentHasToBeInitialized] a été initialisé par la classe parent [AbstractFragment]. Lorsqu'il est à vrai, le fragment vient d'être reconstruit et il faut le réinitialiser. C'est ici que ça se passe. Dans cet exemple précis, il n'y a rien à faire. Nous avons simplement montré qu'on pouvait retrouver la valeur du booléen [hasBeenVisited] dans l'état sauvegardé du fragment (ligne 10) ;
- ligne 11 : il ne faut pas oublier de remettre le [fragmentHasToBeInitialized] à *faux*, pour que lorsqu'on reviendra ultérieurement sur ce fragment sans qu'il y ait eu de rotation du périphérique, on ne refasse pas une initialisation inutile du fragment ;
- lignes 18-26 : incrément du compteur de visites. Ici, il y a une difficulté : lorsqu'on fait une restauration du fragment, on ne veut pas incrémenter ce compteur. Il nous faut distinguer ici entre
 - une simple navigation qui ramène l'utilisateur sur l'onglet [Vue 1] ;
 - une restauration lorsque l'utilisateur fait tourner son périphérique lorsque l'onglet [Vue 1] est affiché ;

On distingue ces deux cas, grâce au n° de vue stocké dans la session. Ce n° est celui de la dernière vue affichée (ligne 28).

- ligne 18 : il y a navigation et non pas restauration si le n° de la dernière vue est différent de celui de la vue courante ;
- lignes 21-25 : incrémentation du compteur de visites et son affichage ;

1.23.7 Restauration du fragment [PlaceHolderFragmant]

La restauration des fragments se fait dans la méthode [updateFragment] :

```

1.  // data
2.  private String text;
3.  private int numVisit;
4.  private String newText;
5.  private boolean hasBeenVisited = false;
6.  private ObjectMapper jsonMapper = new ObjectMapper();
7.  ...
8.
9.  public void updateFragment() {

```

```

10.    // log
11.    if (isDebugEnabled) {
12.        Log.d("PlaceholderFragment", String.format("update %s - %s - %s", getArguments().getInt(ARG_SECTION_NUMBER),
13.                className, getLocalInfos()));
14.    }
15.    // de quel fragment s'agit-il ?
16.    int numSection = getArguments().getInt(ARG_SECTION_NUMBER);
17.    int numView = numSection - 1;
18.    // le fragment doit-il être initialisé ?
19.    if (fragmentHasToBeInitialized) {
20.        // texte initial
21.        text = getString(R.string.section_format, numSection);
22.        fragmentHasToBeInitialized = false;
23.    }
24.    // navigation ?
25.    boolean navigation = session.getCurrentView() != numView;
26.    if (navigation) {
27.        // incrément n° de visite
28.        numVisit = session.getNumVisit();
29.        numVisit++;
30.        session.setNumVisit(numVisit);
31.        // texte modifié
32.        newText = String.format("%s, visite %s", text, numVisit);
33.    } else {
34.        // on a affaire à une restauration
35.        PlaceholderFragmentState state = session.getPlaceHolderFragmentStates()[numView];
36.        newText = state.getText();
37.    }
38.    // affichage texte
39.    textViewInfo.setText(newText);
40.    // vue courante
41.    session.setCurrentView(numView);

```

- lignes 15-16 : on détermine le n° de la vue que l'on est en train de mettre à jour ;
- lignes 18-22 : cas où le fragment est dans un cycle sauvegarde / restauration après un changement d'orientation du périphérique. Il faut ici le restaurer. Il s'agit généralement de restaurer certains champs du fragment ;
- ligne 20 : le champ [text] de la ligne 2 doit contenir le texte initial affiché par le fragment : [Hello world from section i]. Il doit ici être régénéré ;
- ligne 21 : on note que l'initialisation du fragment a été faite ;
- lignes 24-36 : comme précédemment pour le fragment [Vue1Fragment], l'incrémentation du compteur de visites ne doit pas se faire lors d'une restauration. Comme précédemment, il nous faut distinguer entre navigation et restauration ;
- lignes 32-36 : cas de la restauration ;
- ligne 34 : l'état du fragment avant la rotation du périphérique est récupéré dans la session ;
- ligne 35 : on y récupère le texte qui était alors affiché ;
- ligne 38 : ce texte est de nouveau affiché ;
- ligne 40 : on note en session, le n° de la nouvelle vue affichée ;

1.23.8 Gestion des onglets

Les paragraphes précédents n'ont pas abordé la gestion des onglets. Or nous avons vu un problème dans l'exemple 21 lors de la rotation du périphérique : seul le 1er onglet [Vue 1] était conservé. Le second onglet lui était perdu.

Nous résolvons ce problème dans la classe [MainActivity] de la façon suivante :

```

1.  @AfterViews
2.  protected void afterViews() {
3.      // log
4.      if (IS_DEBUG_ENABLED) {
5.          Log.d(className, "afterViews");
6.      }
7.      // barre d'outils
8.      Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
9.      setSupportActionBar(toolbar);
10.
11.     ...
12.
13.     // 1er onglet
14.     TabLayout.Tab tab = tabLayout.newTab();
15.     tab.setText("Vue 1");
16.     tabLayout.addTab(tab);
17.     // 2ième onglet ?
18.     int numFragment = session.getNumFragment();
19.     if (numFragment != -1) {
20.         TabLayout.Tab tab2 = tabLayout.newTab();
21.         tab2.setText(String.format("Fragment n° %s", (numFragment + 1)));
22.         tabLayout.addTab(tab2);
23.     }
24.

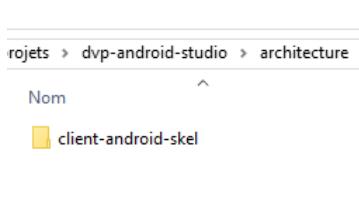
```

```
25.     // quel onglet sélectionner ?
26.     tabLayout.getTabAt(session.getSelectedTab()).select();
27.
28. ...
29.
30. }
```

- lignes 14-16 : création du 1er onglet ;
- lignes 18-23 : création du 2ième onglet. Pour savoir si on doit le créer, on regarde dans la session le n° du fragment affiché dans l'onglet 2. Si ce n° est différent de -1, sa valeur initiale, alors le second onglet est créé. A ce stade, on a deux onglets dont par défaut le 1er est sélectionné ;
- ligne 26 : on va chercher dans la session le n° de l'onglet qui était sélectionné avant la sauvegarde / restauration et on sélectionne celui-ci de nouveau. Si le champ [selectedTab] n'a pas encore été initialisé par le code, c'est sa valeur initiale 0 qui est alors utilisée ;

2 Squelette d'un client Android communiquant avec un service web / JSON

Nous proposons maintenant un squelette d'application Android communiquant avec un ou des services web / JSON. C'est le projet [client-android-skel] qu'on trouvera dans le dossier [architecture] des exemples :



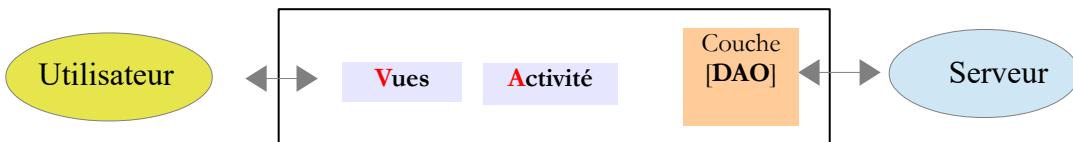
L'étude de cette application squelette va être l'occasion de revoir certains points que nous avons rencontrés dans les exemples précédents. Cette application servira de squelette à toutes les applications à venir de ce document. Elle a été construite après de nombreuses itérations. Elle cherche à factoriser dans des classes abstraites le plus d'éléments possibles des applications à venir pour éviter d'avoir à écrire toujours le même type de code se différenciant par des détails. Ses caractéristiques sont les suivantes :

- la communication asynchrone avec le serveur web / JSON se fait avec la bibliothèque RxJava ;
- le cycle de vie d'un fragment (update, save, restore) est géré par sa classe parent [AbstractFragment] qui appelle à des moments précis certaines méthodes de ses classes filles. La classe fille n'a ainsi pas à se soucier des étapes du cycle de vie mais seulement d'implémenter certaines méthodes imposées par sa classe parent ;
- le cycle de vie de l'activité (save / restore) est géré par une classe abstraite [AbstractActivity] qui elle aussi impose à l'activité fille d'implémenter certaines méthodes ;
- la classe [AbstractActivity] est capable de gérer une application avec ou sans onglets, avec ou sans image d'attente, avec ou sans authentification basique auprès du serveur web / JSON. La présence ou non de ces éléments se fait par configuration ;

Ce squelette a été utilisé pour tous les exemples à venir. A cause de la diversité de ceux-ci, ce qui marchait pour un exemple pouvait ne pas marcher pour l'exemple suivant. Comme le squelette a été utilisé pour sept exemples au total, de nombreuses itérations ont eu lieu. Si on l'utilisait pour un huitième exemple, il est possible qu'on trouverait là encore que la spécificité de ce nouvel exemple génère de nouvelles erreurs. Néanmoins, l'utilisation de ce squelette va considérablement simplifier l'écriture des exemples à venir. En effet, la gestion du cycle de vie d'un fragment (update, save, restore) couplée à la notion d'adjacence des fragments est particulièrement complexe. Ici, elle est totalement cachée dans la classe [AbstractFragment].

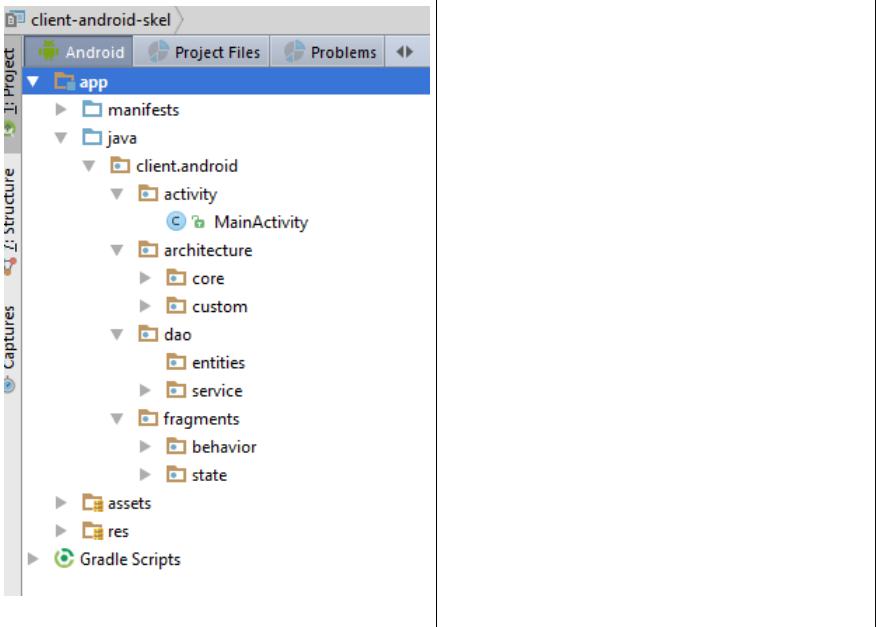
2.1 Architecture du client Android

Le client Android proposé repose sur l'architecture suivante :



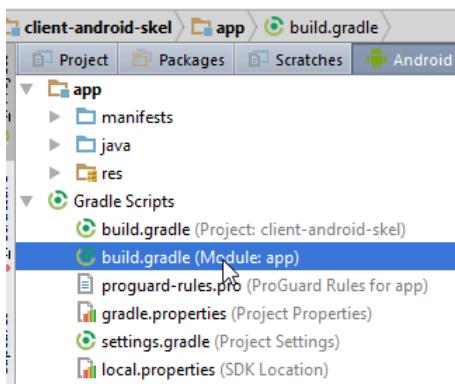
- la couche [DAO] implémente une interface [IDao]. C'est elle qui communique avec le serveur ;
- il n'y a qu'une activité qui implémente également l'interface [IDao]. Les vues s'adressent à elle pour atteindre le serveur ;
- les vues sont implémentées par des fragments ;

Le projet Android reflète cette architecture :



Nous allons présenter un à un les différents éléments de ce projet.

2.2 La configuration Gradle



```

1. buildscript {
2.     repositories {
3.         mavenCentral()
4.     }
5.     dependencies {
6.         // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
7.         classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
8.     }
9. }
10.
11. apply plugin: 'com.android.application'
12. apply plugin: 'android-apt'
13.
14. android {
15.     compileSdkVersion 23
16.     buildToolsVersion "23.0.3"
17.     defaultConfig {
18.         minSdkVersion 15
19.         targetSdkVersion 23
20.         versionCode 1
21.         versionName "1.0"
22.     }
23.
24.     buildTypes {
25.         release {
26.             minifyEnabled false
27.             proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

```

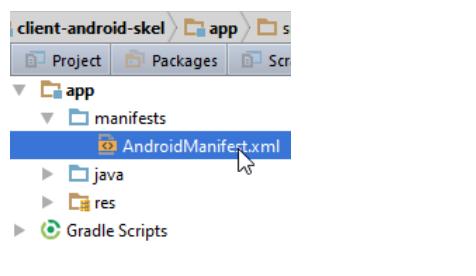
```

28.     }
29.   }
30.
31.   // options de packaging nécessaires pour être capable de produire l'APK
32.   packagingOptions {
33.     exclude 'META-INF/ASL2.0'
34.     exclude 'META-INF/NOTICE'
35.     exclude 'META-INF/LICENSE'
36.     exclude 'META-INF/notice.txt'
37.     exclude 'META-INF/license.txt'
38.   }
39. }
40.
41. def AAVersion = '4.0.0'
42. dependencies {
43.   apt "org.androidannotations:androidannotations:$AAVersion"
44.   compile "org.androidannotations:androidannotations-api:$AAVersion"
45.   apt "org.androidannotations:rest-spring:$AAVersion"
46.   compile "org.androidannotations:rest-spring-api:$AAVersion"
47.   compile 'com.android.support:appcompat-v7:23.4.0'
48.   compile 'com.android.support:design:23.4.0'
49.   compile 'org.springframework.android:spring-android-rest-template:2.0.0.M3'
50.   compile 'com.fasterxml.jackson.core:jackson-databind:2.7.4'
51.   compile 'io.reactivex:rxandroid:1.2.0'
52.   compile fileTree(include: ['*.jar'], dir: 'libs')
53.   testCompile 'junit:junit:4.12'
54. }
55.
56. repositories {
57.   maven {
58.     url 'https://repo.spring.io/libs-milestone'
59.   }
60. }

```

- tous les n°s de version sont sujets à modification. On peut néanmoins partir des n°s actuels si on configure Android Studio pour que ces versions des outils Android (lignes 15-16, 47-48) soient bien présentes (cf paragraphe 6.11, page 484) ;

2.3 Le manifeste de l'application



```

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.           package="client.android">
4.
5.   <uses-permission android:name="android.permission.INTERNET"/>
6.
7.   <application
8.     android:allowBackup="true"
9.     android:icon="@mipmap/ic_launcher"
10.    android:label="@string/app_name"
11.    android:supportsRtl="true"
12.    android:theme="@style/AppTheme">
13.     <activity
14.       android:name=".activity.MainActivity_"
15.       android:label="@string/app_name"
16.       android:windowSoftInputMode="stateHidden"
17.       android:theme="@style/AppTheme.NoActionBar">
18.       <intent-filter>
19.         <action android:name="android.intent.action.MAIN"/>
20.
21.         <category android:name="android.intent.category.LAUNCHER"/>
22.       </intent-filter>
23.     </activity>
24.   </application>
25.
26. </manifest>

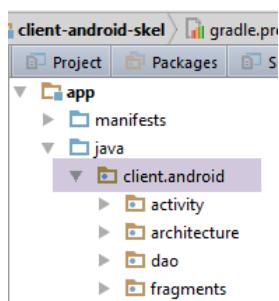
```

- ligne 3 : on changera le package de l'application ;
- lignes 10, 15 : on fixera la valeur de l'item [app_name] dans le fichier [res / values / strings.xml]. Pour l'instant celui-ci est le suivant :

```

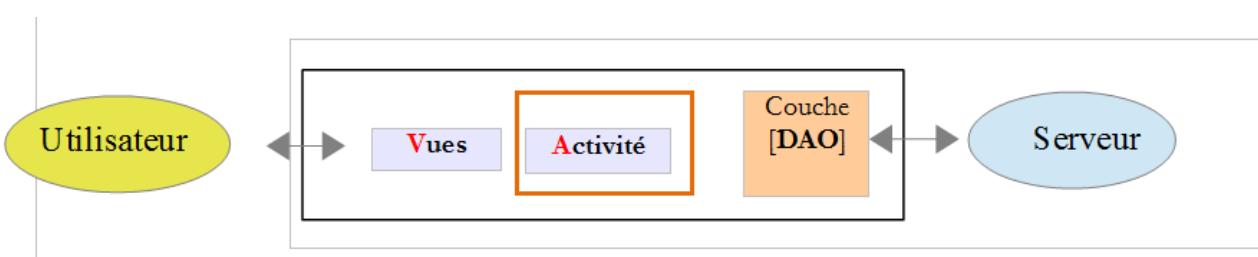
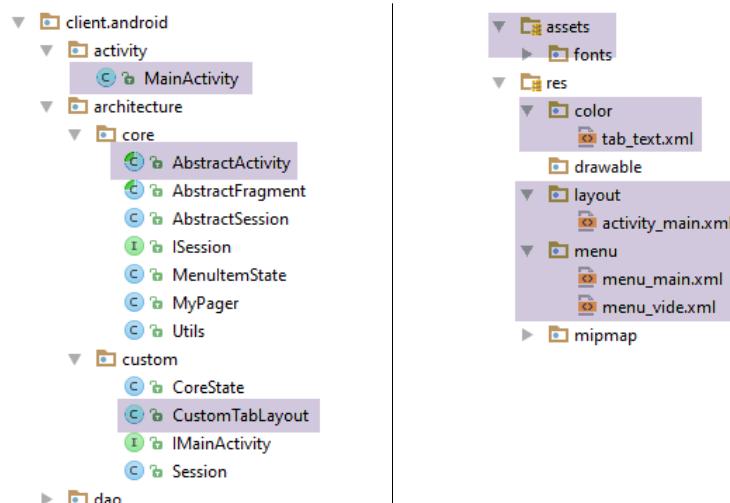
1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.
4.   <!-- nom de l'application -->
5.   <string name="app_name">[Donnez un nom à votre application]</string>
6. </resources>
```

2.4 L'organisation du code Java



- [architecture] regroupe les éléments principaux d'organisation du code ;
- [activity] contient l'activité unique de l'application ;
- [fragments] regroupe les fragments ou vues de l'application ;
- [dao] regroupe les éléments de communication avec le serveur web / JSON ;

2.5 Eléments de l'activité



2.5.1 La vue associée à l'activité

La vue [activity_main.xml] associée à l'activité est la suivante :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     xmlns:app="http://schemas.android.com/apk/res-auto"
5.     android:id="@+id/main_content"
6.     android:layout_width="match_parent"
7.     android:layout_height="match_parent"
8.     android:fitsSystemWindows="true"
9.     tools:context=".activity.MainActivity">
10.
11.    <android.support.design.widget.AppBarLayout
12.        android:id="@+id/appbar"
13.        android:layout_width="match_parent"
14.        android:layout_height="wrap_content"
15.        android:paddingTop="@dimen/appbar_padding_top"
16.        android:theme="@style/AppTheme.AppBarOverlay">
17.
18.        <android.support.v7.widget.Toolbar
19.            android:id="@+id/toolbar"
20.            android:layout_width="match_parent"
21.            android:layout_height="?attr/actionBarSize"
22.            android:background="?attr/colorPrimary"
23.            app:popupTheme="@style/AppTheme.PopupOverlay"
24.            app:layout_scrollFlags="scroll|enterAlways">
25.        </android.support.v7.widget.Toolbar>
26.    </android.support.design.widget.AppBarLayout>
27.
28.    <!-- conteneur de fragments -->
29.    <client.android.architecture.core.MyPagerAdapter
30.        xmlns:android="http://schemas.android.com/apk/res/android"
31.        xmlns:tools="http://schemas.android.com/tools"
32.        android:id="@+id/container"
33.        android:layout_width="match_parent"
34.        android:layout_height="match_parent"
35.        android:paddingLeft="20dp"
36.        android:background="@color/floral_white"/>
37.    </android.support.design.widget.CoordinatorLayout>
```

- ligne 29 : on utilise un conteneur de fragments spécifique ;

L'activité a également un menu [res / menu / menu_main.xml] pour sa vue :

```
1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:app="http://schemas.android.com/apk/res-auto"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     tools:context=".activity.MainActivity">
5. </menu>
```

Pour l'instant, il est vide. Le développeur le complètera si besoin est.

2.5.2 Le conteneur de fragments [MyPagerAdapter]



```
1. package client.android.architecture;
2.
3. import android.content.Context;
4. import android.support.v4.view.ViewPager;
5. import android.util.AttributeSet;
6. import android.view.MotionEvent;
7.
```

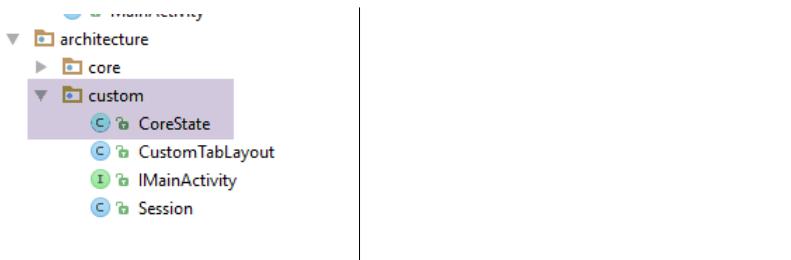
```

8. public class MyPagerAdapter extends ViewPager {
9.
10.    // contrôle le swipe
11.    private boolean isSwipeEnabled;
12.    // contrôle le scrolling
13.    private boolean isScrollingEnabled;
14.
15.    // constructeurs
16.    public MyPagerAdapter(Context context) {
17.        super(context);
18.    }
19.
20.    public MyPagerAdapter(Context context, AttributeSet attrs) {
21.        super(context, attrs);
22.    }
23.
24.    // méthodes à redéfinir pour gérer le swipe
25.    @Override
26.    public boolean onInterceptTouchEvent(MotionEvent event) {
27.        // swipe autorisé ?
28.        if (isSwipeEnabled) {
29.            return super.onInterceptTouchEvent(event);
30.        } else {
31.            return false;
32.        }
33.    }
34.
35.    @Override
36.    public boolean onTouchEvent(MotionEvent event) {
37.        // swipe autorisé ?
38.        if (isSwipeEnabled) {
39.            return super.onTouchEvent(event);
40.        } else {
41.            return false;
42.        }
43.    }
44.
45.    // contrôle du scrolling
46.    @Override
47.    public void setCurrentItem(int position){
48.        super.setCurrentItem(position,isScrollingEnabled);
49.    }
50.
51.    // setters
52.    public void setSwipeEnabled(boolean isSwipeEnabled) {
53.        this.isSwipeEnabled = isSwipeEnabled;
54.    }
55.
56.    public void setScrollingEnabled(boolean scrollingEnabled) {
57.        isScrollingEnabled = scrollingEnabled;
58.    }
59. }
```

Cette classe étend la classe standard Android [ViewPager] uniquement pour gérer le swipe (ligne 11) et le scrolling (ligne 13) entre vues.

- lignes 26-43 : les méthodes qui inhibent le swipe si celui-ci a été désactivé ;
- lignes 46-49 : redéfinition de la méthode [setCurrentItem] qui sert à changer la vue affichée. Si le scrolling a été inhibé, le changement de vue se fera sans scrolling. A noter que le développeur peut contourner ce mode de fonctionnement en utilisant la méthode [setCurrentItem(int position, boolean smoothScrolling)] qui lui permet de préciser le scrolling qu'il désire ;

2.5.3 La classe [CoreState]



La classe [CoreState] est la classe parent des états des différents fragments :

```
1. package client.android.architecture.custom;
```

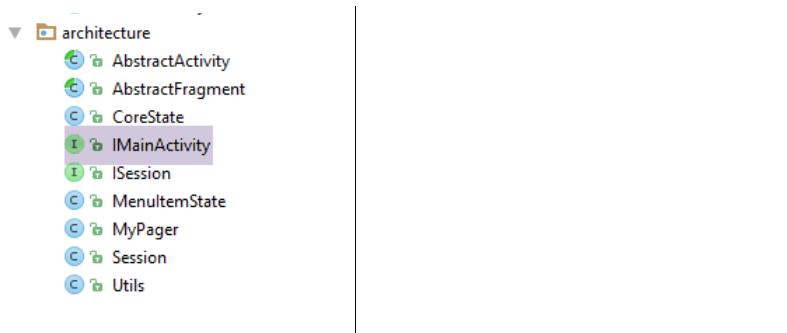
```

2.
3. import client.android.architecture.core.MenuItemState;
4. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
5. import com.fasterxml.jackson.annotation.JsonProperty;
6.
7. @JsonIgnoreProperties(ignoreUnknown = true)
8. @JsonProperty(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
9. // todo : ajouter ici les sous-classes de [CoreState]
10. /*@JsonSubTypes({
11.     @JsonSubTypes.Type(value = Class1.class),
12.     @JsonSubTypes.Type(value = Class2.class)}
13. )*/
14. public class CoreState {
15.     // fragment visité ou non
16.     protected boolean hasBeenVisited = false;
17.     // état de l'éventuel menu du fragment
18.     protected MenuItemState[] menuOptionsState;
19.
20.     // getters et setters
21. ...
22. }

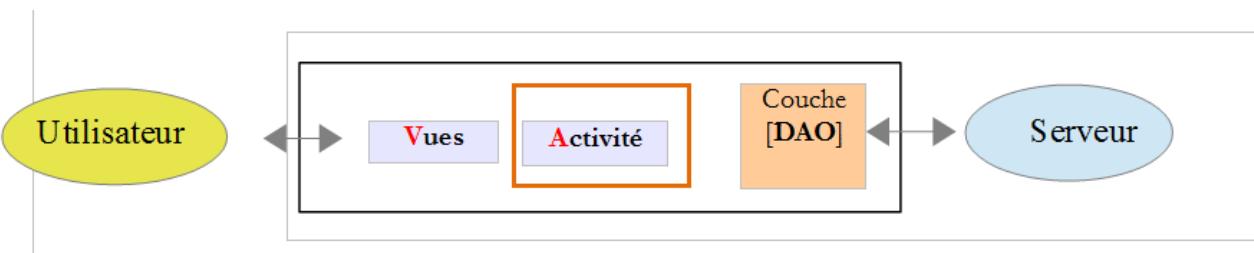
```

- ligne 16 : chaque fragment a dans son état un booléen [hasBeenVisited] qui dit s'il a déjà été visité ou non. Ceci est nécessaire car parfois, lors du 1er affichage d'un fragment, il y a des choses particulière à faire ;
- ligne 18 : le projet [client-android-skel] sauvegarde et restaure automatiquement les menus des fragments s'ils en ont un. Dans le tableau `MenuItemState[] menuOptionsState`, on stocke l'état visible ou non de toutes les options du menu ;
- lignes 10-13 : comme il a été fait dans [Exemple-22], l'état de l'activité et de ses fragments sera sauvegardé dans la session qui elle sera sauvegardée sous la forme d'une chaîne JSON. Nous allons voir que la session mémorise un tableau d'éléments de type [CoreState]. Si on ne fait rien, c'est alors la chaîne JSON d'un type [CoreState] qui sera sauvegardée. Or nous, nous voulons sauvegarder les états des fragments, des états dérivés de [CoreState]. Pour que ce soit la chaîne JSON du type dérivé qui soit produite et non pas celle du type parent, il faut déclarer les types dérivés comme il est indiqué aux lignes 10-13. La classe [CoreState] est l'une des classes de l'architecture que le développeur doit modifier pour chaque nouvelle application ;

2.5.4 L'interface [IMainActivity]



L'interface [IMainActivity] fixe ce que peuvent demander les fragments à l'activité dans l'architecture suivante :



```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.ISession;
4. import client.android.dao.service.IDao;
5.
6. public interface IMainActivity extends IDao {
7.
8.     // accès à la session
9.     ISession getSession();
10.
11.    // changement de vue

```

```

12. void navigateToView(int position, ISession.Action action);
13.
14. // gestion de l'attente
15. void beginWaiting();
16.
17. void cancelWaiting();
18.
19. // constantes de l'application (à modifier) -----
20.
21. // mode debug
22. boolean IS_DEBUG_ENABLED = true;
23.
24. // délai maximal d'attente de la réponse du serveur
25. int TIMEOUT = 1000;
26.
27. // délai d'attente avant exécution de la requête client
28. int DELAY = 0;
29.
30. // authentification basique
31. boolean IS_BASIC_AUTHENTICATION_NEEDED = false;
32.
33. // adjacence des fragments
34. int OFF_SCREEN_PAGE_LIMIT = 1;
35.
36. // barre d'onglets
37. boolean ARE_TABS_NEEDED = false;
38.
39. // image d'attente
40. boolean IS_WAITING_ICON_NEEDED = false;
41.
42. // nombre de fragments de l'application
43. int FRAGMENTS_COUNT = 0;
44.
45. // todo ajoutez ici vos constantes et autres méthodes
46. }

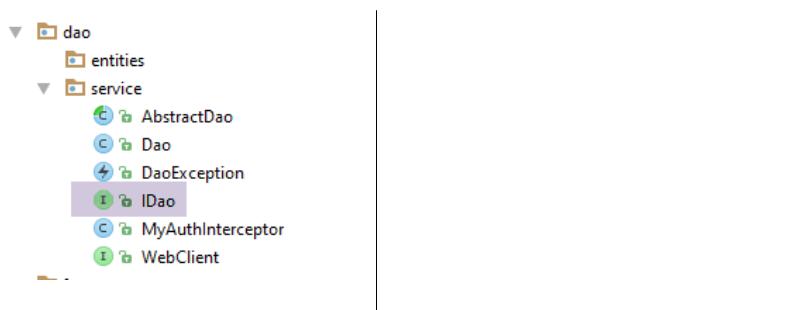
```

- ligne 6 : l'interface [IMainActivity] étend l'interface [IDao] de la couche [DAO] ;
- ligne 9 : c'est l'activité qui donne accès à la session sous la forme d'une instance de l'interface [ISession] ;
- ligne 12 : c'est via l'activité qu'on change de vue. Le second paramètre est l'action qui provoque ce changement de vue, l'une des valeurs SUBMIT, NAVIGATION, RESTORE ;
- lignes 15-17 : c'est l'activité qui gère l'image d'attente ;
- ligne 22 : pour le débogage de l'application ;
- ligne 25 : pour ne pas attendre trop longtemps si le serveur ne répond plus ;
- ligne 28 : en débogage, on mettra une valeur de quelques secondes pour avoir le temps d'annuler l'opération avec le serveur et voir ce qui se passe ;
- ligne 31 : à *true* si le service JSON demande une authentification basique ;
- ligne 34 : adjacence de fragments ;
- ligne 37 : à *vrai* si l'application a des onglets ;
- ligne 39 : à *vrai* si l'application communique avec un serveur web / JSON et qu'on veut montrer une image d'attente lors des échanges ;
- ligne 43 : le nombre de fragments générés par l'application ;

L'interface [IMainActivity] est le second élément de l'architecture que le développeur doit compléter (ligne 45).

2.5.5 L'interface [IDao]

L'interface [IMainActivity] étend l'interface [IDao] suivante :



```

1. package client.android.dao.service;
2.
3. import rx.Observable;
4.

```

```

5. public interface IDao {
6.     // Url du service web
7.     void setUrlServiceWebJson(String url);
8.
9.     // utilisateur
10.    void setUser(String user, String mdp);
11.
12.    // timeout du client
13.    void setTimeout(int timeout);
14.
15.    // authentication basique
16.    void setBasicAuthentification(boolean isBasicAuthentificationNeeded);
17.
18.    // mode debug
19.    void setDebugMode(boolean isDebugEnabled);
20.
21.    // délai d'attente en millisecondes du client avant requête
22.    void setDelay(int delay);
23.
24.    // todo : déclarez votre interface ici
25. }
```

- ligne 24 : le développeur complètera l'interface ici ;

2.5.6 La session



La classe [Session] encapsule les éléments partagés par l'activité et les fragments. Elle implémente l'interface [ISession] suivante :

```

1. package client.android.architecture.core;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. public interface ISession {
6.
7.     // numéro de la dernière vue affichée
8.     int getPreviousView();
9.
10.    void setPreviousView(int numView);
11.
12.    // dernier état d'une vue
13.    CoreState getCoreState(int numView);
14.
15.    void setCoreState(int numView, CoreState coreState);
16.
17.    // action en cours
18.    enum Action {
19.        SUBMIT, NAVIGATION, RESTORE, NONE
20.    }
21.
22.    Action getAction();
23.
24.    void setAction(Action action);
25.
26.    // états de toutes les vues -
27.    // pas utilisé par le code mais est nécessaire pour sérialisation / déserialisation json
28.    CoreState[] getCoreStates();
29.
30.    void setCoreStates(CoreState[] coreStates);
31.
32.    // n° du dernier onglet sélectionné
33.    int getPreviousTab();
34. }
```

```

35.     void setPreviousTab(int position);
36.
37.     // navigation sur sélection onglet
38.     boolean isNavigationOnTabSelectionNeeded();
39.
40.     void setNavigationOnTabSelectionNeeded(boolean navigationOnTabSelection);
41. }

```

Nous introduisons l'interface [ISession] pour imposer la présence de certaines méthodes dans la session :

- lignes 7-10 : le n° de la dernière vue (fragment) affichée ;
- lignes 12-15 : l'état d'une vue particulière ;
- lignes 17-24 : nous introduisons la notion d'action en cours. Il y en a quatre (ligne 17) :
 - RESTORE : une sauvegarde / restauration est en cours. Il n'y a pas de changement de vue ;
 - NAVIGATION : une navigation est en cours. On appellera ici navigation, un changement de vue où la nouvelle vue peut être restaurée à partir de son dernier état stocké en session ;
 - SUBMIT : on donnera le type [SUBMIT] à une action en cours, lorsqu'il y a changement de vue et que la nouvelle vue dépend de l'état de l'activité en général et non de son seul état à elle. Parfois, la distinction entre NAVIGATION et SUBMIT est difficile à faire. Dans ce cas, on prendra le cas le plus général du SUBMIT ;
 - NONE : valeur de l'action lorsque celle-ci n'a pas encore reçu sa première valeur ;
- lignes 26-30 : les états de l'activité et des fragments seront mémorisés dans un tableau de type **CoreState[]**. Pour que celui-ci soit géré correctement lors des sérialisations / déserialisations JSON, il faut qu'il ait un getter et un setter ;
- lignes 32-35 : n° du dernier onglet sélectionné. Est utilisé lors du cycle sauvegarde / restauration pour resélectionner l'onglet qui était sélectionné avant la rotation du périphérique ;
- lignes 37-40 : gestion d'un booléen qui indique si la sélection d'un onglet doit s'accompagner d'un changement de fragment ;

L'interface [ISession] est implémentée par la classe abstraite [AbstractSession] suivante :

```

1. package client.android.architecture.core;
2.
3. import client.android.architecture.custom.CoreState;
4. import client.android.architecture.custom.IMainActivity;
5. import com.fasterxml.jackson.annotation.JsonIgnore;
6.
7. public class AbstractSession implements ISession {
8.     // n° de la vue précédente
9.     private int previousView;
10.
11.    // état des vues
12.    private CoreState[] coreStates = new CoreState[0];
13.
14.    // action en cours
15.    private Action action = Action.NONE;
16.
17.    // onglet sélectionné précédemment
18.    private int previousTab;
19.
20.    // navigation sur sélection onglet
21.    @JsonIgnore
22.    private boolean navigationOnTabSelectionNeeded = true;
23.
24.    // constructeur
25.    public AbstractSession() {
26.        // on initialise le tableau des états des fragments
27.        coreStates = new CoreState[IMainActivity.FRAGMENTS_COUNT];
28.        for (int i = 0; i < coreStates.length; i++) {
29.            coreStates[i] = new CoreState();
30.        }
31.    }
32.
33.
34.    // interface ISession -----
35.    @Override
36.    public int getPreviousView() {
37.        return previousView;
38.    }
39.
40.    @Override
41.    public void setPreviousView(int numView) {
42.        this.previousView = numView;
43.    }
44.
45.    @Override
46.    public CoreState getCoreState(int numView) {
47.        return coreStates[numView];
48.    }
49.

```

```

50.    @Override
51.    public void setCoreState(int numView, CoreState coreState) {
52.        coreStates[numView] = coreState;
53.    }
54.
55.    @Override
56.    public Action getAction() {
57.        return action;
58.    }
59.
60.    @Override
61.    public void setAction(Action action) {
62.        this.action = action;
63.    }
64.
65.    @Override
66.    public CoreState[] getCoreStates() {
67.        return coreStates;
68.    }
69.
70.    @Override
71.    public void setCoreStates(CoreState[] coreStates) {
72.        this.coreStates = coreStates;
73.    }
74.
75.    @Override
76.    public int getPreviousTab() {
77.        return previousTab;
78.    }
79.
80.    @Override
81.    public void setPreviousTab(int position) {
82.        this.previousTab = position;
83.    }
84.
85.    @Override
86.    public boolean isNavigationOnTabSelectionNeeded() {
87.        return navigationOnTabSelectionNeeded;
88.    }
89.
90.    @Override
91.    public void setNavigationOnTabSelectionNeeded(boolean navigationOnTabSelectionNeeded) {
92.        this.navigationOnTabSelectionNeeded = navigationOnTabSelectionNeeded;
93.    }
94. }
```

- ligne 9 : le n° de la vue qui était affichée avant celle actuellement affichée. Cette information est utile lorsqu'on peut arriver sur une vue de plusieurs endroits. C'est typiquement le cas dans une navigation par onglets. La vue affichée peut alors savoir quelle était la vue précédente ;
- ligne 12 : le tableau des états de tous les fragments affichés par l'activité ;
- ligne 18 : le n° de l'onglet précédemment sélectionné. Joue un rôle analogue à celui du n° de la vue précédente de la ligne 9. Cette information est utile lorsqu'il y a rotation du périphérique et qu'on doit se repositionner sur l'onglet qui était sélectionné lors de la rotation ;
- ligne 22 : un booléen indiquant si la sélection d'un onglet doit s'accompagner du changement du fragment affiché. Il faut savoir que le projet [client-android-skel] fait une gestion séparée des onglets et des fragments afin de pouvoir être utilisé dans des cas où le nombre d'onglets est inférieur au nombre de fragments. Il y a deux sortes de sélection :
 - une sélection par l'utilisateur lorsqu'il clique sur un onglet. Dans ce cas, généralement le fragment affiché doit changer ;
 - une sélection logicielle via la méthode [Tablayout.Tab.select()]. Dans ce cas, le changement du fragment affiché n'est pas toujours souhaitable. Voici deux exemples :
 - lors d'une rotation du périphérique, l'activité est recréée et les onglets également. Or lorsque le 1er onglet est créé, il subit automatiquement une opération [select]. Il n'est alors pas souhaitable de changer de fragment affiché car on est dans une phase de recréation de l'activité où le fragment finalement affiché ne sera pas forcément celui associé au 1er onglet ;
 - puisque la gestion des onglets est séparée de celle des fragments, on peut vouloir mettre à jour les onglets (suppression, ajout) sans interférer avec leurs fragments associés. Or certaines de ces opérations peuvent lancer encore déclencher une opération [select] implicite sur l'un des onglets. Cette sélection ne doit pas forcément se traduire par une navigation vers le fragment associé ;
- ligne 21 : le champ [navigationOnTabSelectionNeeded] n'a pas vocation à être sauvegardé lors des opérations de sauvegarde de l'activité et de ses fragments. L'annotation [@JsonIgnore] fait que le champ est ignoré lors des sérialisations / déserialisations JSON ;
- lignes 25-31 : le constructeur initialise le tableau des états des [FRAGMENTS_COUNT] fragments de l'application. Les éléments de ce tableau sont initialisés avec le champ [hasBeeenVisited=false]. Cette information est utilisée pour savoir si on a affaire ou non à la 1ère visite du fragment ;

La classe [Session] est la suivante :

```
1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.AbstractSession;
4.
5. public class Session extends AbstractSession {
6.     // données à partager entre fragments eux-mêmes et entre fragments et activité
7.     // les éléments qui ne peuvent être sérialisés en JSON doivent avoir l'annotation @JsonIgnore
8.     // ne pas oublier les getters et setters nécessaires pour la sérialisation / désérialisation JSON
9. }
```

- ligne 5 : la classe [Session] étend la classe [AbstractSession] que nous venons de voir. Le développeur y placera les éléments à partager entre fragments eux-mêmes et entre fragments et activité. On notera que la classe [Session] n'est plus annotée par l'annotation AA [@EBean]. Elle est devenue une classe normale ;

2.5.7 La classe abstraite [AbstractActivity]



2.5.7.1 Squelette

La classe [AbstractActivity] est une classe de plus de 300 lignes. Nous allons l'étudier par étapes. Son squelette est le suivant :

```
1. package client.android.architecture;
2.
3. import android.os.Bundle;
4. import android.support.design.widget.AppBarLayout;
5. import android.support.design.widget.TabLayout;
6. import android.support.v4.app.FragmentManager;
7. import android.support.v4.app.FragmentPagerAdapter;
8. import android.support.v7.app.AppCompatActivity;
9. import android.support.v7.widget.Toolbar;
10. import android.util.Log;
11. import android.view.View;
12. import android.widget.ProgressBar;
13. import client.android.R;
14. import client.android.dao.service.IDao;
15. import com.fasterxml.jackson.core.JsonProcessingException;
16. import com.fasterxml.jackson.core.type.TypeReference;
17. import com.fasterxml.jackson.databind.ObjectMapper;
18.
19. import java.io.IOException;
20.
21. public abstract class AbstractActivity extends AppCompatActivity implements IMainActivity {
22.     // couche [DAO]
23.     private IDao dao;
24.     // la session
25.     protected Session session;
26.
27.     // le conteneur des fragments
28.     protected MyPagerAdapter mViewPager;
29.     // la barre d'outils
30.     private Toolbar toolbar;
31.     // l'image d'attente
32.     private ProgressBar loadingPanel;
33.     // barre d'onglets
34.     protected TabLayout tabLayout;
35.
36.     // le gestionnaire de fragments ou sections
37.     private FragmentPagerAdapter mSectionsPagerAdapter;
38.     // nom de la classe
39.     protected String className;
```

```

40.    // mappeur JSON
41.    private ObjectMapper jsonMapper;
42.
43.    // constructeur
44.    public AbstractActivity() {
45.        // nom de la classe
46.        className = getClass().getSimpleName();
47.        // log
48.        if (IS_DEBUG_ENABLED) {
49.            Log.d(className, "constructeur");
50.        }
51.        // jsonMapper
52.        jsonMapper = new ObjectMapper();
53.    }
54.
55.    // implémentation IMainActivity -----
56.    ...
57.
58.    // cycle de vie - sauvegarde / restauration de l'activité -----
59.    ...
60.
61.    // gestion de l'image d'attente -----
62.    ...
63.
64.    // interface IDao -----
65.    ...
66.
67.    // le gestionnaire de fragments -----
68.    ...
69.
70.    // classes filles
71.    protected abstract void onCreateActivity();
72.
73.    protected abstract IDao getDao();
74.
75.    protected abstract AbstractFragment[] getFragments();
76.
77.    protected abstract CharSequence getFragmentTitle(int position);
78.
79.    protected abstract void navigateOnTabSelected(int position);
80.
81.    protected abstract int getFirstView();
82.
83. }

```

La classe [AbstractActivity] :

- implémente l'interface [IMainActivity] (lignes 21, 55) ;
- gère la sauvegarde et la restauration de l'activité et de ses fragments lors d'une rotation du périphérique (ligne 58) ;
- gère l'image d'attente lors d'un échange avec le serveur web / JSON (ligne 61) ;
- implémente l'interface IDao de la couche [DAO] (ligne 64) ;
- implémente le gestionnaire de fragments (ligne 67) ;
- impose à ses classes filles la présence de six méthodes (lignes 71-81) ;

2.5.7.2 Implémenter l'interface [IMainActivity]

L'implémentation de l'interface [IMainActivity] (cf paragraphe 2.5.4, page 257) est la suivante :

```

1.    // implémentation IMainActivity -----
2.    @Override
3.    public Session getSession() {
4.        return session;
5.    }
6.
7.    @Override
8.    public void navigateToView(int position, ISession.Action action) {
9.        if (IS_DEBUG_ENABLED) {
10.            Log.d(className, String.format("navigation vers vue %s sur action %s", position, action));
11.        }
12.        // affichage nouveau fragment
13.        mViewPager.setCurrentItem(position);
14.        // on note l'action en cours lors de ce changement de vue
15.        session.setAction(action);
16.    }

```

2.5.7.3 Sauvegarde de l'état de l'activité et de ses fragments

L'état de l'activité et de ses fragments est entièrement dans la session. Il s'agit donc de sauvegarder celle-ci. Nous reprenons ici ce qui a été fait dans le projet [Exemple-22] (cf paragraphe 1.23, page 241) :

```

1.    // gestion sauvegarde / restauration de l'activité -----

```

```

2.     @Override
3.     protected void onSaveInstanceState(Bundle outState) {
4.         // parent
5.         super.onSaveInstanceState(outState);
6.         // sauvegarde session sous la forme d'une chaîne JSON
7.         try {
8.             outState.putString("session", jsonMapper.writeValueAsString(session));
9.         } catch (JsonProcessingException e) {
10.             e.printStackTrace();
11.         }
12.         // log
13.         if (IS_DEBUG_ENABLED) {
14.             try {
15.                 Log.d(className, String.format("onSaveInstanceState session=%s", jsonMapper.writeValueAsString(session)));
16.             } catch (JsonProcessingException e) {
17.                 e.printStackTrace();
18.             }
19.         }
20.     }

```

2.5.7.4 Restauration de l'état de l'activité et de ses fragments

Il s'agit de restaurer la session. Nous faisons comme il a été montré dans [Exemple-22] :

```

1.     @Override
2.     protected void onCreate(Bundle savedInstanceState) {
3.         // parent
4.         super.onCreate(savedInstanceState);
5.         // log
6.         if (IS_DEBUG_ENABLED) {
7.             Log.d(className, "onCreate");
8.         }
9.         // qq chose à restaurer ?
10.        if (savedInstanceState != null) {
11.            // récupération session
12.            try {
13.                session = jsonMapper.readValue(savedInstanceState.getString("session"), new TypeReference<Session>() {
14.                });
15.            } catch (IOException e) {
16.                e.printStackTrace();
17.            }
18.            // log
19.            if (IS_DEBUG_ENABLED) {
20.                try {
21.                    Log.d(className, String.format("onCreate session=%s", jsonMapper.writeValueAsString(session)));
22.                } catch (JsonProcessingException e) {
23.                    e.printStackTrace();
24.                }
25.            }
26.        } else {
27.            // session
28.            session = new Session();
29.        }
30.    ...

```

- lignes 10-26 : si le paramètre [Bundle savedInstanceState] de la ligne 2 est non *null*, alors la session est restaurée (lignes 12-17) ;
- lignes 26-29 : le cas où le paramètre [Bundle savedInstanceState] de la ligne 2 est *null* correspond au 1er démarrage de l'activité. On crée alors une session vide ;

2.5.7.5 Initialisation de la couche [DAO]

```

1.     @Override
2.     protected void onCreate(Bundle savedInstanceState) {
3.         // parent
4.         super.onCreate(savedInstanceState);
5.         // log
6.         if (IS_DEBUG_ENABLED) {
7.             Log.d(className, "onCreate");
8.         }
9.         ...
10.        // couche [DAO]
11.        dao = getDao();
12.        if (dao != null) {
13.            // configuration de la couche [DAO]
14.            setDebugMode(IS_DEBUG_ENABLED);
15.            setTimeout(TIMEOUT);
16.            setDelay(DELAY);
17.            setBasicAuthentification(IS_BASIC_AUTHENTIFICATION_NEEDED);
18.        }
19.        ...
20.    // classes filles

```

```

21.     protected abstract IDao getDao();
22.     ...
23. }

```

- ligne 11 : une référence sur la couche [DAO] est demandée à l'activité fille (ligne 21) ;
- lignes 14-17 : si la couche [DAO] existe, on la configure à partir des informations contenues dans l'interface [IMainActivity] ;

2.5.7.6 Initialisation de la vue associée à l'activité

La vue associée à l'activité a été présentée au paragraphe 2.5.1, page 255 :

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      xmlns:app="http://schemas.android.com/apk/res-auto"
5.      android:id="@+id/main_content"
6.      android:layout_width="match_parent"
7.      android:layout_height="match_parent"
8.      android:fitsSystemWindows="true"
9.      tools:context=".activity.MainActivity">
10.
11.     <android.support.design.widget.AppBarLayout
12.         android:id="@+id/appbar"
13.         android:layout_width="match_parent"
14.         android:layout_height="wrap_content"
15.         android:paddingTop="@dimen/appbar_padding_top"
16.         android:theme="@style/AppTheme.AppBarOverlay">
17.
18.         <android.support.v7.widget.Toolbar
19.             android:id="@+id/toolbar"
20.             android:layout_width="match_parent"
21.             android:layout_height="?attr/actionBarSize"
22.             android:background="?attr/colorPrimary"
23.             app:popupTheme="@style/AppTheme.PopupOverlay"
24.             app:layout_scrollFlags="scroll|enterAlways">
25.             </android.support.v7.widget.Toolbar>
26.         </android.support.design.widget.AppBarLayout>
27.
28.         <!-- conteneur de fragments -->
29.         <client.android.architecture.core.MyPagerAdapter
30.             xmlns:android="http://schemas.android.com/apk/res/android"
31.             xmlns:tools="http://schemas.android.com/tools"
32.             android:id="@+id/container"
33.             android:layout_width="match_parent"
34.             android:layout_height="match_parent"
35.             android:paddingLeft="20dp"
36.             android:background="@color/floral_white"/>
37.     </android.support.design.widget.CoordinatorLayout>

```

Cette vue est initialisée avec le code suivant :

```

1.  @Override
2.  protected void onCreate(Bundle savedInstanceState) {
3.      // parent
4.      super.onCreate(savedInstanceState);
5.      // log
6.      if (IS_DEBUG_ENABLED) {
7.          Log.d(className, "onCreate");
8.      }
9.      ...
10.     // vue associée
11.     setContentView(R.layout.activity_main);
12.     // composants de la vue -----
13.     // barre d'outils
14.     Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
15.     setSupportActionBar(toolbar);
16.     // image d'attente ?
17.     if (IS_WAITING_ICON_NEEDED) {
18.         // on ajoute l'image d'attente
19.         if (IS_DEBUG_ENABLED) {
20.             Log.d(className, "adding loadingPanel");
21.         }
22.         // création ProgressBar
23.         loadingPanel = new ProgressBar(this);
24.         loadingPanel.setVisibility(View.INVISIBLE);
25.         // ajout du ProgressBar à la barre d'outils
26.         toolbar.addView(loadingPanel);
27.     }
28. ...

```

- ligne 11 : la vue XML [activity_main] est associée à l'activité ;

- lignes 14-15 : la barre d'outils est intégrée et supportée ;
- lignes 17-27 : ajout éventuel d'une image d'attente : si le booléen [IS_WAITING_ICON_NEEDED] est à vrai dans l'interface [IMainActivity] ;
- ligne 23 : création de l'image d'attente de type [ProgressBar] référencée par le champ [loadingPanel] ;
- ligne 24 : au départ, cette image est cachée ;
- ligne 26 : elle est ajoutée à la barre d'outils ;

2.5.7.7 Gestion des onglets

L'interface [IMainActivity] peut demander une barre d'onglets. Celle-ci est ajoutée et gérée de la façon suivante :

```

1. // barre d'onglets
2.     protected TabLayout tabLayout;
3. ...
4.
5.     // barre d'onglets ?
6.     if (ARE_TABS_NEEDED) {
7.         // on ajoute la barre d'onglets
8.         if (IS_DEBUG_ENABLED) {
9.             Log.d(className, "adding tablayout");
10.        }
11.        // pas de navigation sur sélection jusqu'à l'affichage d'un fragment
12.        session.setNavigationOnTabSelectionNeeded(false);
13.        // création barre d'onglets
14.        tabLayout = new CustomTabLayout(this);
15.        tabLayout.setTabTextColors(ContextCompat.getColorStateList(this, R.color.tab_text));
16.        // ajout de la barre d'onglets à la barre d'application
17.        AppBarLayout appBarLayout = (AppBarLayout) findViewById(R.id.appbar);
18.        appBarLayout.addView(tabLayout);
19.        // gestionnaire d'évt de la barre d'onglets
20.        tabLayout.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
21.            @Override
22.            public void onTabSelected(TabLayout.Tab tab) {
23.                // un onglet a été sélectionné
24.                if (IS_DEBUG_ENABLED) {
25.                    Log.d(className, String.format("onTabSelected n° %s, action=%s, tabCount=%s isNavigationOnTabSelectionNeeded=%s",
26.                        tab.getPosition(), session.getAction(), tabLayout.getTabCount(),
27.                        session.isNavigationOnTabSelectionNeeded()));
28.                }
29.                if (session.isNavigationOnTabSelectionNeeded()) {
30.                    // position de l'onglet
31.                    int position = tab.getPosition();
32.                    // mémoire
33.                    session.setPreviousTab(position);
34.                    // affichage fragment associé ?
35.                    navigateOnTabSelected(position);
36.                }
37.            }
38.            @Override
39.            public void onTabUnselected(TabLayout.Tab tab) {
40.            }
41.        }
42.        @Override
43.        public void onTabReselected(TabLayout.Tab tab) {
44.        }
45.    });
46. }
47. });
48. }
49. ...
50. ...
51. // classes filles
52. protected abstract void navigateOnTabSelected(int position);
53. ...

```

- lignes 12-48 : ajout et gestion d'une barre d'onglets ;
- ligne 6 : l'ajout de la barre d'onglets se fait si la constante [ARE_TABS_NEEDED] est positionnée à *vrai* dans l'interface [IMainActivity] ;
- ligne 12 : lors de la création de la barre d'onglets, il y a des opérations [TabLayout.Tab.select] implicites qui peuvent survenir (ce n'est pas l'utilisateur qui les provoque). On met le booléen [session.navigationOnTabSelectionNeeded] à *faux* pour éviter toute navigation pendant ces fausses sélections. Ce sera au développeur de sélectionner le fragment à afficher avec la méthode [navigateToView]. Le booléen [session.navigationOnTabSelectionNeeded] sera remis à *vrai* lorsque ce fragment sera affiché (cf classe *AbstractFragment*) ;
- ligne 14 : création d'une barre d'onglets référencée par le champ [tabLayout]. Nous utilisons une barre d'onglets personnalisée [CustomTabLayout] sur laquelle nous allons revenir ;

- ligne 15 : nous fixons les couleurs des titres des onglets. Celles-ci sont trouvées dans le fichier [res / color / tab_txt.xml] suivant :

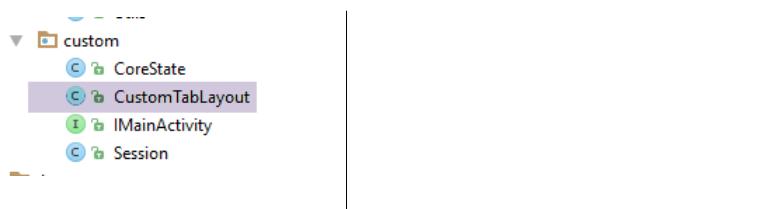
```
a) <?xml version="1.0" encoding="utf-8"?>
b) <selector xmlns:android="http://schemas.android.com/apk/res/android">
c)   <item android:state_selected="true" android:color="#FFFF00" />
d)   <item android:state_selected="false" android:color="#FFFFFF" />
e) </selector>
```

- ligne (c) : la couleur du titre de l'onglet lorsque celui-ci est sélectionné ;
- ligne (d) : la couleur du titre de l'onglet lorsque celui-ci n'est pas sélectionné ;

Ce fichier est bien sûr modifiable. On trouvera les codes hexadécimaux des couleurs par exemple [ici](#).

- lignes 17-18 : ajout de cette barre d'onglets à la barre d'application présente dans la vue XML [activity_main] ;
- lignes 20-47 : gestionnaire des événements de la barre d'onglets ;
- lignes 22-36 : seul l'événement [onTabSelected] est géré. Il correspond à un clic sur l'onglet [Tab tab] passé en paramètre à la méthode ou bien à une opération logicielle [TabLayout.Tab.select] ;
- ligne 30 : position de l'onglet sélectionné ;
- ligne 32 : cette position est mémorisée en session ;
- ligne 34 : il s'agit maintenant d'afficher le fragment associé à cet onglet. Seule la classe fille (ligne 52) peut faire cette association. On notera qu'on n'associe pas la barre d'onglets avec le conteneur de fragments [mViewPager] comme il a été fait dans certains exemples étudiés. Ici, on dissocie totalement la gestion de la barre d'onglets de celle des fragments. C'est pourquoi on est obligé, lorsqu'un onglet est cliqué, d'indiquer quelle vue on veut voir affichée ;
- ligne 28 : on distingue la sélection d'onglet avec ou sans navigation. En général, lorsque l'utilisateur clique sur un onglet, on veut une navigation et lors d'une sélection logicielle on n'en veut pas. C'est le développeur qui distingue ces deux cas avec l'élément [session.navigationOnTabSelectionNeeded]. Lorsque la navigation n'est pas faite, le n° du dernier onglet sélectionné n'est pas enregistré en session. Ce sera au développeur de le faire ;

2.5.7.8 Le gestionnaire d'onglets [CustomTabLayout]



Nous utilisons un gestionnaire d'onglets personnalisé pour pouvoir afficher le titre des onglets avec différentes polices de caractères. La classe [CustomTabLayout] est la suivante :

```
1. package client.android.architecture.custom;
2.
3. import android.content.Context;
4. import android.graphics.Typeface;
5. import android.support.design.widget.TabLayout;
6. import android.util.AttributeSet;
7. import android.view.View;
8. import android.view.ViewGroup;
9. import android.widget.TextView;
10.
11. public class CustomTabLayout extends TabLayout {
12.     private Typeface mTypeface;
13.
14.     public CustomTabLayout(Context context) {
15.         super(context);
16.         init();
17.     }
18.
19.     public CustomTabLayout(Context context, AttributeSet attrs) {
20.         super(context, attrs);
21.         init();
22.     }
23.
24.     public CustomTabLayout(Context context, AttributeSet attrs, int defStyleAttr) {
25.         super(context, attrs, defStyleAttr);
26.         init();
27.     }
28.
29.     private void init() {
30.         mTypeface = Typeface.createFromAsset(getContext().getAssets(), "fonts/Roboto-Bold.ttf");
31.     }
}
```

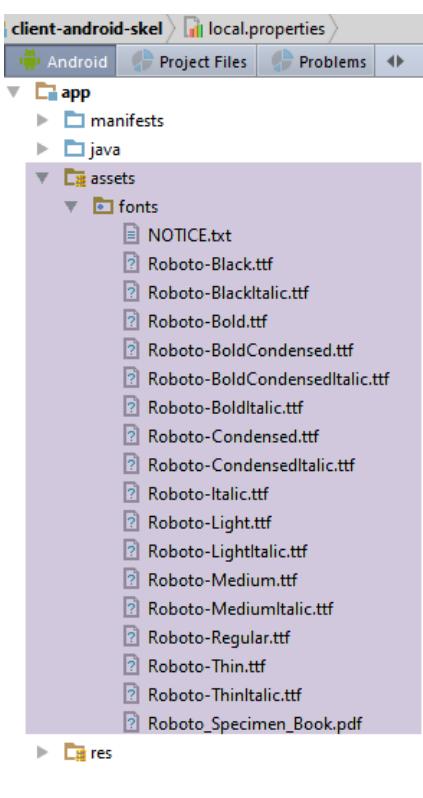
```

32.
33.     @Override
34.     public void addTab(Tab tab) {
35.         super.addTab(tab);
36.
37.         ViewGroup mainView = (ViewGroup) getChildAt(0);
38.         ViewGroup tabView = (ViewGroup) mainView.getChildAt(tab.getPosition());
39.
40.         int tabChildCount = tabView.getChildCount();
41.         for (int i = 0; i < tabChildCount; i++) {
42.             View tabViewChild = tabView.getChildAt(i);
43.             if (tabViewChild instanceof TextView) {
44.                 ((TextView) tabViewChild).setTypeface(mTypeface, Typeface.NORMAL);
45.             }
46.         }
47.     }
48.
49. }

```

- la personnalisation de la police des titres des onglets se fait aux lignes 30 et 44 ;

Le dossier [fonts] est le suivant :



Sources :

- le code de la classe [CustomTabLayout] a été trouvé à l'URL [<http://stackoverflow.com/questions/31067265/change-the-font-of-tab-text-in-android-design-support-tablayout>] ;
- les polices de caractères ont été trouvées à l'URL [<https://www.fontsquirrel.com/fonts/roboto>] ;

2.5.7.9 Dernières initialisations

```

1.     @Override
2.     protected void onCreate(Bundle savedInstanceState) {
3.         // parent
4.         super.onCreate(savedInstanceState);
5.         // log
6.         if (IS_DEBUG_ENABLED) {
7.             Log.d(className, "onCreate");
8.         }
9.         ...
10.        // instanciation du gestionnaire de fragments
11.        mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
12.        // le conteneur de fragments est associé au gestionnaire de fragments

```

```

13.    // c-à-d que le fragment n° i du conteneur de fragments est le fragment n° i délivré par le gestionnaire de fragments
14.    mViewPager = (MyPagerAdapter) findViewById(R.id.container);
15.    mViewPager.setAdapter(mSectionsPagerAdapter);
16.    // on inhibe le swipe entre fragments
17.    mViewPager.setSwipeEnabled(false);
18.    // adjacence des fragments
19.    mViewPager.setOffscreenPageLimit(OFF_SCREEN_PAGE_LIMIT);
20.    // qu'on associe à notre gestionnaire de fragments
21.    mViewPager.setAdapter(mSectionsPagerAdapter);
22.    // on affiche la 1ère vue
23.    if (session.getAction() == ISession.Action.NONE) {
24.        navigateToView(getFirstView(), ISession.Action.NONE);
25.    }
26.    // on passe la main à l'activité fille
27.    onCreateActivity();
28. }
29. ...
30. // classes filles
31. protected abstract void onCreateActivity();
32. protected abstract int getFirstView();
33. ...

```

- lignes 10-21 : on retrouve là du code souvent rencontré dans les exemples étudiés ;
- lignes 23-25 : affichage de la toute première vue. Il y a sans doute plusieurs façons de discriminer ce cas. Ici, nous avons utilisé le fait que pour la toute première vue, la valeur de l'action qui provoque le changement de vue est NONE ;
- ligne 24 : on ne fait pas d'hypothèse sur le 1er fragment à afficher. Dans nos exemples, cela a été souvent le fragment n° 0, mais pas toujours (cf Exemple-22). On demandera donc à l'activité fille (ligne 32) de nous dire quelle est cette première vue ;
- ligne 27 : on a factorisé ici tout ce qu'on pouvait. Maintenant, la classe fille a ses propres initialisations à faire (ligne 31) ;

2.5.7.10 Gestion de l'image d'attente

Dans la classe [AbstractActivity], l'image d'attente est gérée par les deux méthodes suivantes :

```

1.    // gestion de l'image d'attente -----
2.    public void cancelWaiting() {
3.        if (loadingPanel != null) {
4.            loadingPanel.setVisibility(View.INVISIBLE);
5.        }
6.    }
7.
8.    public void beginWaiting() {
9.        if (loadingPanel != null) {
10.            loadingPanel.setVisibility(View.VISIBLE);
11.        }
12.    }

```

2.5.7.11 Implémentation de l'interface [IDao]

Dans la classe [AbstractActivity], l'interface [IDao] (cf paragraphe 2.5.5, page 258) est implémentée de la façon suivante :

```

1.    public abstract class AbstractActivity extends AppCompatActivity implements IMainActivity {
2.        // couche [DAO]
3.        private IDao dao;
4.        ...
5.        // interface IDao -----
6.        @Override
7.        public void setUrlServiceWebJson(String url) {
8.            dao.setUrlServiceWebJson(url);
9.        }
10.
11.        @Override
12.        public void setUser(String user, String mdp) {
13.            dao.setUser(user, mdp);
14.        }
15.
16.        @Override
17.        public void setTimeout(int timeout) {
18.            dao.setTimeout(timeout);
19.        }
20.
21.        @Override
22.        public void setBasicAuthentification(boolean isBasicAuthentificationNeeded) {
23.            dao.setBasicAuthentification(isBasicAuthentificationNeeded);
24.        }
25.
26.        @Override
27.        public void setDebugEnabled(boolean isDebugEnabled) {
28.            dao.setDebugEnabled(isDebugEnabled);
29.        }

```

```

30.
31.     @Override
32.     public void setDelay(int delay) {
33.         dao.setDelay(delay);
34.     }

```

- ligne 3 : on rappelle que la valeur de ce champ a été fournie par l'activité fille dans la méthode [onCreate] ;

2.5.7.12 *Implémentation du gestionnaire de fragments*

Dans la classe [AbstractActivity], le gestionnaire de fragments est implémenté de la façon suivante :

```

1. ...
2.     // le gestionnaire de fragments -----
3.     public class SectionsPagerAdapter extends FragmentPagerAdapter {
4.
5.         AbstractFragment[] fragments;
6.
7.         // constructeur
8.         public SectionsPagerAdapter(FragmentManager fm) {
9.             super(fm);
10.            // fragments de la classe fille
11.            fragments = getFragments();
12.        }
13.
14.        // doit rendre le fragment n° i avec ses éventuels arguments
15.        @Override
16.        public AbstractFragment getItem(int position) {
17.            // on rend le fragment
18.            return fragments[position];
19.        }
20.
21.        // rend le nombre de fragments à gérer
22.        @Override
23.        public int getCount() {
24.            return fragments.length;
25.        }
26.
27.        // rend le titre du fragment n° position
28.        @Override
29.        public CharSequence getPageTitle(int position) {
30.            return getFragmentTitle(position);
31.        }
32.    }
33.
34.    // classes filles
35.    protected abstract AbstractFragment[] getFragments();
36.
37.    protected abstract CharSequence getFragmentTitle(int position);
38.    ...
39. }

```

- ligne 5 : le tableau des fragments associés à l'activité. Tous les fragments seront dérivés de la classe [AbstractFragment] ;
- lignes 8-12 : c'est le constructeur qui initialise le tableau des fragments. Il demande ceux-ci à la classe fille de l'activité (ligne 35) ;
- lignes 28-31 : les titres de fragments peuvent être utilisés dans une application où il y a autant d'onglets que de fragments. Dans ce cas, on peut donner à l'onglet le titre du fragment. Ici, ces titres sont demandés à la classe fille (ligne 37) ;

2.5.7.13 *La méthode [onResume]*

La méthode [onResume] est exécutée un peu avant que la vue associée à l'activité ne soit visible. On l'utilise ici pour sélectionner un onglet après une sauvegarde / restauration :

```

1.     @Override
2.     public void onResume() {
3.         // parent
4.         super.onResume();
5.         if (IS_DEBUG_ENABLED) {
6.             Log.d(className, "onResume");
7.         }
8.         // si restauration, alors il faut restaurer le dernier onglet sélectionné
9.         if (ARE_TABS_NEEDED && session.getAction() == ISession.Action.RESTORE) {
10.             tabLayout.getTabAt(session.getPreviousTab()).select();
11.         }
12.     }

```

- ligne 10 : sélection de l'onglet qui était sélectionné avant le processus de sauvegarde / restauration. Il faut se rappeler ici que dans la méthode [onCreate] qui, dans le cycle de vie de l'activité, est exécutée avant la méthode [onResume], la

navigation sur sélection d'un onglet a été inhibée. Donc ici, il y a sélection d'un onglet mais pas de changement de fragment ;

2.5.7.14 Résumé

La classe abstraite [AbstractActivity] sera la classe parent de l'activité unique de l'application.

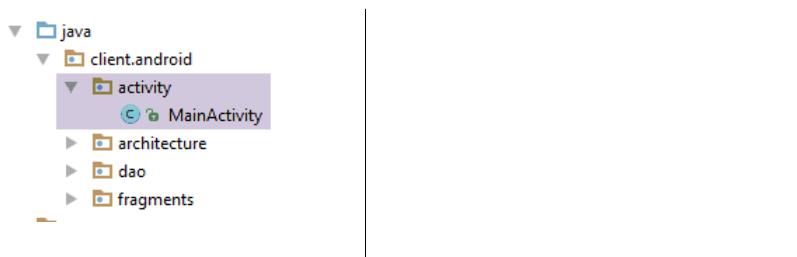
L'activité fille devra implémenter les six méthodes suivantes :

```
1. // classes filles
2. protected abstract void onCreateActivity();
3.
4. protected abstract IDao getDao();
5.
6. protected abstract AbstractFragment[] getFragments();
7.
8. protected abstract CharSequence getFragmentTitle(int position);
9.
10. protected abstract void navigateOnTabSelected(int position);
11.
12. protected abstract int getFirstView();
```

L'activité fille a par ailleurs accès aux éléments protégés de sa classe parent :

```
1. // la session
2. protected ISession session;
3. // le conteneur des fragments
4. protected MyPagerAdapter mViewPager;
5. // barre d'onglets
6. protected CustomTabLayout tabLayout;
7. // nom de la classe
8. protected String className;
```

2.5.8 L'activité [MainActivity]



La classe [MainActivity] peut s'appeler différemment. Sa seule contrainte est d'implémenter l'interface [IMainActivity]. La classe fournie de base est la suivante :

```
1. package client.android.activity;
2.
3. import android.util.Log;
4. import client.android.R;
5. import client.android.architecture.AbstractActivity;
6. import client.android.architecture.AbstractFragment;
7. import client.android.architecture.Session;
8. import client.android.dao.service.Dao;
9. import client.android.dao.service.IDao;
10. import org.androidannotations.annotations.Bean;
11. import org.androidannotations.annotations.EActivity;
12. import org.androidannotations.annotations.OptionsMenu;
13.
14. @EActivity
15. @OptionsMenu(R.menu.menu_main)
16. public class MainActivity extends AbstractActivity {
17.
18.     // couche [DAO]
19.     @Bean(Dao.class)
20.     protected IDao dao;
21.     // session
22.     private Session session;
23.
24.     // méthodes classe parent -----
25.     @Override
26.     protected void onCreateActivity() {
27.         // log
28.         if (IS_DEBUG_ENABLED) {
```

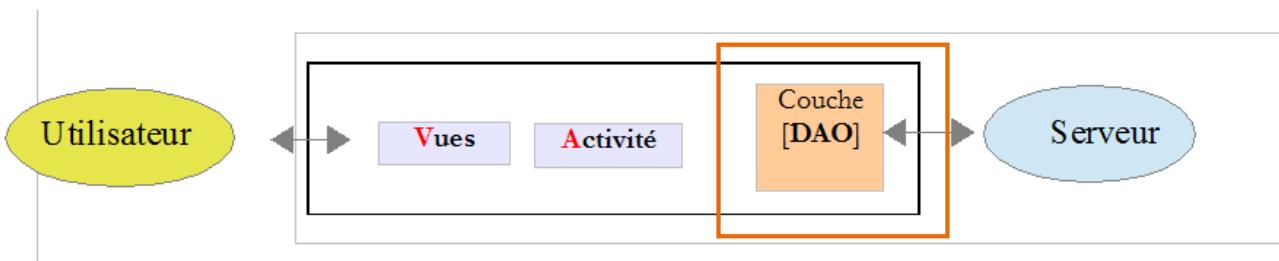
```

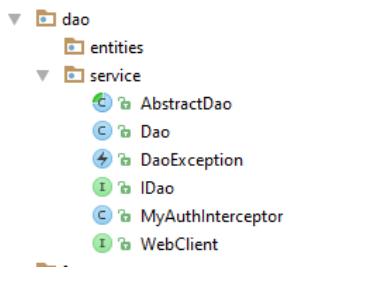
29.     Log.d(className, "onCreateActivity");
30. }
31. // session
32. this.session = (Session) super.session;
33. // todo : on continue les initialisations commencées par la classe parent
34. }
35.
36. @Override
37. protected IDao getDao() {
38.     return dao;
39. }
40.
41. @Override
42. protected AbstractFragment[] getFragments() {
43.     // todo : définir les fragments ici
44.     return new AbstractFragment[0];
45. }
46.
47.
48. @Override
49. protected CharSequence getFragmentTitle(int position) {
50.     // todo : définir les titres des fragments ici
51.     return null;
52. }
53.
54. @Override
55. protected void navigateOnTabSelected(int position) {
56.     // todo : navigation par onglets - définir la vue à afficher
57. }
58.
59. @Override
60. protected int getView() {
61.     // todo : navigation par onglets - définir la première vue à afficher
62.     return 0;
63. }
64. }

```

- ligne 14 : pour que la notation AA [@Bean] de la ligne 19 soit comprise, il faut que l'activité ait la notation AA [@EActivity] ;
- ligne 15 : l'activité est associée au menu XML [menu_main]. Actuellement ce menu est vide. Le développeur aura à le compléter s'il en a besoin ;
- ligne 16 : la classe étend la classe [AbstractActivity] ;
- lignes 19-20 : une référence sur la couche [DAO]. Celle-ci sera instanciée par la bibliothèque AA avant que ce champ ne soit initialisé. Cela entraîne que le bean AA [Dao] doit exister. C'est toujours le cas avec l'application squelette que nous livrons. Même dans une application sans couche [DAO], on peut laisser le package [dao] exister. Cela n'entraîne pas de complications ;
- ligne 22 : la session en tant qu'instance du type [Session]. La session existe dans la classe parent [AbstractActivity] mais en tant qu'instance de l'interface [ISession] (ligne 32) ;
- lignes 24-63 : les six méthodes imposées par la classe parent [AbstractActivity] ;
- lignes 36-39 : la méthode [getDao] rend une référence sur la couche [DAO]. Ici, cette référence n'est jamais *null*. Or dans la classe parent [AbstractActivity], on a prévu le cas où la classe fille rendait une référence *null* pour indiquer qu'il n'y avait pas de couche [DAO]. Si on souhaite user de cette possibilité (pas très utile à mon avis), c'est ici qu'il faut rendre le pointeur *null* ;

2.6 La couche [DAO]





2.6.1 L'interface IDao

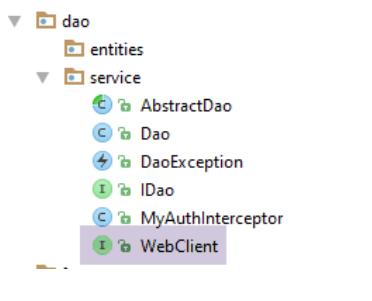
Elle a été présentée au paragraphe 2.5.5, page 258 :

```

1. package client.android.dao.service;
2.
3. import rx.Observable;
4.
5. public interface IDao {
6.     // Url du service web
7.     void setUrlServiceWebJson(String url);
8.
9.     // utilisateur
10.    void setUser(String user, String mdp);
11.
12.    // timeout du client
13.    void setTimeout(int timeout);
14.
15.    // authentification basique
16.    void setBasicAuthentification(boolean isBasicAuthentificationNeeded);
17.
18.    // mode debug
19.    void setDebugMode(boolean isDebugEnabled);
20.
21.    // délai d'attente en millisecondes du client avant requête
22.    void setDelay(int delay);
23.
24.    // todo : déclarez votre interface ici
25. }
  
```

Le développeur ajoutera les méthodes de sa couche [DAO] à partir de la ligne 24.

2.6.2 L'interface [WebClient]



L'interface [WebClient] est la suivante :

```

1. package client.android.dao.service;
2.
3. import org.androidannotations.rest.spring.annotations.Get;
4. import org.androidannotations.rest.spring.annotations.Path;
5. import org.androidannotations.rest.spring.annotations.Rest;
6. import org.androidannotations.rest.spring.api.RestClientRootUrl;
7. import org.androidannotations.rest.spring.api.RestClientSupport;
8. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
9. import org.springframework.web.client.RestTemplate;
10.
11. @Rest(converters = {MappingJackson2HttpMessageConverter.class})
12. public interface WebClient extends RestClientRootUrl, RestClientSupport {
  
```

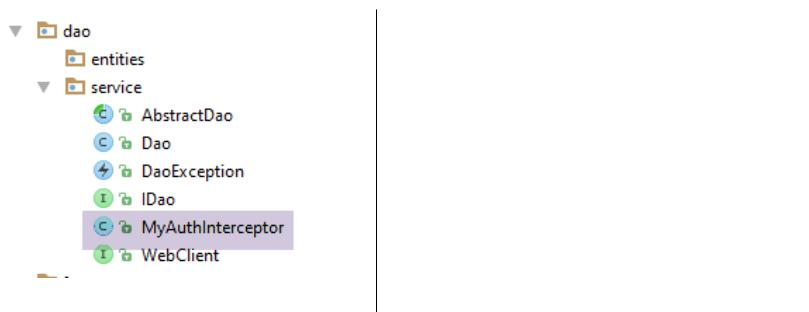
```

13.
14.    // RestTemplate
15.    void setRestTemplate(RestTemplate restTemplate);
16.
17.    // todo : déclarez ici les URL à atteindre
18. }

```

Le développeur ajoutera les méthodes communiquant avec les URL exposées par le serveur JSON à partir de la ligne 17.

2.6.3 L'intercepteur d'authentification[MyAuthInterceptor]



La classe [MyAuthInterceptor] est la suivante :

```

1. package client.android.dao.service;
2.
3. import org.androidannotations.annotations.EBean;
4. import org.springframework.http.HttpAuthentication;
5. import org.springframework.http.HttpBasicAuthentication;
6. import org.springframework.http.HttpHeaders;
7. import org.springframework.http.HttpRequest;
8. import org.springframework.http.client.ClientHttpRequestExecution;
9. import org.springframework.http.client.ClientHttpRequestInterceptor;
10. import org.springframework.http.client.ClientHttpResponse;
11.
12. import java.io.IOException;
13.
14. @EBean(scope = EBean.Scope.Singleton)
15. public class MyAuthInterceptor implements ClientHttpRequestInterceptor {
16.
17.     // utilisateur
18.     private String user;
19.     // mot de passe
20.     private String mdp;
21.
22.     public ClientHttpResponse intercept(HttpRequest request, byte[] body, ClientHttpRequestExecution execution) throws
23.             IOException {
24.         // entêtes HTTP de la requête HTTP interceptée
25.         HttpHeaders headers = request.getHeaders();
26.         // l'entête HTTP d'authentification basique
27.         HttpBasicAuthentication auth = new HttpBasicAuthentication(user, mdp);
28.         // ajout aux entêtes HTTP
29.         headers.setAuthorization(auth);
30.         // on continue le cycle de vie de la requête HTTP
31.         return execution.execute(request, body);
32.     }
33.
34.     // éléments de l'authentification
35.     public void setUser(String user, String mdp) {
36.         this.user = user;
37.         this.mdp = mdp;
38.     }

```

Cette classe génère l'entête HTTP d'authentification suivant :

Authorization: Basic code

où [code] est le code Base64 de la chaîne 'user:mp'. Cette classe ne sert que si le serveur JSON attend cette forme d'authentification. Il en existe d'autres.

Note : l'usage de cette classe est illustrée au paragraphe 3.6.3.1, page 363.

2.6.4 La classe [AbstractDao]



La classe [AbstractDao] est la suivante :

```
1. package client.android.dao.service;
2.
3. import android.util.Log;
4. import client.android.architecture.core.Utils;
5. import com.fasterxml.jackson.core.JsonProcessingException;
6. import com.fasterxml.jackson.databind.ObjectMapper;
7. import rx.Observable;
8. import rx.Subscriber;
9.
10. public abstract class AbstractDao {
11.
12.     // mappeur JSON
13.     private ObjectMapper mapper = new ObjectMapper();
14.     // mode debug
15.     protected boolean isDebugEnabled;
16.     // nom de la classe
17.     protected String className;
18.     // délai d'attente avant exécution requête
19.     private int delay;
20.
21.     // constructeur
22.     public AbstractDao() {
23.         // nom de la classe
24.         className = getClass().getName();
25.         Log.d("AbstractDao", String.format("constructeur, thread=%s", Thread.currentThread().getName()));
26.     }
27.
28.     // méthodes protégées -----
29.     // interface générique
30.     protected interface IRequest<T> {
31.         T getResponse();
32.     }
33.
34.     // requête générique vers un service json
35.     protected <T> Observable<T> getResponse(final IRequest<T> request) {
36.         // log
37.         if (isDebugEnabled) {
38.             Log.d(String.format("%s", className), String.format("delay=%s", delay));
39.         }
40.         // exécution service - on attend une unique réponse
41.         return Observable.create(new Observable.OnSubscribe<T>() {
42.             @Override
43.             public void call(Subscriber<? super T> subscriber) {
44.                 DaoException ex = null;
45.                 // exécution service
46.                 try {
47.                     // attente ?
48.                     if (delay > 0) {
49.                         Thread.sleep(delay);
50.                     }
51.                     // on exécute la requête synchrone
52.                     T response = request.getResponse();
53.                     // log
54.                     if (isDebugEnabled) {
55.                         String log;
56.                         if (response instanceof String) {
57.                             log = (String) response;
58.                         } else {
59.                             log = mapper.writeValueAsString(response);
60.                         }
61.                         Log.d(className, String.format("response=%s sur thread [%s]", log, Thread.currentThread().getName()));
62.                     }
63.                     // on émet la réponse vers l'observateur
64.                     subscriber.onNext(response);
65.                     // on signale la fin de l'observable
```

```

66.         subscriber.onCompleted();
67.     } catch (InterruptedException | JsonProcessingException | RuntimeException e) {
68.         // log
69.         if (isDebugEnabled) {
70.             try {
71.                 Log.d(className, String.format("Thread [%s], Exception communication avec serveur : %s",
72.                     Thread.currentThread().getName(), mapper.writeValueAsString(Utils.getMessagesFromException(e))));
73.             } catch (JsonProcessingException e1) {
74.                 Log.d(className, String.format("Erreur JSON imprévue"));
75.             }
76.         }
77.         // on émet une exception
78.         subscriber.onError(new DaoException(e, 100));
79.     }
80. }
81. }
82.
83. // mode debug
84. public void setDebugMode(boolean isDebugEnabled) {
85.     this.isDebugEnabled = isDebugEnabled;
86. }
87.
88. public void setDelay(int delay) {
89.     this.delay = delay;
90. }
91. }

```

- lignes 35-81 : la méthode [getResponse] utilise la bibliothèque RxAndroid pour rendre un type [Observable<T>]. Contrairement à certains exemples vus précédemment, on ne rend pas un type [Response<T>] qui est un type propriétaire mais un type T quelconque ;
- ligne 35 : la méthode [getResponse] reçoit en paramètre une instance du type [IRequest<T>] des lignes 30-32, dont la méthode [IRequest.getReponse()] obtient le type T par une opération HTTP synchrone ;
- lignes 48-50 : artificiellement on attend [delay] millisecondes. En production on mettra [delay=0]. En phase de débogage on mettra [delay=qqs secondes] pour donner une chance à l'utilisateur d'annuler l'opération asynchrone et ainsi de voir comment se comporte alors le code ;
- ligne 52 : la réponse attendue est demandée avec une requête synchrone ;
- ligne 64 : une fois la réponse reçue, elle est passée à l'observateur ;
- ligne 66 : on indique qu'il n'y aura plus d'émission. On est ici dans le cas particulier d'une action asynchrone qui ne rend qu'un élément ;
- lignes 67-78 : en cas d'exception, on émet l'exception vers l'observateur (ligne 77) ;

2.6.5 La classe [Dao]



La classe [Dao] est la suivante :

```

1. package client.android.dao.service;
2.
3. import android.util.Log;
4. import org.androidannotations.annotations.AfterInject;
5. import org.androidannotations.annotations.Bean;
6. import org.androidannotations.annotations.EBean;
7. import org.androidannotations.rest.spring.annotations.RestService;
8. import org.springframework.http.client.ClientHttpRequestInterceptor;
9. import org.springframework.http.client.SimpleClientHttpRequestFactory;
10. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
11. import org.springframework.web.client.RestTemplate;
12. import rx.Observable;
13.
14. import java.util.ArrayList;
15. import java.util.List;
16.

```

```

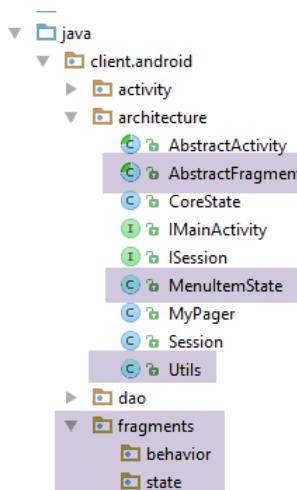
17. @EBean(scope = EBean.Scope.Singleton)
18. public class Dao extends AbstractDao implements IDao {
19.
20.     // client du service web
21.     @RestService
22.     protected WebClient webClient;
23.     // sécurité
24.     @Bean
25.     protected MyAuthInterceptor authInterceptor;
26.     // le RestTemplate
27.     private RestTemplate restTemplate;
28.     // factory du RestTemplate
29.     private SimpleClientHttpRequestFactory factory;
30.
31.     @AfterInject
32.     public void afterInject() {
33.         // log
34.         Log.d(className, "afterInject");
35.         // on construit le restTemplate
36.         factory = new SimpleClientHttpRequestFactory();
37.         restTemplate = new RestTemplate(factory);
38.         // on fixe le convertisseur JSON
39.         restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
40.         // on fixe le restTemplate du client web
41.         webClient.setRestTemplate(restTemplate);
42.     }
43.
44.     @Override
45.     public void setUrlServiceWebJson(String url) {
46.         // on fixe l'URL du service web
47.         webClient.setRootUrl(url);
48.     }
49.
50.     @Override
51.     public void setUser(String user, String mdp) {
52.         // on enregistre l'utilisateur dans l'intercepteur
53.         authInterceptor.setUser(user, mdp);
54.     }
55.
56.     @Override
57.     public void setTimeout(int timeout) {
58.         if (isDebugEnabled) {
59.             Log.d(className, String.format("setTimeout thread=%s, timeout=%s", Thread.currentThread().getName(), timeout));
60.         }
61.         // configuration factory
62.         factory.setReadTimeout(timeout);
63.         factory.setConnectTimeout(timeout);
64.     }
65.
66.     @Override
67.     public void setBasicAuthentification(boolean isBasicAuthentificationNeeded) {
68.         if (isDebugEnabled) {
69.             Log.d(className, String.format("setBasicAuthentification thread=%s, isBasicAuthentificationNeeded=%s",
70. Thread.currentThread().getName(), isBasicAuthentificationNeeded));
71.         }
72.         // intercepteur d'autentification ?
73.         if (isBasicAuthentificationNeeded) {
74.             // on ajoute l'intercepteur d'autentification
75.             List<ClientHttpRequestInterceptor> interceptors = new ArrayList<ClientHttpRequestInterceptor>();
76.             interceptors.add(authInterceptor);
77.             restTemplate.setInterceptors(interceptors);
78.         }
79.
80.         // méthodes privées -----
81.         private void log(String message) {
82.             if (isDebugEnabled) {
83.                 Log.d(className, message);
84.             }
85.         }
86.
87.         // todo : implémentation IDao
88.     }

```

- lignes 21-22 : injection du bean AA [WebClient] qui va assurer les échanges avec le serveur web / JSON ;
- lignes 24-25 : injection de l'intercepteur d'autentification ;
- lignes 31-42 : méthode exécutée après injection des champs des lignes 21-25 ;
- ligne 37 : l'objet [RestTemplate] qui assure les échanges client / serveur est créé à partir d'une *factory*. Ce n'est pas indispensable mais c'est via la *factory* qu'on peut configurer les timeouts des échanges. C'est pourquoi, nous n'utilisons pas le constructeur sans paramètres [RestTemplate()] ;
- ligne 39 : on ajoute un convertisseur JSON aux convertisseurs du [RestTemplate]. Ce sera le seul convertisseur. Aussi lorsqu'une méthode du client [WebClient] va recevoir une chaîne JSON du serveur, celle-ci sera automatiquement déserialisée en l'objet que la méthode doit rendre ;

- ligne 41 : l'objet [RestTemplate] ainsi configuré est passé au client web qui va assurer les échanges client / serveur grâce à lui ;
- lignes 44-48 : on fixe l'URL racine du serveur web / JSON. Toutes les URL déclarées dans la classe [WebClient] sont des URL relatives à cette URL racine ;
- lignes 50-54 : cette méthode permet de préciser le propriétaire de la connexion lorsque celle-ci est contrôlée par une autorisation de type basique (cf paragraphe 2.6.3, page 274) ;
- lignes 56-64 : fixent les *timeouts* des échanges client / serveur. Cela se fait via la *factory* de l'objet [RestTemplate] qui gouverne les échanges ;
- lignes 66-78 : cette méthode permet d'indiquer que le serveur est un serveur protégé par une authentification de type basique ;
- lignes 72-77 : si une authentification de type basique est demandée, l'intercepteur d'authentification injecté ligne 25 est ajouté aux intercepteurs de l'objet [RestTemplate]. Cet intercepteur ajoutera automatiquement à toutes les requêtes du client web, la ligne HTTP d'authentification basique attendue par le serveur ;
- le développeur implémentera l'interface [IDao] à partir de la ligne 87 ;

2.7 Les fragments



2.7.1 La classe [MenuItemState]

La classe [MenuItemState] encapsule l'état d'une option de menu :

```

1. package client.android.architecture;
2.
3. public class MenuItemState {
4.
5.     // identifiant de l'option de menu
6.     private int menuItemId;
7.     // visibilité de l'option
8.     private boolean isVisible;
9.
10.    // constructeurs
11.    public MenuItemState() {
12.
13.    }
14.
15.    public MenuItemState(int menuItemId, boolean isVisible) {
16.        this.menuItemId = menuItemId;
17.        this.isVisible = isVisible;
18.    }
19.
20.    // getters et setters
21.    ...
22. }
```

2.7.2 La classe [Utils]

La classe [Utils] rassemble des méthodes statiques utilitaires :

```

1. package client.android.architecture;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. public class Utils {
7.
8.     // liste de messages d'une exception - version 1
9.     static public List<String> getMessagesFromException(Throwable ex) {
10.         // on crée une liste avec les msg d'erreur de la pile d'exceptions
11.         List<String> messages = new ArrayList<>();
12.         Throwable th = ex;
13.         while (th != null) {
14.             messages.add(th.getMessage());
15.             th = th.getCause();
16.         }
17.         return messages;
18.     }
19.
20.     // liste de messages d'une exception - version 2
21.     static public String getMessageForAlert(Throwable th) {
22.         // on construit le texte à afficher
23.         StringBuilder texte = new StringBuilder();
24.         List<String> messages = getMessagesFromException(th);
25.         int n = messages.size();
26.         for (String message : messages) {
27.             texte.append(String.format("%s : %s\n", n, message));
28.             n--;
29.         }
30.         // résultat
31.         return texte.toString();
32.     }
33.
34.     // liste de messages d'une exception - version 3
35.     static public String getMessageForAlert(List<String> messages) {
36.         // on construit le texte à afficher
37.         StringBuilder texte = new StringBuilder();
38.         int n = messages.size();
39.         for (String message : messages) {
40.             texte.append(String.format("%s : %s\n", n, message));
41.             n--;
42.         }
43.         // résultat
44.         return texte.toString();
45.     }
46. }
```

2.7.3 La classe parent [AbstractFragment]

La classe [AbstractFragment] rassemble ce qui est commun à tous les fragments de l'application. Comme dans la classe [AbstractActivity], son code est complexe. Nous allons là aussi l'analyser par étapes.

2.7.3.1 Le squelette

```

1. package client.android.architecture.core;
2.
3. import android.app.Activity;
4. import android.os.Bundle;
5. import android.support.v4.app.Fragment;
6. import android.util.Log;
7. import android.view.Menu;
8. import android.view.MenuInflater;
9. import android.view.MenuItem;
10. import client.android.architecture.custom.CoreState;
11. import client.android.architecture.custom.IMainActivity;
12. import client.android.architecture.custom.Session;
13. import com.fasterxml.jackson.core.JsonProcessingException;
14. import com.fasterxml.jackson.databind.ObjectMapper;
15. import rx.Observable;
16. import rx.Subscription;
17. import rx.android.schedulers.AndroidSchedulers;
18. import rx.functions.Action0;
19. import rx.functions.Action1;
20. import rx.schedulers.Schedulers;
21.
22. import java.util.ArrayList;
23. import java.util.List;
24.
25. public abstract class AbstractFragment extends Fragment {
26.
27.     // données privées -----
28.     // les abonnements aux observables
29.     private List<Subscription> abonnements = new ArrayList<>();
```

```

30.    // menu du fragment
31.    private Menu menu;
32.    private MenuItemState[] menuOptionsStates = new MenuItemState[0];
33.    // cycle de vie du fragment
34.    private boolean initDone = false;
35.    private boolean isVisibleToUser = false;
36.    private boolean saveFragmentDone = false;
37.    // états du fragment
38.    private CoreState previousState;
39.    // mapeur json
40.    private ObjectMapper jsonMapper = new ObjectMapper();
41.    // cycle de vie du fragment
42.    private boolean fragmentHasToBeInitialized = false;
43.    private boolean viewHasToBeInitialized = false;
44.    // tâches asynchrones
45.    private boolean runningTasksHaveBeenCalled;
46.
47.    // données accessibles aux classes filles -----
48.    // mode debug
49.    final protected boolean isDebugEnabled = IMainActivity.IS_DEBUG_ENABLED;
50.    // nom de la classe
51.    protected String className;
52.    // tâches asynchrones
53.    protected int numberofRunningTasks;
54.    // activité
55.    protected IMainActivity mainActivity;
56.    protected Activity activity;
57.    // session
58.    protected Session session;
59.
60.    // update Fragment -----
61.    ...
62.
63.    // gestion du menu -----
64.    ...
65.    ...
66.
67.    // gestion de l'attente -----
68.    ...
69.
70.    // gestion des opérations asynchrones -----
71.    ...
72.
73.    // gestion exception -----
74.    ....
75.
76.    // gestion du cycle de vie du fragment -----
77.    ...
78.
79.    // classes filles -----
80.    public abstract CoreState saveFragment();
81.
82.    protected abstract int getNumView();
83.
84.    protected abstract void initFragment(CoreState previousState);
85.
86.    protected abstract void initView(CoreState previousState);
87.
88.    protected abstract void updateOnSubmit(CoreState previousState);
89.
90.    protected abstract void updateOnRestore(CoreState previousState);
91.
92.    protected abstract void notifyEndOfUpdates();
93.
94.    protected abstract void notifyEndOfTasks(boolean runningTasksHaveBeenCalled);
95.
96. }

```

- lignes 28-45 : les données privées de la classe ;
- lignes 47-58 : les données protégées accessibles par les classes filles ;
- lignes 61-62 : code qui met à jour le fragment qui va être affiché ;
- lignes 64-65 : code utilitaire pour gérer l'éventuel menu ;
- lignes 67-68 : code utilitaire pour gérer l'attente lors d'une opération asynchrone ;
- lignes 70-71 : code pour faciliter la communication du fragment avec la couche [DAO] ;
- lignes 73-74 : code utilitaire pour gérer toute exception de façon standard ;
- lignes 76-77 : code gérant le cycle de vie du fragment ;
- lignes 80-94 : la classe parent impose 8 méthodes à ses classes filles ;

2.7.3.2 Le constructeur

Le constructeur de la classe est le suivant :

```

1.    // nom de la classe
2.    protected String className;
3.    // cycle de vie du fragment
4.    private boolean fragmentHasToBeInitialized = false;
5.    ...
6.    // constructeur -----
7.    public AbstractFragment() {
8.        // init
9.        className = getClass().getSimpleName();
10.       fragmentHasToBeInitialized = true;
11.       // log
12.       if (isDebugEnabled) {
13.           Log.d(className, "constructeur");
14.       }
15.    }

```

- ligne 9 : on note le nom de la classe fille qui est ici instanciée. Ce nom est utilisé dans tous les logs de la classe parent ;
- ligne 10 : on note que le fragment subit une construction. Cette information sera utilisée lorsqu'il sera demandé au fragment fille de se mettre à jour ;

2.7.3.3 Gestion du menu

Dans notre architecture, **tout fragment doit avoir un menu, même vide**. Les logs ont en effet montré que lorsque la méthode [onCreateOptionsMenu] qui est exécutée lorsque le fragment a un menu, s'exécute, le fragment a déjà été associé à son activité, sa vue et son menu et va devenir visible. C'est donc un moment où la mise à jour de l'interface visuelle et du menu peut être faite. C'est dans cette méthode [onCreateOptionsMenu] que nous demandons au fragment fille de se mettre à jour.

La gestion du menu regroupe des méthodes utilitaires qui permettent au fragment fille d'afficher ou non des éléments du menu :

```

1.    // menu du fragment
2.    private Menu menu;
3.    private MenuItemState[] menuOptionsStates;
4.    ...
5.    // gestion du menu -----
6.    private void getMenuOptions(Menu menu, List<Integer> menuOptionsIds) {
7.        // on parcourt tous les items du menu
8.        for (int i = 0; i < menu.size(); i++) {
9.            // item n° i
10.            MenuItem menuItem = menu.getItem(i);
11.            menuOptionsIds.add(menuItem.getItemId());
12.            // si item n° i est un sous-menu, alors on recommence
13.            if (menuItem.hasSubMenu()) {
14.                // récursivité
15.                getMenuOptions(menuItem.getSubMenu(), menuOptionsIds);
16.            }
17.        }
18.    }
19.
20.   private void getMenuOptionsStates(Menu menu) {
21.       // résultat
22.       if (isDebugEnabled) {
23.           Log.d(className, "getMenuOptionsStates(Menu)");
24.       }
25.       // on récupère les identifiants des options du menu
26.       List<Integer> menuOptionsIds = new ArrayList<>();
27.       getMenuOptions(menu, menuOptionsIds);
28.       // on transfère les options du menu dans un tableau
29.       menuOptionsStates = new MenuItemState[menuOptionsIds.size()];
30.       for (int i = 0; i < menuOptionsStates.length; i++) {
31.           // identifiant option
32.           int id = menuOptionsIds.get(i);
33.           // état option
34.           menuOptionsStates[i] = new MenuItemState(id, menu.findItem(id).isVisible());
35.       }
36.       // résultat
37.       if (isDebugEnabled) {
38.           Log.d(className, String.format("Nombre d'options de menu=%s", menuOptionsStates.length));
39.       }
40.   }
41.
42.   // états des options de menu
43.   private MenuItemState[] getMenuOptionsStates() {
44.       MenuItemState[] menuOptionsStates = new MenuItemState[this.menuOptionsStates.length];
45.       for (int i = 0; i < menuOptionsStates.length; i++) {
46.           // état
47.           MenuItemState state = this.menuOptionsStates[i];
48.           // id du menu
49.           int id = state.getItemId();
50.           // initialisation état
51.           menuOptionsStates[i] = new MenuItemState(id, menu.findItem(id).isVisible());

```

```

52.     }
53.     // résultat
54.     return menuOptionsStates;
55.   }
56.
57.   // affichage options de menu -----
58.   protected void setAllMenuOptionsStates(boolean isVisible) {
59.     // on met à jour toutes les options du menu
60.     for (MenuItemState menuItemState : menuOptionsStates) {
61.       menu.findItem(menuItemState.getMenuItemId()).setVisible(isVisible);
62.     }
63.   }
64.
65.   protected void setMenuOptionsStates(MenuItemState[] menuItemStates) {
66.     // on met à jour certaines options du menu
67.     for (MenuItemState menuItemState : menuItemStates) {
68.       menu.findItem(menuItemState.getMenuItemId()).setVisible(menuItemState.isVisible());
69.     }
70.   }

```

- ligne 6-18 : cette méthode permet d'obtenir les identifiants numériques de toutes les options du menu ;
- ligne 6 : la méthode [getMenuOptions] reçoit deux paramètres :
 - [Menu menu] : le menu du fragment ;
 - [List<Integer> menuOptionsIds] : la liste des identifiants Android des options de menu. Au départ cette liste est vide. Elle est remplie ensuite par un parcours récursif (ligne 15) de l'arbre du menu ;
- lignes 20-40 : à partir du menu, construit le tableau des états (identifiant, visibilité) des options du menu. Ce tableau est stocké en ligne 3. La classe [MenuItemState] a été décrite au paragraphe 2.7.1 ;
- lignes 43-55 : une variante de la méthode précédente. Elle fait la même chose, mais au lieu de recalculer les identifiants de toutes les options du menu, ce qui a déjà été fait, elle utilise les identifiants du tableau des états de la ligne 3 ;
- lignes 58-63 : la méthode [setAllMenuOptionsStates] permet de cacher ou montrer la totalité des options du menu du fragment ;
- lignes 65-69 : la méthode [setMenuOptionsStates] permet, de façon sélective, d'afficher ou cacher certaines des options du menu ;
- les méthodes [getMenuOptions, getMenuOptionsStates] sont déclarées privées car utilisées uniquement dans [AbstractFragment]. Les méthodes [setAllMenuOptionsStates] (ligne 58) et [setMenuOptionsStates] (ligne 65) sont déclarées protégées afin d'être disponibles aux classes filles ;

2.7.3.4 Gestion de l'attente de la fin d'une tâche asynchrone

```

1.   // les abonnements aux observables
2.   private List<Subscription> abonnements = new ArrayList<>();
3.   // tâches asynchrones
4.   protected int numberOfWorkingTasks;
5.   protected boolean tasksInBackgroundHaveBeenCalled;
6.   ...
7.
8.   // gestion de l'attente de la fin d'une opération asynchrone -----
9.   protected void beginWorkingTasks(int numberOfWorkingTasks) {
10.    // on note le nombre de tâches qui vont s'exécuter
11.    this.numberOfWorkingTasks = numberOfWorkingTasks;
12.    // on met l'image d'attente
13.    mainActivity.beginWaiting();
14.    // on vide la liste des abonnements
15.    abonnements.clear();
16.    // pas encore d'annulation
17.    runningTasksHaveBeenCalled = false;
18.  }
19.
20.  protected void cancelWaitingTasks() {
21.    // on cache l'image d'attente
22.    mainActivity.cancelWaiting();
23.  }
24.

```

- lignes 9-18 : pour démarrer une ou des opérations asynchrones, le fragment fille appellera la méthode parent [beginWaitingTasks]. Le paramètre de cette méthode est le nombre de tâches asynchrones que le fragment fille va lancer ;
- ligne 11 : on mémorise le paramètre de la méthode ;
- ligne 13 : l'image d'attente est rendue visible ;
- ligne 15 : on nettoie la liste des abonnements aux opérations asynchrones. Celles-ci n'ont pas encore été créées par le fragment fille ;
- ligne 17 : on entretient un booléen pour signaler que les tâches asynchrones demandées par le fragment fille ont été annulées. Au départ de booléen a la valeur *false* ;
- lignes 20-25 : le fragment fille appelle la méthode parent [cancelWaitingTasks] pour indiquer qu'il veut annuler les tâches qu'il a lancées ;
- ligne 22 : l'image d'attente est cachée ;

2.7.3.5 Gestion des exceptions

```
1. // gestion exception -----
2.
3. // affichage alerte sur exception
4. protected void showAlert(Throwable th) {
5.     // on affiche les messages de la pile d'exceptions du Throwable th
6.     new android.app.AlertDialog.Builder(activity).setTitle("Des erreurs se sont
produites").setMessage(Utils.getMessageForAlert(th)).setNeutralButton("Fermer", null).show();
7. }
8.
9. // affichage liste de messages
10. protected void showAlert(List<String> messages) {
11.     // on affiche la liste des messages
12.     new android.app.AlertDialog.Builder(activity).setTitle("Des erreurs se sont
produites").setMessage(Utils.getMessageForAlert(messages)).setNeutralButton("Fermer", null).show();
13. }
```

- lignes 4-7 : la méthode [showAlert(Throwable)] permet à un fragment fille de faire afficher dans une fenêtre les messages de la pile d'exceptions du *Throwable* passé en paramètre ;
- lignes 10-13 : la méthode [showAlert(List<String>)] permet à un fragment fille de faire afficher dans une fenêtre la liste de messages passée en paramètre ;
- la classe [Utils] utilisée aux lignes 6 et 12 a été décrite au paragraphe 2.7.2, page 278 ;

2.7.3.6 Gestion des opérations asynchrones

```
1. ...
2. // les abonnements aux observables
3. private List<Subscription> abonnements = new ArrayList<>();
4. // tâches asynchrones
5. private boolean runningTasksHaveBeenCalled;
6. protected int numberOfRunningTasks;
7. ...
8. // exécution d'une tâche asynchrone avec RxAndroid
9. protected <T> void executeInBackground(Observable<T> process, Action1<T> consumeResult) {
10.     // process : l'observable à exécuter / observer
11.     // consumeResult : la méthode qui exploite la réponse obtenue
12.     // endOfTask : la méthode à appeler lorsque le processus observé émet la marque de fin d'émission
13.     // on ne crée de nouveaux abonnements que s'il n'y a pas eu annulation
14.     if (!runningTasksHaveBeenCalled) {
15.         // exécution sur thread d'E/S et observation sur thread de l'Ui
16.         process = process.subscribeOn(Schedulers.io()).observeOn(AndroidSchedulers.mainThread());
17.         // on exécute l'observable
18.         try {
19.             abonnements.add(process.subscribe(
20.                 // consommation résultat
21.                 consumeResult,
22.                 // consommation exception
23.                 new Action1<Throwable>() {
24.                     @Override
25.                     public void call(Throwable th) {
26.                         consumeThrowable(th);
27.                     }
28.                 },
29.                 // fin de tâche
30.                 new Action0() {
31.
32.                     @Override
33.                     public void call() {
34.                         endOfTask();
35.                     }
36.                 });
37.             } catch (Throwable th) {
38.                 consumeThrowable(th);
39.             }
40.         }
41.     }
42.     private void endOfTask() {
43.     ...
44.     }
45. }
46.
47. // une opération asynchrone a émis une exception
48. // ou une exception s'est produite pendant l'exécution d'une opération asynchrone
49. private void consumeThrowable(Throwable th) {
50. ...
51. }
52.
```

- lignes 9-41 : exécutent une tâche asynchrone ;
- ligne 9 : la méthode [executeInBackground] attend deux paramètres :

- [Observable<T> process] : le processus asynchrone à exécuter ;
- [Action1<T> consumeResult] : la méthode du fragment fille à appeler pour lui transmettre les éléments émis par le processus. Dans nos exemples précédents, les processus n'ont toujours émis qu'un élément. Le type T de [Action1<T>] est le type T du résultat rendu par le processus observé ;
- ligne 14 : on ne lance la tâche asynchrone que si une annulation par l'utilisateur ou par le programme (à cause d'une exception) n'a pas déjà eu lieu ;
- ligne 16 : le processus est configuré pour s'exécuter sur un thread d'E/S et observé sur le thread de l'Ui ;
- ligne 16 : l'instruction [process.subscribe] lance l'exécution du processus dans le thread d'E/S. A l'intérieur de ce thread, les choses s'exécutent de façon synchrone parce que nous utilisons une bibliothèque H'ITP qui est synchrone ;
- ligne 19 : la méthode [process.subscribe] a trois paramètres :
 - ligne 21 : [consumeResult] : la méthode du fragment fille qui va consommer les éléments émis par le processus ;
 - lignes 22-28 : la méthode exécutée lorsqu'il y a eu une exception pendant le traitement de la tâche asynchrone. Le traitement est délégué à la méthode [consumeThrowable] de la ligne 49 ;
 - lignes 29-36 : la méthode exécutée lorsque la tâche émet la notification de fin d'émission. Le traitement est délégué à la méthode [endOfTask] de la ligne 43 ;
- ligne 19 : la tâche asynchrone qui vient d'être lancée est enregistrée dans le champ [abonnements] qui enregistre toutes les tâches asynchrones lancées. Cela va permettre de les annuler si nécessaire ;
- lignes 37-39 : méthode exécutée lorsqu'il y a eu une exception pendant le traitement de la tâche asynchrone. Le traitement est délégué à la méthode [consumeThrowable] de la ligne 49 ;

La méthode [endOfTask] est la suivante :

```

1.   // tâches asynchrones
2.   protected int numberOfRunningTasks;
3. ...
4.   private void endOfTask() {
5.     // une tâche en moins à attendre
6.     numberOfRunningTasks--;
7.     // fini ?
8.     if (numberOfRunningTasks == 0) {
9.       // fin attente
10.      cancelWaitingTasks();
11.      // on signale la fin des tâches à la classe fille
12.      notifyEndOfTasks(false);
13.    }
14.  }
15. ...
16. // classes filles -----
17. ...
18. protected abstract void notifyEndOfTasks(boolean runningTasksHaveBeenCalled);

```

- ligne 6 : une tâche asynchrone vient de s'achever. On décrémente le compteur des tâches actives ;
- ligne 8 : s'il n'y a plus de tâches actives, alors le fragment fille a obtenu toutes ses réponses ;
- ligne 10 : on annule l'attente ;
- ligne 12 : on signale au fragment fille que toutes les tâches qu'elle a lancées sont terminées en appelant sa méthode [notifyEndOfTasks]. Le paramètre de cette méthode indique comment les tâches se sont terminées, normalement ou sur annulation de l'utilisateur ou du code parce qu'une exception s'est produite. Ligne 12, on signale une fin normale. On notera que le fragment fille n'a pas à se soucier de tenir un compte des tâches encore actives. Sa classe parent le fait pour elle ;

La méthode [consumeThrowable] est la suivante :

```

1.   // tâches asynchrones
2.   protected int numberOfRunningTasks;
3.   private boolean runningTasksHaveBeenCalled;
4. ...
5.   // une opération asynchrone a émis une exception
6.   // ou une exception s'est produite pendant l'exécution d'une opération asynchrone
7.   private void consumeThrowable(Throwable th) {
8.     // th : l'exception à traiter
9.     // cancelWaiting : la méthode à appeler pour signaler qu'il faut arrêter d'attendre
10.    // log
11.    if (isDebugEnabled) {
12.      Log.d(className, "Exception reçue");
13.    }
14.    // on annule les tâches déjà lancées
15.    cancelRunningTasks();
16.    // on affiche les messages d'erreur
17.    showAlert(th);
18.  }
19.
20.  // annulation des tâches
21.  protected void cancelRunningTasks() {
22.    // log

```

```

23.     if (isDebugEnabled) {
24.         Log.d(className, "Annulation des tâches lancées");
25.     }
26.     // on annule toutes les tâches asynchrones enregistrées
27.     for (Subscription abonnement : abonnements) {
28.         abonnement.unsubscribe();
29.     }
30.     // on note l'annulation
31.     runningTasksHaveBeenCalled = true;
32.     numberOfRunningTasks = 0;
33.     // fin de l'attente
34.     cancelWaitingTasks();
35.     // on signale l'annulation des tâches au fragment fille
36.     notifyEndOfTasks(true);
37. }
38.
39. ...
40. // classes filles -----
41. ...
42. protected abstract void notifyEndOfTasks(boolean runningTasksHaveBeenCalled);

```

- ligne 3 : la méthode [consumeThrowable] reçoit l'exception qui s'est produite ;
- ligne 15 : toutes les tâches encore actives sont annulées ;
- ligne 17 : on affiche le texte de l'exception ;
- lignes 21-37 : annulation de toutes les tâches ;
- lignes 27-29 : tous les abonnements sont annulés ;
- ligne 31 : on note qu'il y a eu annulation ;
- ligne 32 : on remet le compteur de tâches à zéro ;
- ligne 34 : l'attente est annulée ;
- ligne 36 : on signale au fragment fille la fin des tâches sur annulation ;

2.7.3.7 Gestion du cycle de vie du fragment

```

1.  // cycle de vie -----
2.  @Override
3.  public void onDestroyView() {
4.      // parent
5.      super.onDestroyView();
6.      // log
7.      if (isDebugEnabled) {
8.          Log.d(className, "onDestroyView");
9.      }
10. }
11.
12. @Override
13. public void onDestroy() {
14.     // parent
15.     super.onDestroy();
16.     // log
17.     if (isDebugEnabled) {
18.         Log.d(className, "onDestroy");
19.     }
20. }
21.
22. @Override
23. public void setUserVisibleHint(boolean isVisibleToUser) {
24. ...
25. }
26.
27. private void saveState() {
28. ...
29. }
30.
31. @Override
32. public void onActivityCreated(Bundle savedInstanceState) {
33. ...
34. }
35.
36.
37. @Override
38. public void onSaveInstanceState(final Bundle outState) {
39. ...
40. }

```

- lignes 2-20 : les méthodes [onDestroyView, onDestroy] ne sont là que pour les logs. Ceux-ci permettent au développeur de mieux appréhender le cycle de vie des fragments ;

La sauvegarde du fragment lors d'une rotation du périphérique est réalisée par les méthodes [setUserVisibleHint, onSaveInstanceState, saveState] suivantes :

```

1. // cycle de vie du fragment
2. private boolean isVisibleToUser = false;
3. private boolean saveFragmentDone = false;
4. ...
5.
6. @Override
7. public void setUserVisibleHint(boolean isVisibleToUser) {
8.     // parent
9.     super.setUserVisibleHint(isVisibleToUser);
10.    // sauvegarde ?
11.    if (this.isVisibleToUser && !isVisibleToUser) {
12.        // le fragment va être caché - on le sauvegarde
13.        if (!saveFragmentDone) {
14.            saveState();
15.        }
16.    }
17.    // mémoire
18.    this.isVisibleToUser = isVisibleToUser;
19. }
20.
21. private void saveState() {
22. ...
23. }
24.
25. @Override
26. public void onSaveInstanceState(Bundle outState) {
27.     // log
28.     if (isDebugEnabled) {
29.         Log.d(className, String.format("onSaveInstanceState isVisibleToUser=%s, saveFragmentDone=%s", isVisibleToUser, saveFragmentDone));
30.     }
31.     // parent
32.     super.onSaveInstanceState(outState);
33.     // sauvegarde du fragment seulement s'il est visible
34.     if (isVisibleToUser) {
35.         // peut-être que la sauvegarde a déjà été faite
36.         if (!saveFragmentDone) {
37.             saveState();
38.         }
39.         // restauration à faire dans tous les cas
40.         session.setAction(ISession.Action.RESTORE);
41.     }
42. }

```

- lignes 6-19 : la sauvegarde du fragment est faite si celui-ci passe de l'état affiché à l'état caché (ligne 11). C'est la méthode [setUserVisibleHint] qui nous donne cette information ;
- ligne 14 : la sauvegarde est faite par la méthode privée des lignes 21-23 ;
- lignes 25-41 : lors d'une rotation du périphérique, la méthode [onSaveInstanceState] va être appelée. Le fragment est sauvegardé à deux conditions :
 - il est visible (ligne 34) ;
 - il n'a pas encore été sauvegardé (ligne 36). Il est possible que les méthodes [setUserVisibleHint, onSaveInstanceState] ne puissent pas s'exécuter toutes deux lorsque le fragment est visible, et que donc la gestion du booléen [saveFragmentDone] soit inutile. Dans le doute, j'ai préféré utiliser celui-ci ;
- ligne 40 : après la sauvegarde viendra la restauration. On note, pour la prochaine fois que le fragment devra se mettre à jour, qu'il devra le faire sur une opération [RESTORE] ;

On notera les deux moments où une sauvegarde du fragment est demandée :

1. lorsque celui-ci passe de l'état visible à l'état caché ;
2. lorsqu'il y a une rotation du périphérique ;

La méthode privée [saveState] est la suivante :

```

1. ...
2. private void saveState() {
3.     // tâches à annuler ?
4.     if (numberOfRunningTasks != 0) {
5.         // on annule les tâches
6.         cancelRunningTasks();
7.     }
8.     // on sauvegarde l'état du fragment
9.     CoreState currentState = saveFragment();
10.    // le fragment a été visité
11.    currentState.setHasBeenVisited(true);
12.    // sauvegarde état du menu
13.    currentState.setMenuOptionsState(getMenuOptionsStates());
14.    // mise en session
15.    session.setCoreState(getNumView(), currentState);
16.    // sauvegarde faite

```

```

17.     saveFragmentDone = true;
18.     // log
19.     if (isDebugEnabled) {
20.         try {
21.             Log.d(className, String.format("saveFragment state=%s", jsonMapper.writeValueAsString(currentState)));
22.         } catch (JsonProcessingException e) {
23.             e.printStackTrace();
24.         }
25.     }
26. }
27.
28.
29. ...
30. // classes filles -----
31. public abstract CoreState saveFragment();
32.
33. protected abstract int getNumView();

```

- ligne 4-7 : la rotation du périphérique peut avoir lieu alors que des opérations asynchrones sont en cours. On prend la décision ici de toutes les annuler. Ce n'est pas une bonne décision pour l'utilisateur qui va devoir refaire une nouvelle requête potentiellement longue alors qu'il a seulement bougé son téléphone ou sa tablette ou bien reçu un appel téléphonique. Il est possible de conserver les connexions réseau au travers d'un cycle sauvegarde / restauration. Simplement les solutions ne sont pas évidentes et j'ai décidé de ne pas les aborder dans ce cours pour débutants. La voie à suivre est de faire ses connexions réseau via un fragment sans interface visuelle attachée et qui n'est pas détruit lors du cycle sauvegarde / restauration. Il suffit pour cela d'utiliser l'instruction [Fragment.setRetainInstance(true)] ;
- ligne 9 : on demande au fragment fille de sauvegarder son état dans un type dérivé de [CoreState] (ligne 31) ;
- ligne 11 : on note que le fragment a été visité. Cette information est utile. Lorsqu'un fragment est visité pour la 1ère fois, sa mise à jour peut être différente des suivantes car alors il n'a pas d'état précédent dans la session ;
- ligne 13 : on sauvegarde l'état du menu ce qui nous permettra de le restaurer automatiquement ;
- ligne 15 : cet état courant est sauvegardé en session. Dans celle-ci, les états sont regroupés par vue / fragment, chacune d'elles ayant un état. Le n° de la vue est fourni par le fragment fille (ligne 33) ;
- ligne 17 : on note que la sauvegarde du fragment a été faite. Ceci parce que deux méthodes sont susceptibles d'appeler la méthode [saveState] et qu'il est inutile de faire deux sauvegardes ;

La régénération de la vue associée au fragment est assurée par la méthode suivante :

```

1.     @Override
2.     public void onActivityCreated(Bundle savedInstanceState) {
3.         // parent
4.         super.onActivityCreated(savedInstanceState);
5.         // log
6.         if (isDebugEnabled) {
7.             Log.d(className, "onActivityCreated");
8.         }
9.         // la vue doit être restaurée
10.        viewHasToBeInitialized = true;
11.    }

```

Dans le cycle de vie, la méthode [onActivityCreated] est exécutée juste après la méthode [onCreateView]. L'appel de cette dernière méthode indique que la vue associée au fragment doit être reconstruite. On se contente de le noter ligne 10.

2.7.3.8 Mise à jour du fragment

La mise à jour du fragment est la dernière opération faite sur le fragment avant qu'il ne soit visible et ne se mette en attente des actions de l'utilisateur. Elle est assurée par le code suivant :

```

1.     // menu du fragment
2.     private Menu menu;
3.     private MenuItemState[] menuOptionsStates;
4.     // cycle de vie du fragment
5.     private boolean initDone = false;
6.     private boolean isVisibleToUser = false;
7.     private boolean saveFragmentDone = false;
8.     // états du fragment
9.     private CoreState previousState;
10.    // mapeur JSON
11.    private ObjectMapper jsonMapper = new ObjectMapper();
12.    // cycle de vie du fragment
13.    private boolean fragmentHasToBeInitialized = false;
14.    private boolean viewHasToBeInitialized = false;
15.    ...
16.
17.    // update Fragment -----
18.    @Override
19.    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
20.        // log

```

```

21.     if (isDebugEnabled) {
22.         Log.d(className, "onCreateOptionsMenu");
23.     }
24.     // mémoire
25.     this.menu = menu;
26.     // on récupère les # options du menu si cela n'a pas déjà été fait
27.     if (fragmentHasToBeInitialized) {
28.         // on récupère les # options du menu
29.         getMenuOptionsStates(menu);
30.         // activité
31.         this.activity = getActivity();
32.         this.mainActivity = (IMainActivity) activity;
33.         this.session = (Session) this.mainActivity.getSession();
34.     }
35.     // on récupère l'état précédent du fragment (la toute 1ère fois, seul le booléen hasBeenVisited représente quelque chose)
36.     previousState = session.getCoreState(getNumView());
37.     // mise à jour du fragment fille en plusieurs étapes
38.     // étape 1 - est-ce la 1ère visite ?
39.     if (!previousState.getHasBeenVisited()) {
40.         if (isDebugEnabled) {
41.             Log.d(className, "initFragment initView updateForFirstVisit");
42.         }
43.         ...
44.     } else {
45.         // ce n'est pas la 1ère visite
46.         // étape 2 : le fragment doit-il être initialisé ?
47.         ...
48.         // étape 3 : la vue doit elle être initialisée ?
49.         ...
50.     }
51.     // étape 4 : un submit, une navigation, un restore ?
52.     ...
53.
54.     // étape 5 : mises à jour terminales -----
55. ...
56. }
57. ...
58. // classes filles -----
59. protected abstract void initFragment(CoreState previousState);
60.
61. protected abstract void initView(CoreState previousState);
62.
63. protected abstract void updateOnSubmit(CoreState previousState);
64.
65. protected abstract void updateOnRestore(CoreState previousState);
66.
67. protected abstract void notifyEndOfUpdates();

```

- ligne 19 : c'est la méthode [onCreateOptionsMenu] qui est utilisée pour mettre à jour le fragment. Pour cette raison, le fragment doit avoir un menu, vide si besoin est. Lorsque cette méthode s'exécute, le fragment a été associé à sa vue et son activité et il est de plus visible ;
- ligne 25 : on mémorise le menu qui a été passé en paramètre (ligne 22) à la méthode ;
- lignes 27-34 : si le fragment doit être initialisé :
 - ligne 29 : les états des options du menu sont mis dans le tableau [menuOptionsStates] de la ligne 3 ;
 - ligne 31 : l'activité est mémorisée comme instance du type Android [Activity] ;
 - ligne 32 : l'activité est mémorisée comme instance de l'interface [IMainActivity] ;
 - ligne 33 : la session est mémorisée. Le changement de type est nécessaire, car la méthode [mainActivity.getSession()] rend un type [ISession] ;
- ligne 36 : on récupère dans la session, l'état précédent du fragment. Si c'est la 1ère visite du fragment, seul le booléen [previousState.hasBeenVisited] a une signification ;
- lignes 39-44 : code exécuté lorsque c'est la 1ère visite faite au fragment. Dans ce cas, son état précédent n'est pas significatif ;
- lignes 44-50 : code exécuté lorsque ce n'est pas la 1ère visite faite au fragment ;
- lignes 46-47 : code exécuté si le constructeur du fragment a été appelé (fragmentHasToBeInitialized==true) ;
- lignes 48-49 : code exécuté si la vue associée au fragment a été reconstruite (viewHasToBeInitialized==true) ;
- lignes 51-52 : code exécuté selon l'action (SUBMIT, NAVIGATION, RESTORE) en cours ;
- lignes 54-55 : code toujours exécuté ;

Les cinq étapes de la mise à jour sont les suivantes :

étape 1

```

1.     // menu du fragment
2.     private Menu menu;
3.     private MenuItemState[] menuOptionsStates;
4.     // cycle de vie du fragment

```

```

5.  private boolean initDone = false;
6.  private boolean isVisibleToUser = false;
7.  private boolean saveFragmentDone = false;
8.  // états du fragment
9.  private CoreState previousState;
10. // mapeur JSON
11. private ObjectMapper jsonMapper = new ObjectMapper();
12. // cycle de vie du fragment
13. private boolean fragmentHasToBeInitialized = false;
14. private boolean viewHasToBeInitialized = false;
15. ...
16.
17.
18.    // on récupère l'état précédent du fragment (la toute 1ère fois, seul le booléen hasBeenVisited représente quelque chose)
19.    previousState = session.getCoreState(getNumView());
20.    // mise à jour du fragment fille en plusieurs étapes
21.    // étape 1 - est-ce la 1ère visite ?
22.    if (!previousState.getHasBeenVisited()) {
23.        if (isDebugEnabled) {
24.            Log.d(className, "initFragment initView updateForFirstVisit");
25.        }
26.        // initialisation fragment et vue
27.        initFragment(null);
28.        initView(null);
29.        // raz previousState pour la suite
30.        previousState = null;
31.    } else {
32.        // ce n'est pas la 1ère visite
33.    ...
34.
35.    protected abstract void initFragment(CoreState previousState);
36.
37.    protected abstract void initView(CoreState previousState);

```

- ligne 19 : l'état précédent du fragment est récupéré dans la session ;
- lignes 22-31 : code exécuté si le fragment n'a jamais été visité ;
- ligne 27 : on demande à la classe fille d'initialiser le fragment. Le paramètre de la méthode [initFragment] de la ligne 35 est l'état précédent du fragment. Ici, on passe `null` pour indiquer au fragment fille que c'est sa 1ère visite ;
- ligne 28 : on demande à la classe fille d'initialiser la vue associée au fragment. Le paramètre de la méthode [initView] de la ligne 37 est l'état précédent du fragment. Ici, on passe `null` pour indiquer au fragment fille que c'est sa 1ère visite ;
- ligne 30 : on met l'état précédent à `null` pour les étapes qui vont suivre ;

étapes 2 et 3

```

1.  // menu du fragment
2.  private Menu menu;
3.  private MenuItemState[] menuOptionsStates;
4.  // cycle de vie du fragment
5.  private boolean initDone = false;
6.  private boolean isVisibleToUser = false;
7.  private boolean saveFragmentDone = false;
8.  // états du fragment
9.  private CoreState previousState;
10. // mapeur JSON
11. private ObjectMapper jsonMapper = new ObjectMapper();
12. // cycle de vie du fragment
13. private boolean fragmentHasToBeInitialized = false;
14. private boolean viewHasToBeInitialized = false;
15. ...
16.
17.
18.    // on récupère l'état précédent du fragment (la toute 1ère fois, seul le booléen hasBeenVisited représente quelque chose)
19.    previousState = session.getCoreState(getNumView());
20.    // mise à jour du fragment fille en plusieurs étapes
21.    // étape 1 - est-ce la 1ère visite ?
22.    if (!previousState.getHasBeenVisited()) {
23.    ...
24.    } else {
25.        // ce n'est pas la 1ère visite
26.        // étape 2 : le fragment doit-il être initialisé ?
27.        if (fragmentHasToBeInitialized) {
28.            if (isDebugEnabled) {
29.                Log.d(className, "initialisation fragment");
30.            }
31.            // fragment fille
32.            initFragment(previousState);
33.        }
34.        // étape 3 : la vue doit elle être initialisée ?
35.        if (viewHasToBeInitialized) {
36.            if (isDebugEnabled) {
37.                Log.d(className, "initialisation vue");

```

```

38.         }
39.         // fragment fille
40.         initView(previousState);
41.     }
42. }
43.
44. ...
45.
46.     protected abstract void initFragment(CoreState previousState);
47.
48.     protected abstract void initView(CoreState previousState);

```

- lignes 24-42 : exécutées lorsque ce n'est pas la 1ère visite du fragment ;
- lignes 27-33 : si le fragment vient d'être reconstruit, on le réinitialise en appelant la méthode [initFragment] de la classe fille (lignes 32, 46). On lui passe l'état précédent du fragment ;
- lignes 35-51 : si la vue associée au fragment doit être initialisée ou réinitialisée, on demande au fragment fille de le faire (lignes 40, 48). Là encore, on lui passe le dernier état connu du fragment ;

étape 4

```

1. // menu du fragment
2. private Menu menu;
3. private MenuItemState[] menuOptionsStates;
4. // cycle de vie du fragment
5. private boolean initDone = false;
6. private boolean isVisibleToUser = false;
7. private boolean saveFragmentDone = false;
8. // états du fragment
9. private CoreState previousState;
10. // mappeur JSON
11. private ObjectMapper jsonMapper = new ObjectMapper();
12. // cycle de vie du fragment
13. private boolean fragmentHasToBeInitialized = false;
14. private boolean viewHasToBeInitialized = false;
15. ...
16.
17.
18.     // on récupère l'état précédent du fragment (la toute 1ère fois, seul le booléen hasBeenVisited représente quelque chose)
19.     previousState = session.getCoreState(getNumView());
20.     // mise à jour du fragment fille en plusieurs étapes
21. ...
22.
23.     // étape 4 : un submit, une navigation, un restore ?
24.     // log
25.     if (isDebugEnabled) {
26.         try {
27.             Log.d(className, String.format("session=%s", jsonMapper.writeValueAsString(session)));
28.             Log.d(className, String.format("état précédent=%s", jsonMapper.writeValueAsString(previousState)));
29.         } catch (JsonProcessingException e) {
30.             e.printStackTrace();
31.         }
32.     }
33.     // action en cours
34.     ISession.Action action = session.getAction();
35.     switch (action) {
36.         case SUBMIT:
37.             if (isDebugEnabled) {
38.                 Log.d(className, "updateOnSubmit");
39.             }
40.             // fragment fille
41.             updateOnSubmit(previousState);
42.             break;
43.         case NAVIGATION:
44.             if (isDebugEnabled) {
45.                 Log.d(className, "updateForNavigation");
46.             }
47.             if (previousState != null) {
48.                 // restauration menu
49.                 setMenuOptionsStates(previousState getMenuOptionsState());
50.                 // fragment fille
51.                 updateOnRestore(previousState);
52.             } else {
53.                 // il s'agit d'une 1ère visite - rien à faire
54.             }
55.             break;
56.         case RESTORE:
57.             // restauration
58.             if (isDebugEnabled) {
59.                 Log.d(className, "updateOnRestore");
60.             }
61.             // restauration menu (previousState ne peut être null)
62.             setMenuOptionsStates(previousState getMenuOptionsState());
63.             // fragment fille

```

```

64.         updateOnRestore(previousState);
65.     }
66. }
67. ....
68. protected abstract void updateOnSubmit(CoreState previousState);
69.
70. protected abstract void updateOnRestore(CoreState previousState);

```

- lignes 34-66 : on traite l'action en cours qui peut être l'une des trois suivantes :
 - RESTORE : on est en train de faire une restauration du fragment après une rotation du périphérique ;
 - NAVIGATION : on revient vers le fragment en voulant le retrouver dans l'état où on l'a laissé la dernière fois qu'on l'a utilisé ;
 - SUBMIT : tous les autres cas ;
- ligne 34 : on récupère l'action en cours ;
- lignes 36-42 : pour une action de type SUBMIT, on appelle la méthode [updateOnSubmit] du fragment fille (lignes 41, 68) en lui passant le dernier état connu du fragment ;
- lignes 43-55 : pour une action de type NAVIGATION ;
- lignes 47-54 : nous voulons remettre le fragment dans son dernier état connu. L'opération de NAVIGATION peut se conjuguer avec une 1ère visite. Ce serait le cas par exemple dans une application à onglets : si je passe de l'onglet 1 à l'onglet 4 :
 - je dois initialiser le fragment de l'onglet 4 s'il s'agit de la 1ère visite ;
 - rétablir le fragment de l'onglet 4 dans son état précédent s'il ne s'agit pas de la 1ère visite ;
- lignes 52-54 : on ne fait rien s'il s'agit d'une 1ère visite. Ce sera à la méthode fille [initView(CoreState previousState)] de faire cette initialisation. La 1ère visite est caractérisée par la condition [previousState==null] ;
- ligne 49 : si ce n'est pas la 1ère visite du fragment, on lui restitue son menu ;
- ligne 51 : on demande à la classe fille de se mettre à jour en appelant la méthode de la ligne 70. On lui passe l'état précédent du fragment pour qu'elle puisse faire son travail ;
- lignes 56-66 : dans le cas d'une opération de restauration du fragment, on fait la même chose que dans le cas d'une navigation hors 1ère visite ;

étape 5

```

1. // menu du fragment
2. private Menu menu;
3. private MenuItemState[] menuOptionsStates;
4. // cycle de vie du fragment
5. private boolean initDone = false;
6. private boolean isVisibleToUser = false;
7. private boolean saveFragmentDone = false;
8. // états du fragment
9. private CoreState previousState;
10. // mappage json
11. private ObjectMapper jsonMapper = new ObjectMapper();
12. // cycle de vie du fragment
13. private boolean fragmentHasToBeInitialized = false;
14. private boolean viewHasToBeInitialized = false;
15. ...
16.
17.
18. // étape 5 : mise à jour terminale -----
19. // on a changé de vue
20. session.setPreviousView(getNumView());
21. // plus d'action en cours
22. session.setAction(ISession.Action.NONE);
23. // lorsqu'on quittera ce fragment, il devra être sauvegardé
24. saveFragmentDone = false;
25. // tant que le fragment n'est pas reconstruit, il n'a pas à être initialisé
26. fragmentHasToBeInitialized = false;
27. // tant que la vue n'est pas reconstruite, elle n'a pas à être initialisée
28. viewHasToBeInitialized = false;
29. // on revient à un fonctionnement normal de la sélection d'onglets
30. session.setNavigationOnTabSelectionNeeded(true);
31.
32. // on signale au fragment que la vue est prête
33. if (isDebugEnabled) {
34.     Log.d(className, "notifyEndOfUpdates");
35. }
36. notifyEndOfUpdates();
37. ...
38. protected abstract void notifyEndOfUpdates();

```

- lignes 18-30 : lorsqu'on arrive ici, le fragment a été initialisé et est prêt à être affiché. On remet alors tous les indicateurs utilisés dans la gestion de vie du fragment dans un état initial ;
- ligne 20 : on a changé de vue : on le note dans la session ;
- ligne 22 : il n'y a plus d'action en cours ;

- ligne 24 : lorsqu'on va quitter le fragment maintenant affiché, il faudra le sauvegarder lorsqu'on le quittera ;
- ligne 26 : le fragment n'a plus besoin d'être reconstruit. Cet indicateur sera remis à *vrai* lorsque le constructeur du fragment sera de nouveau exécuté ;
- ligne 28 : la vue associée au fragment n'a plus besoin d'être initialisée. Cet indicateur sera remis à *vrai* lorsque la méthode [onActivityCreated] sera de nouveau exécutée ;
- ligne 30 : le fragment est affiché peut-être dans une application à onglets. Dans ce cas, lorsque l'utilisateur va cliquer sur l'un d'eux, un changement de fragment doit s'opérer ;
- ligne 36 : on indique à la classe fille que le fragment est prêt. Celle-ci peut mettre dans la méthode [notifyEndOfUpdates] des mises à jour qui seraient à faire dans tous les cas, lancer une opération asynchrone pour obtenir de nouvelles données, ...

2.7.4 Un exemple de fragment



On a mis dans le projet [client-android-skel] un exemple de fragment pour montrer au lecteur la structure typique d'un fragment d'une application basée sur ce projet.

La classe [DummyFragment] est la suivante :

```

1. package client.android.fragments.behavior;
2.
3. import client.android.architecture.core.AbstractFragment;
4. import client.android.architecture.custom.CoreState;
5. import client.android.fragments.state.DummyFragmentState;
6.
7. public class DummyFragment extends AbstractFragment {
8.
9.     // champs hérités de la classe parent -----
10.
11.    // mode debug
12.    //--- final protected boolean isDebugEnabled = IMainActivity.IS_DEBUG_ENABLED;
13.    // nom de la classe
14.    //--- protected String className;
15.    // tâches asynchrones
16.    //--- protected int numberofRunningTasks;
17.    // activité
18.    //--- protected IMainActivity mainActivity;
19.    //--- protected Activity activity;
20.    // session
21.    //--- protected Session session;
22.
23.    // méthodes héritées de la classe parent -----
24.
25.    // affichage options de menu
26.    //--- protected void setAllOptionsMenuStates(boolean isVisible) {
27.    //--- protected void setMenuOptionsStates(MenuItemsState[] menuItemsStates) {
28.    // gestion de l'attente de la fin d'une série de tâches asynchrones
29.    //--- protected void beginRunningTasks(int numberofRunningTasks) {
30.    //--- protected void cancelWaitingTasks() {
31.    // exécution d'une tâche asynchrone avec RxAndroid
32.    //--- protected <T> void executeInBackground(Observable<T> process, Action1<T> consumeResult) {
33.    // annulation des tâches
34.    //--- protected void cancelRunningTasks() {
35.    // affichage alerte sur exception
36.    //--- protected void showAlert(Throwable th) {
37.    // affichage liste de messages

```

```

38.    //--- protected void showAlert(List<String> messages) {
39.    // méthodes imposées par la classe parent -----
40.
41.    @Override
42.    public CoreState saveFragment() {
43.        // il faut sauvegarder le fragment
44.        DummyFragmentState state=new DummyFragmentState();
45.        // ...
46.        return state;
47.        // s'il n'y a rien à sauvegarder faire [return new CoreState();] et supprimer la classe [DummyFragmentState]
48.    }
49.
50.
51.    @Override
52.    protected int getNumView() {
53.        // il faut retourner le n° du fragment dans le tableau des fragments générés par l'activité (cf MainActivity)
54.        return 0;
55.    }
56.
57.    @Override
58.    protected void initFragment(CoreState previousState) {
59.        // le fragment devient visible et a subi une construction dans cette étape ou une étape précédente
60.        // cela se produit au démarrage de l'application et à chaque rotation du périphérique Android
61.        // est forcément suivie par l'exécution de [initFragment]
62.        // il faut initialiser les champs du fragment qui a été reconstruit
63.        // previousState est la dernière sauvegarde du fragment - vaut null si c'est la 1ère visite du fragment
64.    }
65.
66.    @Override
67.    protected void initView(CoreState previousState) {
68.        // le fragment devient visible et la vue associée a été reconstruite dans cette étape ou une étape précédente
69.        // cela se produit à chaque fois que [initFragment] est exécutée et à chaque fois que le fragment sort de l'adjacence
70.        // du fragment affiché
71.        // il faut initialiser les composants de la vue qui a été reconstruite
72.        // previousState est la dernière sauvegarde du fragment - vaut null si c'est la 1ère visite du fragment
73.    }
74.
75.    @Override
76.    protected void updateOnSubmit(CoreState previousState) {
77.        // est exécutée après [initFragment, initView] si ces méthodes sont exécutées
78.        // la vue va être affichée après une opération de type SUBMIT
79.        // il faut en général initialiser fragment et vue associée à partir de la session
80.        // previousState est la dernière sauvegarde du fragment - vaut null si c'est la 1ère visite du fragment
81.        // il n'y a rien à faire si on ne peut arriver au fragment par une opération SUBMIT
82.        // si on peut arriver sur le fragment par plusieurs opérations SUBMIT à partir de fragments différents, on peut
83.        // connaître la vue précédente par [session.getPreviousView]
84.        // si on peut arriver sur le fragment par plusieurs opérations SUBMIT à partir du même fragment, alors il faut mettre
85.        // en session un indicateur pour différencier
86.        // les différents types de SUBMIT à partir de ce fragment
87.    }
88.
89.    @Override
90.    protected void updateOnRestore(CoreState previousState) {
91.        // est exécutée après [initFragment, initView] si ces méthodes sont exécutées
92.        // la vue va être affichée après une opération de type RESTORE ou NAVIGATION
93.        // previousState est la dernière sauvegarde du fragment - ne vaut jamais null
94.        // il faut remettre la vue dans son état précédent
95.
96.    @Override
97.    protected void notifyEndOfUpdates() {
98.        // intervient après les méthodes [updateOnSubmit, updateOnRestore]
99.        // lorsqu'on est là, la vue a été construite et initialisée
100.       // il n'y a souvent rien à faire ici mais on peut aussi factoriser ici des actions qu'il faudrait faire quelque soit la
101.       // façon dont on arrive sur cette vue
102.    }
103.
104.   @Override
105.   protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
106.       // appelée lorsque les tâches asynchrones lancées par le fragment sont soit terminées soit annulées
107.       // ces deux cas peuvent être différenciés grâce au paramètre runningTasksHaveBeenCalled
108.       // il faut en général remettre la vue dans un état différent de celui qu'elle avait pendant qu'elle attendait les
109.       // réponses des tâches asynchrones
110.   }

```

La classe [DummyFragment] peut ne pas avoir d'état. Ici, on en a mis un pour rappeler ce qui est attendu dedans :

```

1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. public class DummyFragmentState extends CoreState {

```

```

6.    // état du fragment [DummyFragment]
7.    // ne mettre que des champs sérialisables en JSON
8.    // mettre l'annotation @JsonIgnore sur les autres mais on voit mal à quoi ils pourraient servir
9.    // ne pas oublier les getters / setters - ils sont utilisés pour la sérialisation / désérialisation
10. }

```

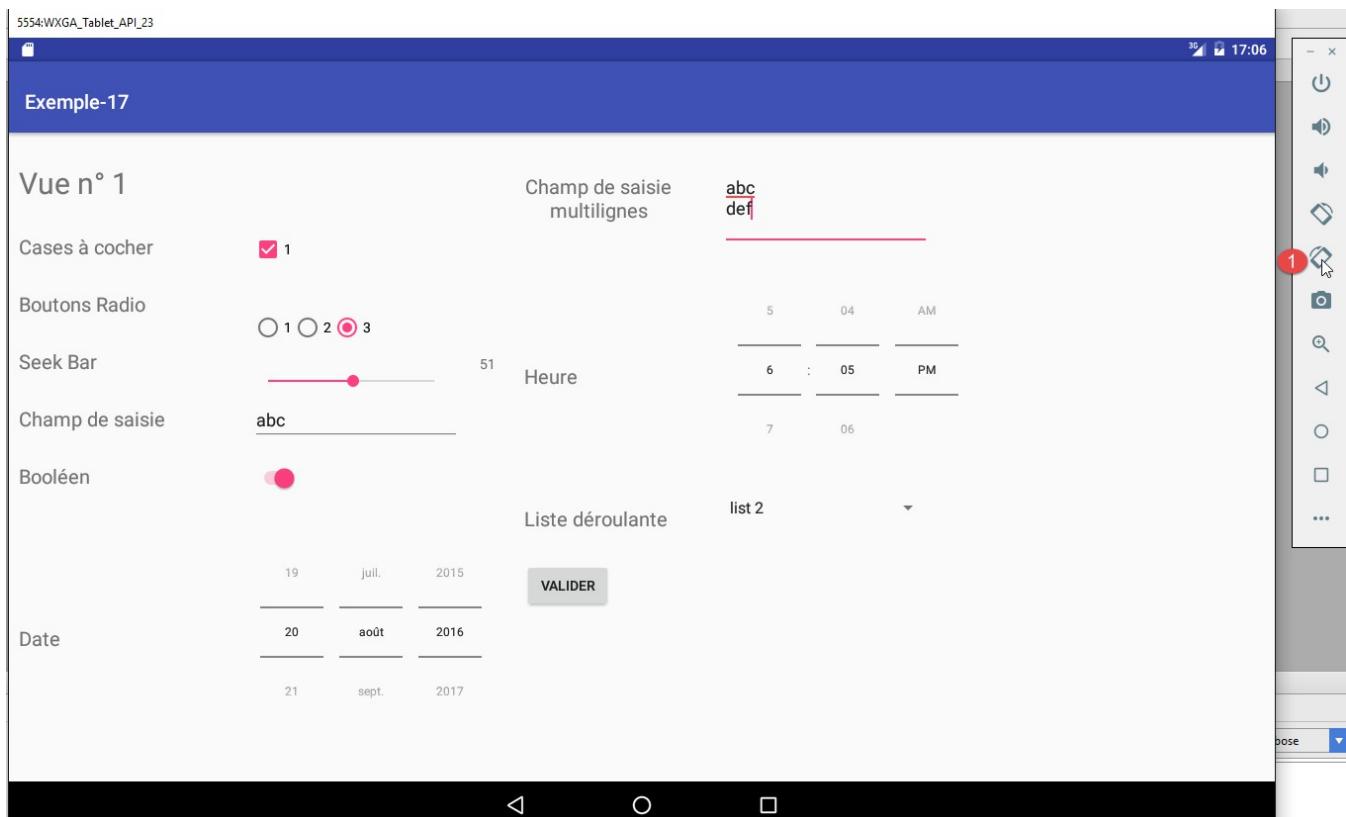
Pour illustrer l'utilisation du projet [client-android-skel], nous allons d'abord utiliser des exemples simples avant de passer à une étude de cas plus complète.

2.8 Exercices d'illustration

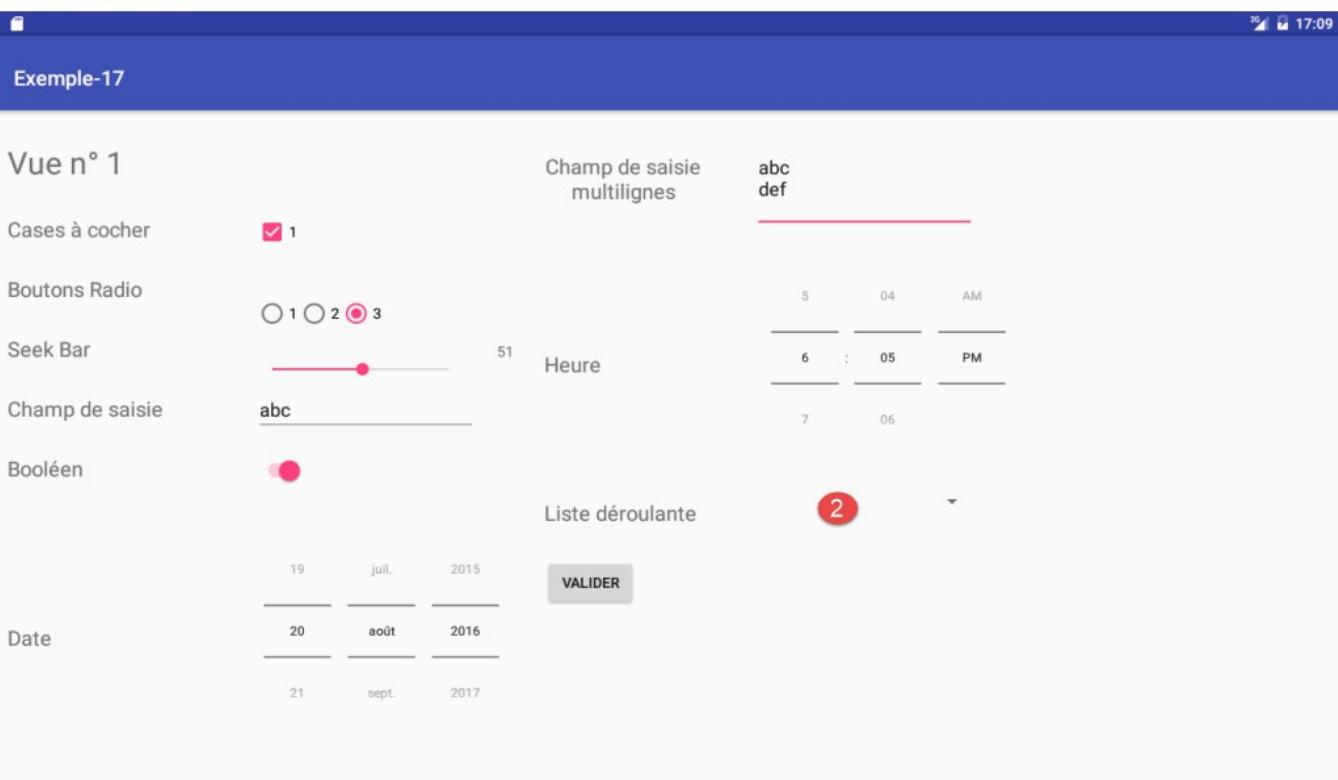
Nous allons commencer par refactoriser des exemples déjà écrits.

2.8.1 Exemple-17B

Nous reprenons l'exemple 17 étudié au paragraphe 1.18, page 210. C'est une application avec un unique fragment sans tâches asynchrones et sans onglets. Nous l'examinons pour voir comment elle se comporte lors d'une rotation du périphérique. Nous faisons les saisies suivantes :



Puis en [1], nous faisons tourner le périphérique deux fois. La nouvelle vue est alors la suivante :



Si nous comparons les vues, tout a été conservé sauf la liste [2] qui est désormais vide.

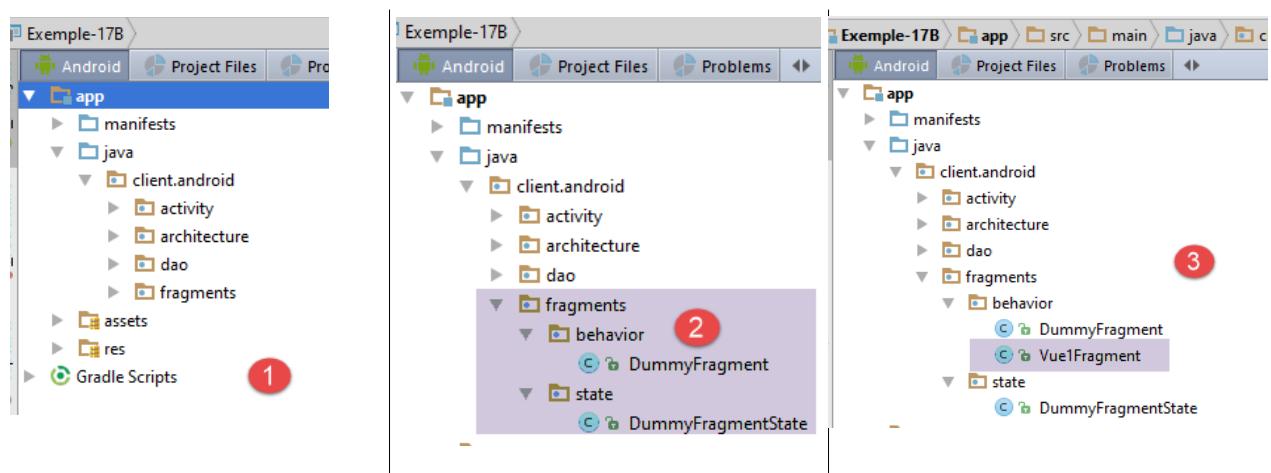
Par ailleurs, si on clique sur le bouton [Valider] on a une boîte de dialogue montrant les saisies faites sur le formulaire. Si à ce moment là, on fait tourner le périphérique, on perd la boîte de dialogue.

Il nous faudra donc, lors d'une rotation, régénérer :

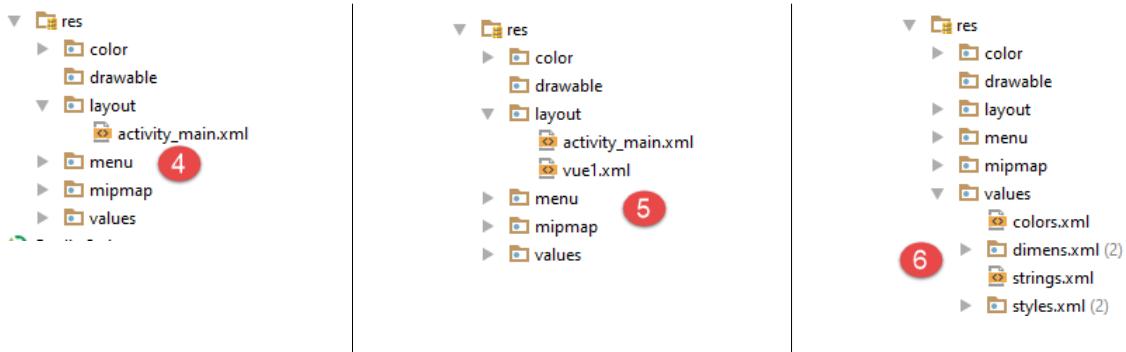
- la liste déroulante et son élément sélectionné ;
- la boîte de dialogue si elle était affichée lors de la rotation ;

2.8.1.1 Le projet [Exemple-17B]

Nous dupliquons le projet [client-android-skel] dans exemples/Exemple-17B. Puis nous chargeons le nouveau projet [1] :



- en [2-3], dans le dossier [behavior], nous collons le fragment [Vue1Fragment] du projet [Exemple-17] ;



- en [4-5], dans le dossier [layout] de [Exemple-17B], on colle la vue [vue1.xml] de [Exemple-17]. C'est la vue associée au fragment ;
- en [6], le dossier [values] de [Exemple-17B] est remplacé par le dossier [values] de [Exemple-17] ;

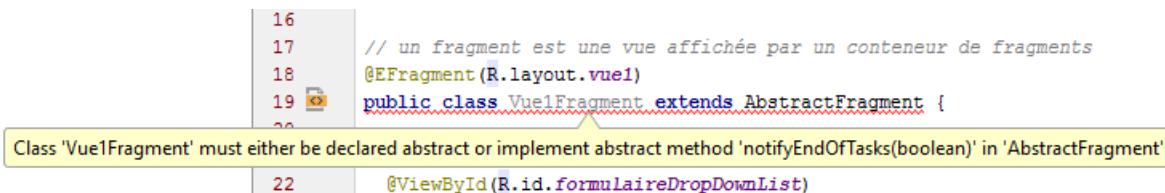
On modifiera la marge haute de la vue [vie1.xml] à 80 dp :

```

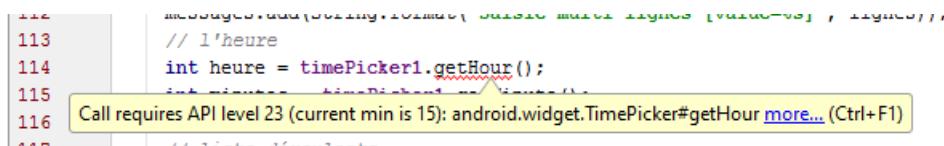
1. <TextView
2.     android:id="@+id/textViewFormulaireTitre"
3.     android:layout_width="wrap_content"
4.     android:layout_height="wrap_content"
5.     android:layout_alignParentLeft="true"
6.     android:layout_alignParentTop="true"
7.     android:layout_marginLeft="10dp"
8.     android:layout_marginTop="80dp"
9.     android:text="@string/titre_vue1"
10.    android:textSize="30sp"/>

```

A ce stade, on peut tenter une première compilation pour voir les erreurs. Les premières erreurs signalées proviennent d'*imports* de packages qui ont changé de place. On les corrige. D'autres erreurs proviennent du fait que la vue [Vue1Fragment] n'implémente pas toutes les méthodes imposées par sa classe parent [AbstractParent] :



Une autre erreur de compilation signalée est la suivante :



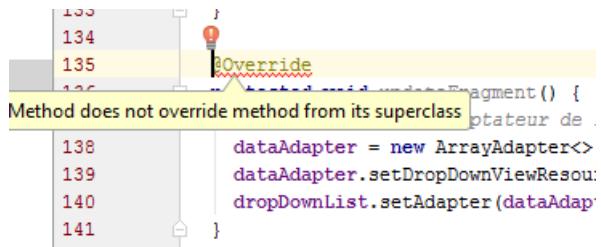
On corrige cela dans le fichier [build.gradle] du module (ligne 20 ci-dessous) :

```

18
19     defaultConfig {
20         minSdkVersion 23
21         targetSdkVersion 23
22         versionCode 1
23         versionName "1.0"
24     }

```

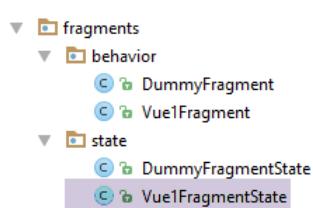
A ce stade, on peut recompiler pour voir les erreurs restantes. L'unique erreur signalée est sur la méthode [Vue1Fragment.updateFragment] :



Il faut supprimer l'annotation [@Override] de la ligne 135. Il n'y a désormais plus d'erreurs. Nous allons partir de là pour modifier le projet.

2.8.1.2 L'état du fragment [Vue1Fragment]

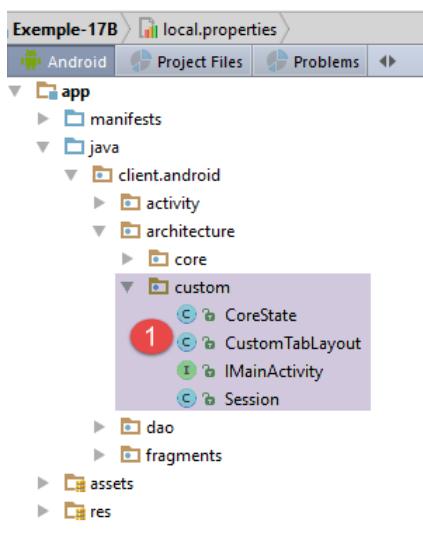
Le fragment [Vue1Fragment] a besoin de sauver des informations lors de la rotation du périphérique afin qu'il puisse être restauré complètement. Nous créons une classe [Vue1FragmentState] pour cela :



Pour l'instant, cette classe est vide :

```
1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. public class Vue1FragmentState extends CoreState {
6.
7. }
```

2.8.1.3 Personnalisation du projet



Dans le dossier [custom] se trouvent les éléments d'architecture personnalisables par le développeur.

Les constantes de l'interface [IMainActivity] seront les suivantes :

```
1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.ISession;
4. import client.android.dao.service.IDao;
5.
6. public interface IMainActivity extends IDao {
7.
8.     // accès à la session
9.     ISession getSession();
10.
11.    // changement de vue
12.    void navigateToView(int position, ISession.Action action);
13.
14.    // gestion de l'attente
15.    void beginWaiting();
16.
17.    void cancelWaiting();
18.
19.    // constantes de l'application -----
20.
21.    // mode debug
22.    boolean IS_DEBUG_ENABLED = true;
23.
24.    // délai maximal d'attente de la réponse du serveur
25.    int TIMEOUT = 1000;
26.
27.    // délai d'attente avant exécution de la requête client
28.    int DELAY = 0;
29.
30.    // authentification basique
31.    boolean IS_BASIC_AUTHENTICATION_NEEDED = false;
32.
33.    // adjacence des fragments
34.    int OFF_SCREEN_PAGE_LIMIT = 1;
35.
36.    // barre d'onglets
37.    boolean ARE_TABS_NEEDED = false;
38.
39.    // image d'attente
40.    boolean IS_WAITING_ICON_NEEDED = false;
41.
42.    // nombre de fragments de l'application
43.    int FRAGMENTS_COUNT = 1;
44.
45. }
```

- lignes 24-31 : l'application n'utilise pas ici sa couche [DAO]. Ces constantes ne seront pas utilisées ;
- ligne 34 : une adjacence de fragments de 1 qui est la valeur par défaut. Comme l'application n'a qu'un fragment (ligne 43), cette valeur n'a pas d'importance ;
- lignes 39-40 : comme il n'y a pas d'opérations avec la couche [DAO], il est inutile d'avoir une image d'attente ;
- ligne 37 : ce n'est pas une application à onglets ;

La classe [Session] est la suivante :

```
1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.AbstractSession;
4.
5. public class Session extends AbstractSession {
6.     // les éléments qui ne peuvent être sérialisés en JSON doivent avoir l'annotation @JsonIgnore
7.
8. }
```

Elle est vide. En effet, comme il n'y a qu'un fragment, il n'y a pas lieu de prévoir une communication inter-fragments avec une session.

Enfin, la classe [CoreState] est la suivante :

```
1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.MenuInstanceState;
4. import client.android.fragments.state.Vue1FragmentState;
5. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
6. import com.fasterxml.jackson.annotation.JsonSubTypes;
7. import com.fasterxml.jackson.annotation.JsonProperty;
```

```

9. @JsonIgnoreProperties(ignoreUnknown = true)
10. @JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
11. @JsonSubTypes({
12.     @JsonSubTypes.Type(value = Vue1FragmentState.class)}
13. )
14. public class CoreState {
15.     // fragment visité ou non
16.     protected boolean hasBeenVisited = false;
17.     // état de l'éventuel menu du fragment
18.     protected MenuItemState[] menuOptionsState;
19.
20.     // getters et setters
21. ...
22. }

```

- lignes 11-13 : il nous faut mettre toutes les classes dérivée de [CoreState] qui mémorisent l'état des différents fragments. Ici, il n'y en a qu'une (ligne 12) ;

2.8.1.4 L'activité [MainActivity]

L'activité [MainActivity] est actuellement la suivante :

```

1. package client.android.activity;
2.
3. import android.util.Log;
4. import client.android.R;
5. import client.android.architecture.core.AbstractActivity;
6. import client.android.architecture.core.AbstractFragment;
7. import client.android.architecture.custom.Session;
8. import client.android.dao.service.Dao;
9. import client.android.dao.service.IDao;
10. import org.androidannotations.annotations.Bean;
11. import org.androidannotations.annotations.EActivity;
12. import org.androidannotations.annotations.OptionsMenu;
13.
14. @EActivity
15. @OptionsMenu(R.menu.menu_main)
16. public class MainActivity extends AbstractActivity {
17.
18.     // couche [DAO]
19.     @Bean(Dao.class)
20.     protected IDao dao;
21.     // session
22.     private Session session;
23.
24.     // méthodes classe parent -----
25.     @Override
26.     protected void onCreateActivity() {
27.         // log
28.         if (IS_DEBUG_ENABLED) {
29.             Log.d(className, "onCreateActivity");
30.         }
31.         // session
32.         this.session = (Session) super.session;
33.         // todo : on continue les initialisations commencées par la classe parent
34.     }
35.
36.     @Override
37.     protected IDao getDao() {
38.         return dao;
39.     }
40.
41.     @Override
42.     protected AbstractFragment[] getFragments() {
43.         // todo : définir les fragments ici
44.         return new AbstractFragment[0];
45.     }
46.
47.
48.     @Override
49.     protected CharSequence getFragmentTitle(int position) {
50.         // todo : définir les titres des fragments ici
51.         return null;
52.     }
53.
54.     @Override
55.     protected void navigateOnTabSelected(int position) {
56.         // todo : navigation par onglets - définir la vue à afficher lorsque l'onglet n° [position] est sélectionné
57.     }
58.
59.     @Override
60.     protected int getViewId() {
61.         // todo : définir le n° de la première vue (fragment) à afficher
62.         return 0;

```

```
63. }
64. }
```

Les commentaires [//todo] indiquent ce que le développeur doit faire. La classe [MainActivity] évolue comme suit :

```
1. package client.android.activity;
2.
3. import android.util.Log;
4. import client.android.R;
5. import client.android.architecture.core.AbstractActivity;
6. import client.android.architecture.core.AbstractFragment;
7. import client.android.architecture.custom.Session;
8. import client.android.dao.Dao;
9. import client.android.dao.service.IDao;
10. import client.android.fragments.behavior.Vue1Fragment_;
11. import org.androidannotations.annotations.Bean;
12. import org.androidannotations.annotations.EActivity;
13. import org.androidannotations.annotations.OptionsMenu;
14.
15. @EActivity
16. @OptionsMenu(R.menu.menu_main)
17. public class MainActivity extends AbstractActivity {
18.
19.     // couche [DAO]
20.     @Bean(Dao.class)
21.     protected IDao dao;
22.     // session
23.     private Session session;
24.
25.     // méthodes classe parent -----
26.     @Override
27.     protected void onCreateActivity() {
28.         // log
29.         if (IS_DEBUG_ENABLED) {
30.             Log.d(className, "onCreateActivity");
31.         }
32.         // session
33.         this.session = (Session) super.session;
34.     }
35.
36.     @Override
37.     protected IDao getDao() {
38.         return dao;
39.     }
40.
41.     @Override
42.     protected AbstractFragment[] getFragments() {
43.         return new AbstractFragment[]{new Vue1Fragment_()};
44.     }
45.
46.     @Override
47.     protected CharSequence getFragmentTitle(int position) {
48.         return null;
49.     }
50.
51.     @Override
52.     protected void navigateOnTabSelected(int position) {
53.
54.     }
55.
56.     @Override
57.     protected int getViewType() {
58.         return 0;
59.     }
60. }
```

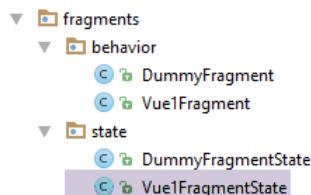
Seule la méthode des lignes 41-44 doit être modifiée. Elle doit rendre le tableau des fragments de l'application. Ligne 43, il ne faut pas oublier de mettre l'underscore derrière le nom du fragment.

2.8.1.5 L'état du fragment [FragmentState]

Suite aux tests de rotation faits sur le projet [Exemple-17], on décide de mémoriser les éléments suivants du fragment :

- la liste des valeurs de la liste déroulante ;
- la position de l'élément sélectionné dans cette liste ;
- le message affiché par la boîte de dialogue si celle-ci est présente au moment de la rotation ;

La classe [FragmentState] sera la suivante :



```

1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. import java.util.List;
6.
7. public class Vue1FragmentState extends CoreState {
8.
9.     // les valeurs de la liste déroulante
10.    private List<String> list;
11.    // l'élément sélectionné dans la liste déroulante
12.    private int listSelectedPosition;
13.    // le message affiché dans la boîte de dialogue
14.    private String message;
15.
16.    // getters et setters
17. ...
18. }
```

2.8.1.6 Le fragment [AbstractFragment]

Actuellement le cycle de vie du fragment est géré par deux méthodes (lignes 6 et 32) :

```

1. // liste déroulante
2. private List<String> list;
3. private ArrayAdapter<String> dataAdapter;
4.
5. @AfterViews
6. void afterViews() {
7.     // on coche le premier bouton
8.     radioButton1.setChecked(true);
9.     // le calendrier
10.    datePicker1.setCalendarViewShown(false);
11.    // le seekBar
12.    seekBar.setMax(100);
13.    seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
14.
15.        public void onStopTrackingTouch(SeekBar seekBar) {
16.            }
17.
18.        public void onStartTrackingTouch(SeekBar seekBar) {
19.            }
20.
21.        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
22.            seekBarValue.setText(String.valueOf(progress));
23.        }
24.    });
25.    // la liste déroulante
26.    list = new ArrayList<>();
27.    list.add("list 1");
28.    list.add("list 2");
29.    list.add("list 3");
30. }
31. ...
32. protected void updateFragment() {
33.     // initialisation adaptateur de la liste déroulante
34.     dataAdapter = new ArrayAdapter<>(activity, android.R.layout.simple_spinner_item, list);
35.     dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
36.     dropDownList.setAdapter(dataAdapter);
37. }
```

Le code de ces deux méthodes va migrer dans les méthodes imposées par la classe [AbstractFragment] de la façon suivante :

```

1. // gestion du cycle de vie du fragment -----
2. @Override
3. public CoreState saveFragment() {
4.     Vue1FragmentState state = new Vue1FragmentState();
5.     state.setList(list);
6.     state.setListSelectedPosition(dropDownList.getSelectedItemPosition());
7.     state.setMessage(message);
```

```

8.     return state;
9. }
10.
11. @Override
12. protected int getNumView() {
13.     return 0;
14. }
15.
16. @Override
17. protected void initFragment(CoreState previousState) {
18.     // 1ère visite ?
19.     if (previousState == null) {
20.         // on crée les valeurs de la liste déroulante
21.         list = new ArrayList<>();
22.         list.add("list 1");
23.         list.add("list 2");
24.         list.add("list 3");
25.     } else {
26.         // on restitue les valeurs de la liste déroulante
27.         Vue1FragmentState state = (Vue1FragmentState) previousState;
28.         list = state.getList();
29.         // et le message de la boîte de dialogue
30.         message = state.getMessage();
31.     }
32.     // initialisation adaptateur de la liste déroulante
33.     dataAdapter = new ArrayAdapter<>(activity, android.R.layout.simple_spinner_item, list);
34.     dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
35. }
36.
37. @Override
38. protected void initView(CoreState previousState) {
39.     // le calendrier
40.     datePicker1.setCalendarViewShown(false);
41.     // le seekBar
42.     seekBar.setMax(100);
43.     seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
44.
45.         public void onStopTrackingTouch(SeekBar seekBar) {
46.         }
47.
48.         public void onStartTrackingTouch(SeekBar seekBar) {
49.         }
50.
51.         public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
52.             seekBarValue.setText(String.valueOf(progress));
53.         }
54.     });
55.     // initialisation adaptateur de la liste déroulante
56.     dropDownList.setAdapter(dataAdapter);
57.     // 1ère visite ?
58.     if (previousState == null) {
59.         // on coche le premier bouton
60.         radioButton1.setChecked(true);
61.     }
62. }
63.
64. @Override
65. protected void updateOnSubmit(CoreState previousState) {
66.
67. }
68.
69. @Override
70. protected void updateOnRestore(CoreState previousState) {
71.     // valeur seekbar
72.     seekBarValue.setText(String.valueOf(seekBar.getProgress()));
73.     // élément sélectionné dans liste déroulante
74.     Vue1FragmentState state = (Vue1FragmentState) previousState;
75.     dropDownList.setSelection(state.getListSelectedPosition());
76.     // dialogue visible ?
77.     if (message != null) {
78.         // on l'affiche
79.         showMessage();
80.     }
81. }
82.
83. @Override
84. protected void notifyEndOfUpdates() {
85.
86. }
87.
88. @Override
89. protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
90.
91. }

```

- lignes 2-9 : la méthode [saveFragment] doit mettre les éléments du fragment à mémoriser dans une classe dérivée de [CoreState] et rendre l'instance de celle-ci ;
- lignes 11-14 : la méthode [getNumView] doit rendre le n° du fragment. Ici, il n'y a qu'un fragment dont le n° est 0 ;
- lignes 16-34 : la méthode [initFragment] doit initialiser les champs du fragment. Elle reçoit l'état précédent du fragment. Si [previousState] vaut `null`, alors il s'agit de la 1ère visite ;
- lignes 19-25 : lors de la 1ère visite, on crée les valeurs de la liste déroulante ;
- lignes 26-30 : s'il ne s'agit pas de la 1ère visite, les champs [list, message] du fragment sont restaurés à partir de l'état précédent ;
- lignes 33-34 : initialisation du champ [dataAdapter] du fragment. C'est la source de données de la liste déroulante ;
- lignes 37-62 : la méthode [initView] sert à initialiser les composants de l'interface visuelle. Elle reçoit comme paramètre, l'état précédent [previousState]. Si [previousState==null], alors il s'agit de la 1ère visite ;
- on retrouve ici, ce qu'il y avait auparavant dans la méthode [@AfterViews] ;
- lignes 57-61 : lors de la 1ère visite, on s'assure que c'est le premier bouton radio qui est coché ;
- lignes 64-67 : la méthode [updateOnSubmit] est exécutée lorsque l'action en cours est [SUBMIT]. Ici, il n'y a pas de navigation inter-fragments et donc pas d'action en cours ;
- lignes 69-81 : la méthode [updateOnRestore] est exécutée lorsque l'action en cours est [NAVIGATION] ou [RESTORE]. Ici, il n'y a pas de navigation inter-fragments et donc pas d'action [NAVIGATION] possible ;
- ligne 72 : on recalcule (pas restaure) la valeur du *TextView seekBarValue*. En effet, lors de rotations, on perdait parfois sa valeur ;
- lignes 74-75 : on positionne la liste sur l'élément qui était sélectionné avant la rotation. Sans cela, la liste se positionnait sur son 1er élément ;
- lignes 76-80 : on réaffiche la boîte de dialogue si le message de l'état précédent est non `null`. Nous reviendrons sur la méthode [showMessage] (ligne 79) ;
- lignes 83-86 : la méthode [notifyEndOfUpdates] est la dernière méthode appelée par la classe parent avant de laisser le fragment fille tranquille. Ici il n'y a rien à faire ;
- lignes 88-91 : la méthode [notifyEndOfTasks] signale la fin des tâches asynchrones lancées par le fragment. Ici, il n'y en a pas ;

La restauration de la boîte de dialogue se fait de la façon suivante :

```

1.   // le message de la boîte de dialogue
2.   private String message;
3.   ...
4.   @Click(R.id.formulaireButtonValider)
5.   protected void doValider() {
6.       // liste des messages à afficher
7.       List<String> messages = new ArrayList<>();
8.       ...
9.       // affichage
10.      doAfficher(messages);
11.  }
12.
13.  private void doAfficher(final List<String> messages) {
14.      // on construit le texte à afficher
15.      StringBuilder texte = new StringBuilder();
16.      for (String message : messages) {
17.          texte.append(String.format("%s\n", message));
18.      }
19.      // on mémorise le message
20.      message = texte.toString();
21.      // on l'affiche
22.      showMessage();
23.  }
24.
25.  private void showMessage() {
26.      // on l'affiche
27.      new AlertDialog.Builder(activity).setTitle("Valeurs saisies").setMessage(message).setNeutralButton("Fermer", new
DialogInterface.OnClickListener() {
28.          @Override
29.          public void onClick(DialogInterface dialog, int which) {
30.              // reset du message
31.              message = null;
32.          }
33.      }).show();
34.  }

```

Lorsque l'utilisateur valide le formulaire, la méthode [doValider] (ligne 5) construit une liste de messages qu'elle fait ensuite afficher (ligne 10) dans la boîte de dialogue.

- lignes 14-20 : la liste de messages est concaténée en un seul message qui est mémorisé en ligne 2 ;
- lignes 25-33 : c'est ce message qu'affiche la boîte de dialogue et c'est ce même message que la méthode [updateOnRestore] fait afficher ;

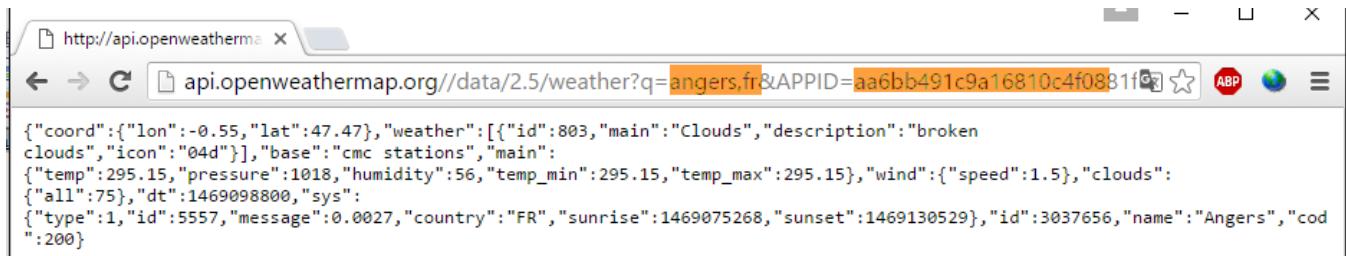
- ligne 27 : le second paramètre de la méthode [setNeutralButton] est la méthode exécutée lorsque l'utilisateur clique sur le bouton [Fermer] de la boîte de dialogue ;
- ligne 31 : sur la fermeture de la boîte de dialogue, on remet le message à *null* pour indiquer que la boîte de dialogue n'est plus présente ;

2.8.1.7 Tests

Le lecteur est invité à tester ce projet et à vérifier que le fragment est bien conservé après une ou plusieurs rotations successives.

2.8.2 Exemple-23 : client météo

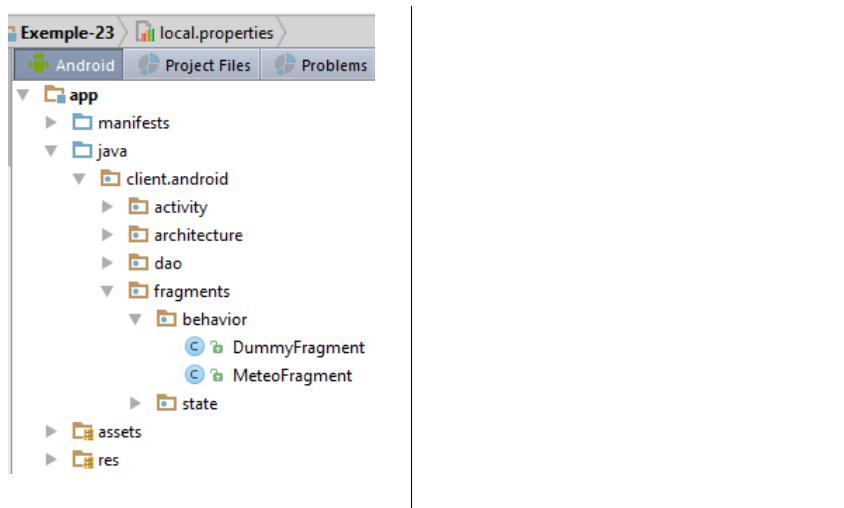
Certains sites permettent d'avoir des informations météo sous forme de chaînes JSON. Voici un exemple :



L'URL est de la forme : `http://api.openweathermap.org/data/2.5/weather?q={city},{country}&APPID={APPID}` avec :

- city : la ville dont on veut la météo, ici Angers ;
- country : le pays de la ville, ici la France (fr) ;
- APPID : une clé obtenue en s'inscrivant sur le site [https://home.openweathermap.org/users/sign_up];

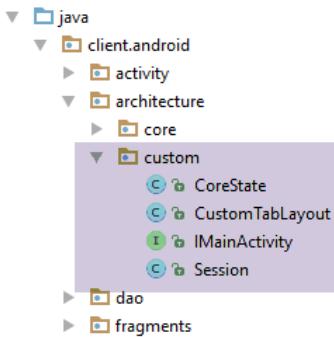
2.8.2.1 Le projet



Le projet a été construit à partir du projet [client-android-skel]. Il présente les caractéristiques suivantes :

- il n'a qu'un fragment dont on n'a pas à conserver l'état ;
- il fait des requêtes asynchrones ;

2.8.2.2 Personnalisation du projet



L'interface [IMainActivity] permet de spécifier certaines caractéristiques du projet :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.ISession;
4. import client.android.dao.service.IDao;
5.
6. public interface IMainActivity extends IDao {
7.
8.     // accès à la session
9.     ISession getSession();
10.
11.    // changement de vue
12.    void navigateToView(int position, ISession.Action action);
13.
14.    // gestion de l'attente
15.    void beginWaiting();
16.
17.    void cancelWaiting();
18.
19.    // constantes de l'application -----
20.
21.    // mode debug
22.    boolean IS_DEBUG_ENABLED = true;
23.
24.    // délai maximal d'attente de la réponse du serveur
25.    int TIMEOUT = 1000;
26.
27.    // délai d'attente avant exécution de la requête client
28.    int DELAY = 5000;
29.
30.    // authentification basique
31.    boolean IS_BASIC_AUTHENTICATION_NEEDED = false;
32.
33.    // adjacence des fragments
34.    int OFF_SCREEN_PAGE_LIMIT = 1;
35.
36.    // barre d'onglets
37.    boolean ARE_TABS_NEEDED = false;
38.
39.    // image d'attente
40.    boolean IS_WAITING_ICON_NEEDED = true;
41.
42.    // nombre de fragments de l'application
43.    int FRAGMENTS_COUNT = 1;
44.
45. }
  
```

- lignes 25, 28, 31, 40 : caractéristiques de la couche [DAO]. Ligne 31, il n'y a pas besoin d'authentification basique ;
- ligne 34 : adjacence des fragments. Ici cette constante n'a pas d'importance puisqu'il n'y a qu'un fragment ;
- ligne 37 : ce n'est pas une application à onglets ;
- ligne 43 : il n'y a qu'un fragment ;

La classe [CoreState] qui mémorise l'état des fragments sera la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.MenuItemState;
4. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
5. import com.fasterxml.jackson.annotation.JsonTypeInfo;
6.
7. @JsonIgnoreProperties(ignoreUnknown = true)
  
```

```

8. @JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
9. // todo : ajouter ici les sous-classes de [CoreState]
10. /*@JsonSubTypes({
11.     @JsonSubTypes.Type(value = Class1.class),
12.     @JsonSubTypes.Type(value = Class2.class)}
13.)*/
14. public class CoreState {
15.     // fragment visité ou non
16.     protected boolean hasBeenVisited = false;
17.     // état de l'éventuel menu du fragment
18.     protected MenuItemState[] menuOptionsState;
19.
20.     // getters et setters
21. ...
22. }

```

- lignes 10-13 : il n'y a rien à déclarer puisque dans cette application il n'y a qu'un fragment dont on ne conserve pas l'état ;

La classe [Session] est la suivante :

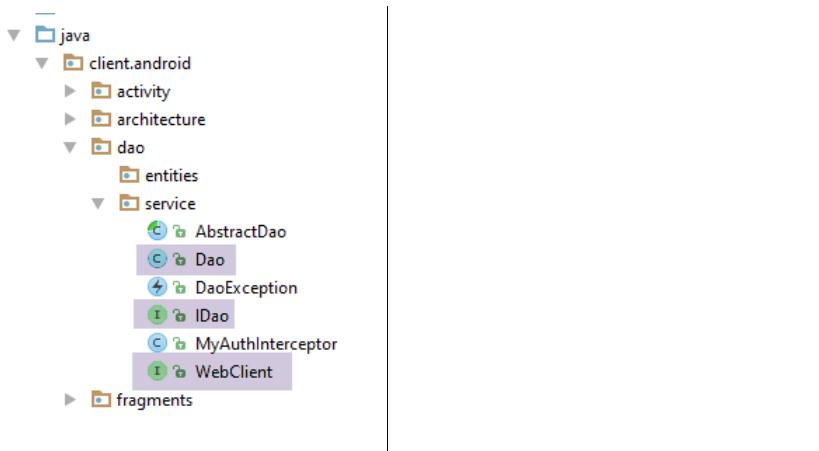
```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.AbstractSession;
4.
5. public class Session extends AbstractSession {
6.     // les éléments qui ne peuvent être sérialisés en JSON doivent avoir l'annotation @JsonIgnore
7. }

```

Elle est vide puisque dans cette application, il n'y a pas de communication inter-fragments.

2.8.2.3 La couche [DAO]



Dans la couche [DAO], trois classes doivent être personnalisées :

- l'interface **IDao** ;
- son implémentation **Dao** ;
- l'interface **WebClient** de dialogue avec le serveur web / JSON ;

L'interface [WebClient] sera la suivante :

```

1. package client.android.dao.service;
2.
3. import org.androidannotations.rest.spring.annotations.Get;
4. import org.androidannotations.rest.spring.annotations.Path;
5. import org.androidannotations.rest.spring.annotations.Rest;
6. import org.androidannotations.rest.spring.api.RestClientRootUrl;
7. import org.androidannotations.rest.spring.api.RestClientSupport;
8. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
9. import org.springframework.web.client.RestTemplate;
10.
11. @Rest(converters = {MappingJackson2HttpMessageConverter.class})
12. public interface WebClient extends RestClientRootUrl, RestClientSupport {
13.
14.     // RestTemplate
15.     void setRestTemplate(RestTemplate restTemplate);
16.
17.     // service météo
18.     @Get("/data/2.5/weather?q={city},{country}&APPID={APPID}")

```

```

19.     String getWeatherForecast(@Path String city, @Path String country, @Path String APPID);
20. }

```

- lignes 18-19 : l'URL du service météo. On rappelle que celle-ci est relative à l'URL racine (RestClientRootUrl, ligne 12) du client. Ici cette URL racine sera [http://api.openweathermap.org/];

L'interface [IDao] sera la suivante :

```

1. package client.android.dao.service;
2.
3. import rx.Observable;
4.
5. public interface IDao {
6.     // Url du service web
7.     void setUrlServiceWebJson(String url);
8.
9.     // utilisateur
10.    void setUser(String user, String mdp);
11.
12.    // timeout du client
13.    void setTimeout(int timeout);
14.
15.    // authentification basique
16.    void setBasicAuthentification(boolean isBasicAuthentificationNeeded);
17.
18.    // mode debug
19.    void setDebugMode(boolean isDebugEnabled);
20.
21.    // délai d'attente en millisecondes du client avant requête
22.    void setDelay(int delay);
23.
24.    // service météo
25.    Observable<String> getWeatherForecast(String city, String country, String APPID);
26. }

```

- on rappelle que les méthodes des lignes 6-22 sont présentes par défaut dans l'interface IDao du projet [client-android-skel] ;
- ligne 25 : la méthode [getWeatherForecast] permet d'obtenir la chaîne JSON de la météo de la ville [city] du pays [country]. Le 3ième paramètre est la clé obtenue sur le site [https://home.openweathermap.org/users/sign_up];

L'interface [IDao] est implémentée par la classe [Dao] suivante :

```

1. package client.android.dao.service;
2.
3. import android.util.Log;
4. import org.androidannotations.annotations.AfterInject;
5. import org.androidannotations.annotations.Bean;
6. import org.androidannotations.annotations.EBean;
7. import org.androidannotations.rest.spring.annotations.RestService;
8. import org.springframework.http.client.ClientHttpRequestInterceptor;
9. import org.springframework.http.client.SimpleClientHttpRequestFactory;
10. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
11. import org.springframework.web.client.RestTemplate;
12. import rx.Observable;
13.
14. import java.util.ArrayList;
15. import java.util.List;
16.
17. @EBean(scope = EBean.Scope.Singleton)
18. public class Dao extends AbstractDao implements IDao {
19.
20.     // client du service web
21.     @RestService
22.     protected WebClient webClient;
23.     // sécurité
24.     @Bean
25.     protected MyAuthInterceptor authInterceptor;
26.     // le RestTemplate
27.     private RestTemplate restTemplate;
28.     // factory du RestTemplate
29.     private SimpleClientHttpRequestFactory factory;
30.     // timeout
31.     private int timeout;
32.
33.     @AfterInject
34.     public void afterInject() {
35.         // log
36.         Log.d(className, "afterInject");
37.         // on construit le restTemplate
38.         factory = new SimpleClientHttpRequestFactory();
39.         restTemplate = new RestTemplate(factory);
40.         // on fixe le convertisseur JSON

```

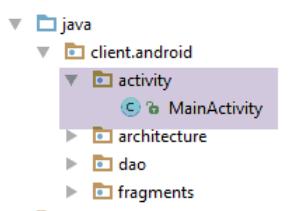
```

41.     restTemplate.setMessageConverters().add(new MappingJackson2HttpMessageConverter());
42.     // on fixe le restTemplate du client web
43.     webClient.setRestTemplate(restTemplate);
44. }
45.
46. @Override
47. public void setUrlServiceWebJson(String url) {
48.     // on fixe l'URL du service web
49.     webClient.setRootUrl(url);
50. }
51.
52. @Override
53. public void setUser(String user, String mdp) {
54.     // on enregistre l'utilisateur dans l'intercepteur
55.     authInterceptor.setUser(user, mdp);
56. }
57.
58. @Override
59. public void setTimeout(int timeout) {
60.     if (isDebugEnabled) {
61.         Log.d(className, String.format("setTimeout thread=%s, timeout=%s", Thread.currentThread().getName(), timeout));
62.     }
63.     // mémoire
64.     this.timeout = timeout;
65.     // configuration factory
66.     factory.setReadTimeout(timeout);
67.     factory.setConnectTimeout(timeout);
68. }
69.
70. @Override
71. public void setBasicAuthentification(boolean isBasicAuthentificationNeeded) {
72.     if (isDebugEnabled) {
73.         Log.d(className, String.format("setBasicAuthentification thread=%s, isBasicAuthentificationNeeded=%s",
74.             Thread.currentThread().getName(), isBasicAuthentificationNeeded));
75.     }
76.     // intercepteur d'authentification ?
77.     if (isBasicAuthentificationNeeded) {
78.         // on ajoute l'intercepteur d'authentification
79.         List<ClientHttpRequestInterceptor> interceptors = new ArrayList<ClientHttpRequestInterceptor>();
80.         interceptors.add(authInterceptor);
81.         restTemplate.setInterceptors(interceptors);
82.     }
83.
84.
85. // méthodes privées -----
86. private void log(String message) {
87.     if (isDebugEnabled) {
88.         Log.d(className, message);
89.     }
90. }
91.
92. // service météo -----
93. @Override
94. public Observable<String> getWeatherForecast(final String city, final String country, final String APPID) {
95.     // log
96.     if (isDebugEnabled) {
97.         Log.d(className, String.format("getWeatherForecast city=%s, country=%s, APIID=%s, thread=%s, timeout=%s", city,
98.             country, APPID, Thread.currentThread().getName(), timeout));
99.     }
100.    // résultat
101.    return getResponse(new IRequest<String>()) {
102.        @Override
103.        public String getResponse() {
104.            return webClient.getWeatherForecast(city, country, APPID);
105.        }
106.    });
107. }

```

- on rappelle que les lignes 17-90 sont présentes par défaut dans la classe [Dao] du projet [client-android-skel]. Il faut juste ajouter les méthodes d'implémentation de l'interface [IDao], spécifiques à l'application (ligne 92) ;
- lignes 93-105 : implémentation de la méthode [getWeatherForecast]. Celle-ci est très simple et réalisée en 6 lignes, lignes 100-105 ;
- ligne 100 : la méthode [getResponse] est une méthode de la classe parent [AbstractDao]. Elle attend un paramètre de type [IRequest<T>] où T est le type de la réponse attendue du serveur, ici un *String* puisqu'on attend une chaîne JSON. Le type T de [IRequest<T>] doit être le type T de la méthode [Observable<T> getWeatherForecast] ;
- l'interface [IRequest<T>] n'a qu'une méthode : *getResponse*. Celle-ci a pour rôle de fournir la réponse de type T que doit rendre la méthode [Observable<T> getWeatherForecast] ;
- ligne 103 : c'est l'interface [WebClient] qui fournit cette réponse. On lui passe les trois paramètres reçus ligne 94. Pour cette raison, ceux-ci doivent avoir l'attribut **final** ;

2.8.2.4 L'activité [MainActivity]



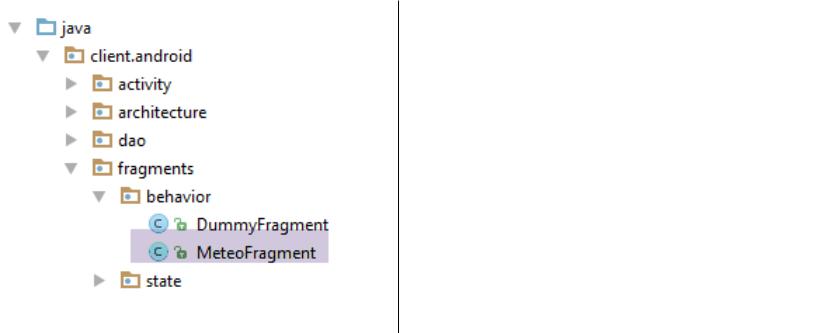
L'activité [MainActivity] est la suivante :

```
1. package client.android.activity;
2.
3. import android.util.Log;
4. import client.android.R;
5. import client.android.architecture.core.AbstractActivity;
6. import client.android.architecture.core.AbstractFragment;
7. import client.android.dao.Dao;
8. import client.android.dao.service.IDao;
9. import client.android.fragments.behavior.MeteoFragment_;
10. import org.androidannotations.annotations.Bean;
11. import org.androidannotations.annotations.EActivity;
12. import org.androidannotations.annotations.OptionsMenu;
13. import rx.Observable;
14.
15. @EActivity
16. @OptionsMenu(R.menu.menu_main)
17. public class MainActivity extends AbstractActivity {
18.
19.     // couche [DAO]
20.     @Bean(Dao.class)
21.     protected IDao dao;
22.
23.     // méthodes classe parent -----
24.     @Override
25.     protected void onCreateActivity() {
26.         // log
27.         if (IS_DEBUG_ENABLED) {
28.             Log.d(class.getName(), "onCreateActivity");
29.         }
30.     }
31.
32.     @Override
33.     protected IDao getDao() {
34.         return dao;
35.     }
36.
37.     @Override
38.     protected AbstractFragment[] getFragments() {
39.         return new AbstractFragment[]{new MeteoFragment_()};
40.     }
41.
42.
43.     @Override
44.     protected CharSequence getFragmentTitle(int position) {
45.         return null;
46.     }
47.
48.     @Override
49.     protected void navigateOnTabSelected(int position) {
50.     }
51.
52.     @Override
53.     protected int getFirstView() {
54.         return 0;
55.     }
56.
57.     // interface IDao -----
58.     @Override
59.     public Observable<String> getWeatherForecast(String city, String country, String APPID) {
60.         return dao.getWeatherForecast(city, country, APPID);
61.     }
62. }
```

- on rappelle que les lignes 15-55 sont présentes par défaut dans le projet [client-android-skel]. Il faut juste les personnaliser ;
- lignes 37-40 : le tableau des fragments. Il n'y en a qu'un ici ;

- lignes 43-46 : pas de titres de fragments nécessaires ;
- lignes 48-50 : pas d'onglets ici ;
- lignes 52-55 : la 1ère vue à afficher est la vue n° 0, celle de [MeteoFragment] ;
- lignes 58-61 : implémentation de l'interface [IDao]. Ici, il n'y a rien à faire d'autre qu'à déléguer le travail à la couche [DAO] de la ligne 21 ;

2.8.2.5 Le fragment [MeteoFragment]



Le fragment [MeteoFragment] interroge le service web / JSON de météo. Son squelette est le suivant :

```

1. package client.android.fragments;
2.
3. import android.util.Log;
4. import android.widget.Toast;
5. import client.android.R;
6. import client.android.architecture.AbstractFragment;
7. import client.android.architecture.MenuInstanceState;
8. import org.androidannotations.annotations.EFragment;
9. import org.androidannotations.annotations.OptionsItem;
10. import org.androidannotations.annotationsOptionsMenu;
11. import rx.functions.Action0;
12. import rx.functions.Action1;
13.
14. @EFragment(R.layout.meteo_fragment)
15. @OptionsMenu(R.menu.menu_meteo)
16. public class FirstFragment extends AbstractFragment {
17. ...
18. }
  
```

- ligne 14 : la vue [res / layout / meteo_fragment.xml] est la suivante :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.   android:layout_width="match_parent"
4.   android:layout_height="match_parent">
5.
6.   <TextView
7.     android:layout_width="wrap_content"
8.     android:layout_height="wrap_content"
9.     android:textAppearance="?android:attr/textAppearanceLarge"
10.    android:text="Construisez votre interface visuelle"
11.    android:id="@+id/textView" android:layout_alignParentTop="true" android:layout_alignParentLeft="true"
12.    android:layout_alignParentStart="true" android:layout_marginLeft="64dp" android:layout_marginStart="64dp"
13.    android:layout_marginTop="120dp"/>
14. </RelativeLayout>
  
```

La vue n'affiche que le texte de la ligne 10 ;

- ligne 15 : le menu [res / menu / menu_meteo.xml] est le suivant :

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.   xmlns:app="http://schemas.android.com/apk/res-auto"
3.   xmlns:tools="http://schemas.android.com/tools"
4.   tools:context=".activity.MainActivity">
5.   <item
6.     android:id="@+id/menuActions"
7.     app:showAsAction="ifRoom"
8.     android:title="@string/menuActions">
9.     <menu>
10.       <item
11.         android:id="@+id/actionMeteo"
12.         android:title="@string/actionMeteo"/>
13.       <item
  
```

```

14.        android:id="@+id/actionAnnuler"
15.        android:title="@string/actionAnnuler"/>
16.    <item
17.        android:id="@+id/actionTerminer"
18.        android:title="@string/actionTerminer"/>
19.    </menu>
20.  </item>
21. </menu>

```

- lignes 10-12 : cette option de menu sert à demander la météo d'une ville ;
- lignes 14-15 : cette option de menu sert à annuler cette demande si elle est en cours ;
- lignes 16-18 : cette option de menu termine l'application ;

Le code complet du fragment est le suivant :

```

1. package client.android.fragments.behavior;
2.
3. import android.util.Log;
4. import android.widget.Toast;
5. import client.android.R;
6. import client.android.architecture.core.AbstractFragment;
7. import client.android.architecture.core.MenuItemState;
8. import client.android.architecture.custom.CoreState;
9. import org.androidannotations.annotations.EFragment;
10. import org.androidannotations.annotations.OptionsItem;
11. import org.androidannotations.annotations.OptionsMenu;
12. import rx.functions.Action1;
13.
14. @EFragment(R.layout.meteo_fragment)
15. @OptionsMenu(R.menu.menu_meteo)
16. public class MeteoFragment extends AbstractFragment {
17.
18.     // données locales
19.     private int nbReponsesRecues;
20.
21.     // gestion des événements -----
22.     // villes dont on veut la météo
23.     final String[] paysDeLoire = new String[]{"angers", "le mans", "nantes", "laval", "la roche sur yon"};
24.
25.     @OptionsItem(R.id.actionMeteo)
26.     protected void doMeteo() {
27.         // son pays
28.         String country = "fr";
29.         // obtenez un identifiant API en créant un compte [https://home.openweathermap.org/users/sign_up]
30.         String APPID = "xyz";
31.         // URL du service web / json
32.         mainActivity.setUrlServiceWebJson("http://api.openweathermap.org");
33.         // début de l'attente
34.         beginWaiting(paysDeLoire.length);
35.         // nbre de réponses reçues
36.         nbReponsesRecues = 0;
37.         // on fait les appels asynchrones en parallèle
38.         for (String city : paysDeLoire) {
39.             // météo
40.             executeInBackground(mainActivity.getWeatherForecast(city, country, APPID), new Action1<String>() {
41.                 @Override
42.                 public void call(String response) {
43.                     // exploitation de la réponse
44.                     consumeResponse(response);
45.                     // une réponse de +
46.                     nbReponsesRecues++;
47.                 }
48.             });
49.         }
50.     }
51.
52.     // exploitation réponse du serveur
53.     private void consumeResponse(String response) {
54.         // log
55.         Log.d(className, String.format("thread=%s, response=%s", Thread.currentThread().getName(), response));
56.     }
57.
58.     // début de l'attente
59.     protected void beginWaiting(int numberOfRunningTasks) {
60.         // log
61.         if (isDebugEnabled) {
62.             Log.d(className, "beginWaiting");
63.         }
64.         // parent
65.         beginRunningTasks(numberOfRunningTasks);
66.         // on affiche l'option [Annuler]
67.         setAllMenuOptionsStates(false);
68.         setMenuOptionsStates(new MenuItemState[]{
69.             new MenuItemState(R.id.menuActions, true),

```

```

70.         new MenuItemState(R.id.actionAnnuler, true)));
71.
72.     }
73.
74.     @Override
75.     protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
76.         // menu
77.         initMenu();
78.         // affichage résultats
79.         String message;
80.         switch (nbReponsesRecues) {
81.             case 0:
82.                 message = "Aucune réponse n'a été reçue";
83.                 break;
84.             case 1:
85.                 message = "Une réponse a été reçue. Consultez vos logs...";
86.                 break;
87.             default:
88.                 message = String.format("%s réponses ont été reçues. Consultez vos logs...", nbReponsesRecues);
89.                 break;
90.         }
91.         Toast.makeText(activity, message, Toast.LENGTH_SHORT).show();
92.     }
93.
94.     // méthodes privées -----
95.     private void initMenu() {
96.         if (isDebugEnabled) {
97.             Log.d(className, "initMenu");
98.         }
99.         // menu
100.        setAllMenuOptionsStates(true);
101.        setMenuOptionsStates(new MenuItemState[]{new MenuItemState(R.id.actionAnnuler, false)});
102.    }
103.
104.    // gestion du cycle de vie -----
105. ...
106. }
```

- lignes 25-50 : gestion du clic sur l'option de menu [Météo] ;
- ligne 32 : construction de l'URL du service web / JSON du service météo. Celle-ci est ensuite passée à la couche [DAO] via l'activité ;
- ligne 34 : on commence l'attente. On passe le nombre de tâches qui vont être lancées, ceci afin que la classe parent puisse nous signaler la fin de celles-ci. Ici, il y a cinq tâches car on va demander la météo des cinq villes de la ligne 23 ;
- ligne 16 : on va compter le nombre de réponses reçues afin de pouvoir l'afficher ;
- lignes 38-50 : on boucle sur les villes dont on veut la météo ;
- ligne 40 : on va faire 5 requêtes HTTP en parallèle ;
- ligne 40 : on demande à la classe parent [AbstractParent] d'interroger le service web / JSON ;
- lignes 40-48 : la méthode [executeInBackground] attend deux paramètres :
 - ligne 40 : le processus à observer et exécuter est fourni par la méthode [mainActivity.getWeatherForecast] ;
 - lignes 40-48 : l'instance [Action1] à exécuter lorsqu'on reçoit la réponse du service asynchrone. Le type T de [Action1<T>] doit être le type T du résultat de la méthode [getWeatherForecast] ;
- ligne 44 : une réponse a été reçue. On la passe à la méthode [consumeResponse] de la ligne 53 ;
- ligne 46 : on incrémente le compteur des réponses reçues
- lignes 53-56 : consommation d'une réponse JSON du service météo ;
- ligne 55 : on se contente de loguer la chaîne JSON ;
- lignes 59-72 : code exécuté avant le lancement des tâches asynchrones ;
- ligne 65 : on passe le nombre de tâches à exécuter à la classe parent [AbstractParent]. C'est ce qui permet à celle-ci de nous avertir lorsqu'elles seront toutes terminées ;
- lignes 67-70 : préparation du menu pour une attente. On ne garde que l'option [Actions/Annuler] qui va permettre à l'utilisateur d'annuler les tâches lancées ;
- lignes 74-92 : code exécuté lorsque la classe parent nous avertit que toutes les tâches lancées sont terminées ;
- ligne 77 : on remet le menu dans son état initial. La méthode [initMenu] (lignes 95-102) affiche le menu avec toutes ses options sauf l'option [Actions/Annuler] qui est cachée ;
- lignes 80-91 : on affiche le nombre de réponses reçues ;

Le clic sur l'option de menu [Annuler] est géré par le code suivant :

```

1.     @OptionsItemSelected(R.id.actionAnnuler)
2.     protected void doAnnuler() {
3.         if (isDebugEnabled) {
4.             Log.d(className, "Annulation demandée");
5.         }
6.         // on annule les tâches asynchrones
7.         cancelRunningTasks();
8.     }
```

- ligne 7 : on demande à la classe parent d'annuler les tâches encore actives ;

Le clic sur l'option de menu [Terminer] est géré par le code suivant :

```

1.  @OptionsItem(R.id.actionTerminer)
2.  protected void doTerminer() {
3.      // on arrête tout
4.      System.exit(0);
5.  }

```

La gestion du cycle de vie du fragment est assuré par les méthodes suivantes :

```

1.  // gestion du cycle de vie -----
2.
3.  @Override
4.  public CoreState saveFragment() {
5.      return new CoreState();
6.  }
7.
8.  @Override
9.  protected int getNumView() {
10.     return 0;
11. }
12.
13. @Override
14. protected void initFragment(CoreState previousState) {
15.
16. }
17.
18. @Override
19. protected void initView(CoreState previousState) {
20.     // 1ère visite ?
21.     if (previousState == null) {
22.         initMenu();
23.     }
24. }
25.
26.
27. @Override
28. protected void updateOnSubmit(CoreState previousState) {
29.
30. }
31.
32. @Override
33. protected void updateOnRestore(CoreState previousState) {
34.
35. }
36.
37. @Override
38. protected void notifyEndOfUpdates() {
39.
40. }

```

- lignes 3-6 : servent à mémoriser l'état du fragment dans une classe dérivée de [CoreState]. Si le fragment n'a pas d'état à mémoriser comme ici, on se contente de rendre une instance de [CoreState]. Il ne faut pas rendre `null` car cela amènerait ultérieurement à un plantage ;
- lignes 8-11 : doivent rendre le n° de la vue. Ici le fragment [MeteoFragment] a le n° 0 ;
- ligne 13-16 : servent à initialiser le fragment une fois qu'il a été construit (`previousState==null`) ou reconstruit (`previousState!=null`). Ici, il n'y a rien à faire. Le seul champ susceptible d'initialisation est le suivant :

```

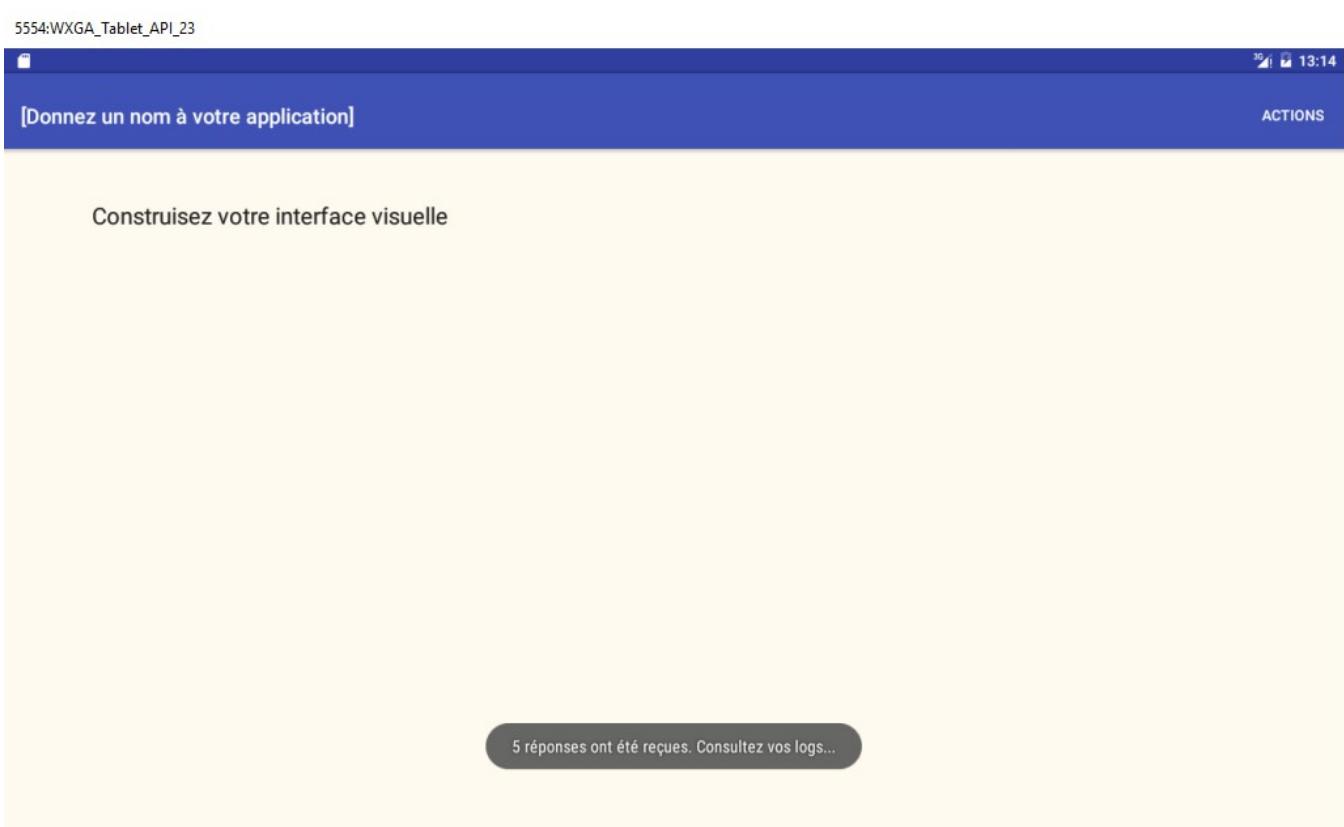
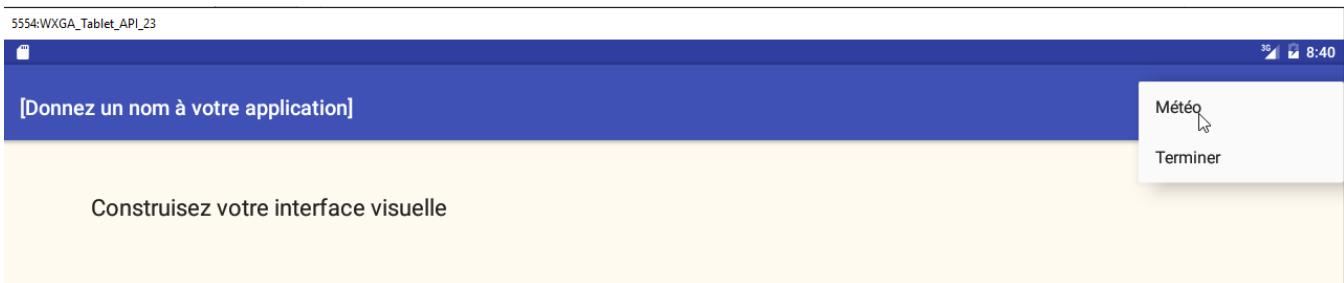
1.  // villes dont on veut la météo
2.  final String[] paysDeLoire = new String[]{"angers", "le mans", "nantes", "laval", "la roche sur yon"};

```

- mais il s'initialise tout seul ;
- lignes 18-24 : servent à initialiser la vue associée au fragment une fois qu'elle a été construite (`previousState==null`) ou reconstruite (`previousState!=null`) ;
 - lignes 21-23 : si c'est la 1ère visite faite au fragment, on initialise son menu pour cacher l'option [Annuler] ;
 - lignes 27-30 : appelées si pour arriver au fragment, il y a eu navigation avec une action de type [SUBMIT]. Ici, il n'y a pas de navigation inter-fragments puisqu'il n'y a qu'un fragment ;
 - lignes 32-35 : appelées lors d'un cycle sauvegarde / restauration due à une rotation du périphérique ou une autre raison. Ici, comme on n'a pas sauvegardé d'état, il n'y a rien à faire ;
 - lignes 37-40 : appelées lorsque toutes les mises à jour précédentes ont été effectuées. Ici, il n'y a rien à faire ;

2.8.2.6 Tests

Nous exécutons maintenant l'exemple :



Les logs sont alors les suivants :

```
1. 07-23 13:24:30.899 2642-2642/client.android D/MainActivity_: constructeur
2. 07-23 13:24:30.945 2642-2642/client.android D/AbstractDao: constructeur, thread=main
3. 07-23 13:24:32.861 2642-2642/client.android D/client.android.dao.service.Dao_: afterInject
4. 07-23 13:24:32.950 2642-2642/client.android D/MainActivity_: onCreate
5. 07-23 13:24:32.951 2642-2642/client.android D/client.android.dao.service.Dao_: setTimeout thread=main, timeout=1000
6. 07-23 13:24:32.952 2642-2642/client.android D/client.android.dao.service.Dao_: setBasicAuthentication thread=main, isBasicAuthenticationNeeded=false
7. 07-23 13:24:33.041 2642-2642/client.android D/MainActivity_: adding loadingPanel
8. 07-23 13:24:33.043 2642-2642/client.android D/MeteoFragment_: constructeur
9. 07-23 13:24:33.044 2642-2642/client.android D/MainActivity_: navigation vers vue 0 sur action NONE
10. 07-23 13:24:33.044 2642-2642/client.android D/MainActivity_: onCreateActivity
11. 07-23 13:24:33.080 2642-2642/client.android D/MainActivity_: onResume
12. 07-23 13:24:33.325 2642-2642/client.android D/MeteoFragment_: onActivityCreated
13. 07-23 13:24:33.518 2642-2642/client.android D/MeteoFragment_: onCreateOptionsMenu
14. 07-23 13:24:33.518 2642-2642/client.android D/MeteoFragment_: getMenuOptionsStates(Menu)
15. 07-23 13:24:33.519 2642-2642/client.android D/MeteoFragment_: Nombre d'options de menu=4
16. 07-23 13:24:33.519 2642-2642/client.android D/MeteoFragment_: initFragment initView updateForFirstVisit
17. 07-23 13:24:33.519 2642-2642/client.android D/MeteoFragment_: initMenu
18. 07-23 13:24:33.557 2642-2642/client.android D/MeteoFragment_: session={"action":"NONE","coreStates": [{"@type":"CoreState","hasBeenVisited":false,"menuOptionsState":null}],"previousTab":0,"previousView":0}
19. 07-23 13:24:33.557 2642-2642/client.android D/MeteoFragment_: état précédent=null
```

```

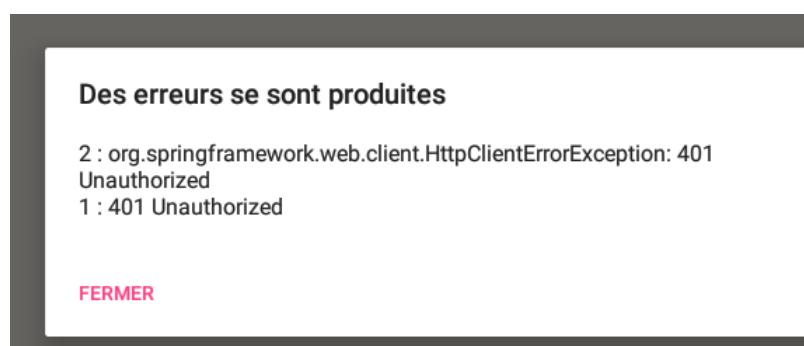
20. 07-23 13:24:33.558 2642-2642/client.android D/MeteoFragment_: notifyEndOfUpdates
21. 07-23 13:24:39.766 2642-2642/client.android D/MeteoFragment_: beginWaiting
22. 07-23 13:24:39.831 2642-2642/client.android D/client.android.dao.service.Dao_: getWeatherForecast city=angers, country=fr, APIID=aa6bb491c9a16810c4f0881f17e888c7, thread=main, timeout=1000
23. 07-23 13:24:39.831 2642-2642/client.android D/client.android.dao.service.Dao_: delay=5000
24. 07-23 13:24:39.882 2642-2642/client.android D/client.android.dao.service.Dao_: getWeatherForecast city=le mans, country=fr, APIID=aa6bb491c9a16810c4f0881f17e888c7, thread=main, timeout=1000
25. 07-23 13:24:39.882 2642-2642/client.android D/client.android.dao.service.Dao_: delay=5000
26. 07-23 13:24:39.885 2642-2642/client.android D/client.android.dao.service.Dao_: getWeatherForecast city=nantes, country=fr, APIID=aa6bb491c9a16810c4f0881f17e888c7, thread=main, timeout=1000
27. 07-23 13:24:39.885 2642-2642/client.android D/client.android.dao.service.Dao_: delay=5000
28. 07-23 13:24:39.886 2642-2642/client.android D/client.android.dao.service.Dao_: getWeatherForecast city=laval, country=fr, APIID=aa6bb491c9a16810c4f0881f17e888c7, thread=main, timeout=1000
29. 07-23 13:24:39.886 2642-2642/client.android D/client.android.dao.service.Dao_: delay=5000
30. 07-23 13:24:39.887 2642-2642/client.android D/client.android.dao.service.Dao_: getWeatherForecast city=la roche sur yon, country=fr, APIID=aa6bb491c9a16810c4f0881f17e888c7, thread=main, timeout=1000
31. 07-23 13:24:39.887 2642-2642/client.android D/client.android.dao.service.Dao_: delay=5000
32. 07-23 13:24:45.035 2642-2961/client.android D/client.android.dao.service.Dao_: response={"coord":{"lon":-1.55,"lat":47.22},"weather":[{"id":800,"main":"Clear","description":"clear sky","icon":"01d"}],"base":"cmc stations","main":{"temp":298.05,"pressure":1022,"humidity":47,"temp_min":297.15,"temp_max":299.15},"wind":{"speed":2.6,"deg":310},"clouds":{"all":0},"dt":1469277000,"sys": {"type":1,"id":5641,"message":0.0032,"country":"FR","sunrise":1469248505,"sunset":1469303378},"id":2990969,"name":"Nantes","cod":200} sur thread [RxIoScheduler-4]
33. 07-23 13:24:45.035 2642-2963/client.android D/client.android.dao.service.Dao_: response={} sur thread [RxIoScheduler-6]
34. 07-23 13:24:45.035 2642-2959/client.android D/client.android.dao.service.Dao_: response={} sur thread [RxIoScheduler-2]
35. 07-23 13:24:45.035 2642-2962/client.android D/client.android.dao.service.Dao_: response={} sur thread [RxIoScheduler-5]
36. 07-23 13:24:45.036 2642-2960/client.android D/client.android.dao.service.Dao_: response={} sur thread [RxIoScheduler-3]
37. 07-23 13:24:45.039 2642-2642/client.android D/MeteoFragment_: thread=main, response={"coord":{"lon":-1.55,"lat":47.22},"weather":[{"id":800,"main":"Clear","description":"clear sky","icon":"01d"}],"base":"cmc stations","main":{"temp":298.05,"pressure":1022,"humidity":47,"temp_min":297.15,"temp_max":299.15},"wind":{"speed":2.6,"deg":310},"clouds":{"all":0},"dt":1469277000,"sys": {"type":1,"id":5641,"message":0.0032,"country":"FR","sunrise":1469248505,"sunset":1469303378},"id":2990969,"name":"Nantes","cod":200}
38. 07-23 13:24:45.039 2642-2642/client.android D/MeteoFragment_: thread=main, response={}
39. 07-23 13:24:45.039 2642-2642/client.android D/MeteoFragment_: thread=main, response={}
40. 07-23 13:24:45.039 2642-2642/client.android D/MeteoFragment_: thread=main, response={}
41. 07-23 13:24:45.039 2642-2642/client.android D/MeteoFragment_: thread=main, response={}
42. 07-23 13:24:45.039 2642-2642/client.android D/MeteoFragment_: initMenu

```

- lignes 32-36 : les réponses JSON sont obtenues sur des threads d'E/S
- lignes 37-41 : le fragment récupère les 5 réponses sur le thread de l'Ui ;

Maintenant, on fait la requête avec un identifiant API incorrect :

```
String APIID = "";
```



Les logs sont alors les suivants :

```

1. 07-23 13:34:43.853 11240-11240/client.android D/MeteoFragment_: beginWaiting
2. ...
3. 07-23 13:34:49.121 11240-11464/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-2], Exception communication avec serveur : [org.springframework.web.client.HttpClientErrorException,["401 Unauthorized"]]
4. 07-23 13:34:49.121 11240-11466/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-4], Exception communication avec serveur : [org.springframework.web.client.HttpClientErrorException,["401 Unauthorized"]]
5. 07-23 13:34:49.162 11240-11468/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-6], Exception communication avec serveur : [org.springframework.web.client.HttpClientErrorException,["401 Unauthorized"]]
6. 07-23 13:34:49.162 11240-11467/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-5], Exception communication avec serveur : [org.springframework.web.client.HttpClientErrorException,["401 Unauthorized"]]
7. 07-23 13:34:49.163 11240-11240/client.android D/MeteoFragment_: Exception recue
8. 07-23 13:34:49.163 11240-11240/client.android D/MeteoFragment_: Annulation des tâches lancées
9. 07-23 13:34:49.163 11240-11240/client.android D/MeteoFragment_: initMenu
10. 07-23 13:34:49.167 11240-11465/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-3], Exception communication avec serveur : [org.springframework.web.client.HttpClientErrorException,["401 Unauthorized"]]

```

- lignes 3-6, 10 : les 5 appels HTTP ont générés 5 exceptions ;
- ligne 7 : le fragment [MeteoFragment] reçoit la 1ère exception. Il va alors annuler toutes les tâches ;

Maintenant mettons un temps d'attente de 5 secondes [IMainActivity.DELAY] et annulons l'opération. Les logs sont alors les suivants :

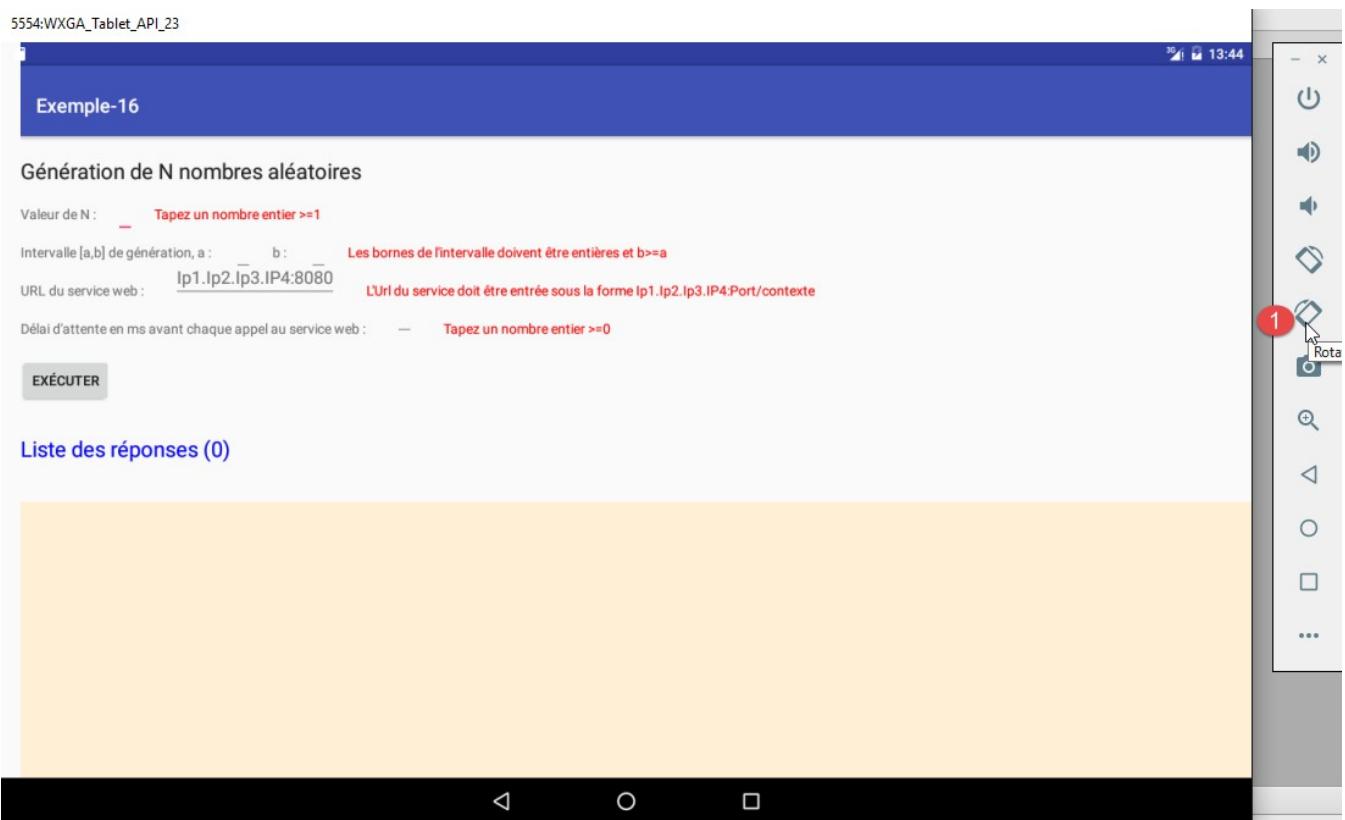
```

1. 07-21 13:16:20.329 20390-20390/client.android D/MeteoFragment_: beginWaiting
2. ...
3. 07-21 13:16:23.635 20390-20390/client.android D/MeteoFragment_: Annulation demandée
4. 07-21 13:16:23.635 20390-20390/client.android D/MeteoFragment_: Annulation des tâches lancées
5. 07-21 13:16:23.635 20390-20390/client.android D/MeteoFragment_: initMenu
6. 07-21 13:25:02.948 29965-30197/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-6], Exception communication avec serveur : [java.lang.InterruptedException,[null]]
7. 07-21 13:25:02.948 29965-30195/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-4], Exception communication avec serveur : [java.lang.InterruptedException,[null]]
8. 07-21 13:25:02.948 29965-30194/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-3], Exception communication avec serveur : [java.lang.InterruptedException,[null]]
9. 07-21 13:25:02.951 29965-30193/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-2], Exception communication avec serveur : [java.lang.InterruptedException,[null]]
10. 07-21 13:25:02.951 29965-30196/client.android D/client.android.dao.service.Dao_: Thread [RxIoScheduler-5], Exception communication avec serveur : [java.lang.InterruptedException,[null]]
```

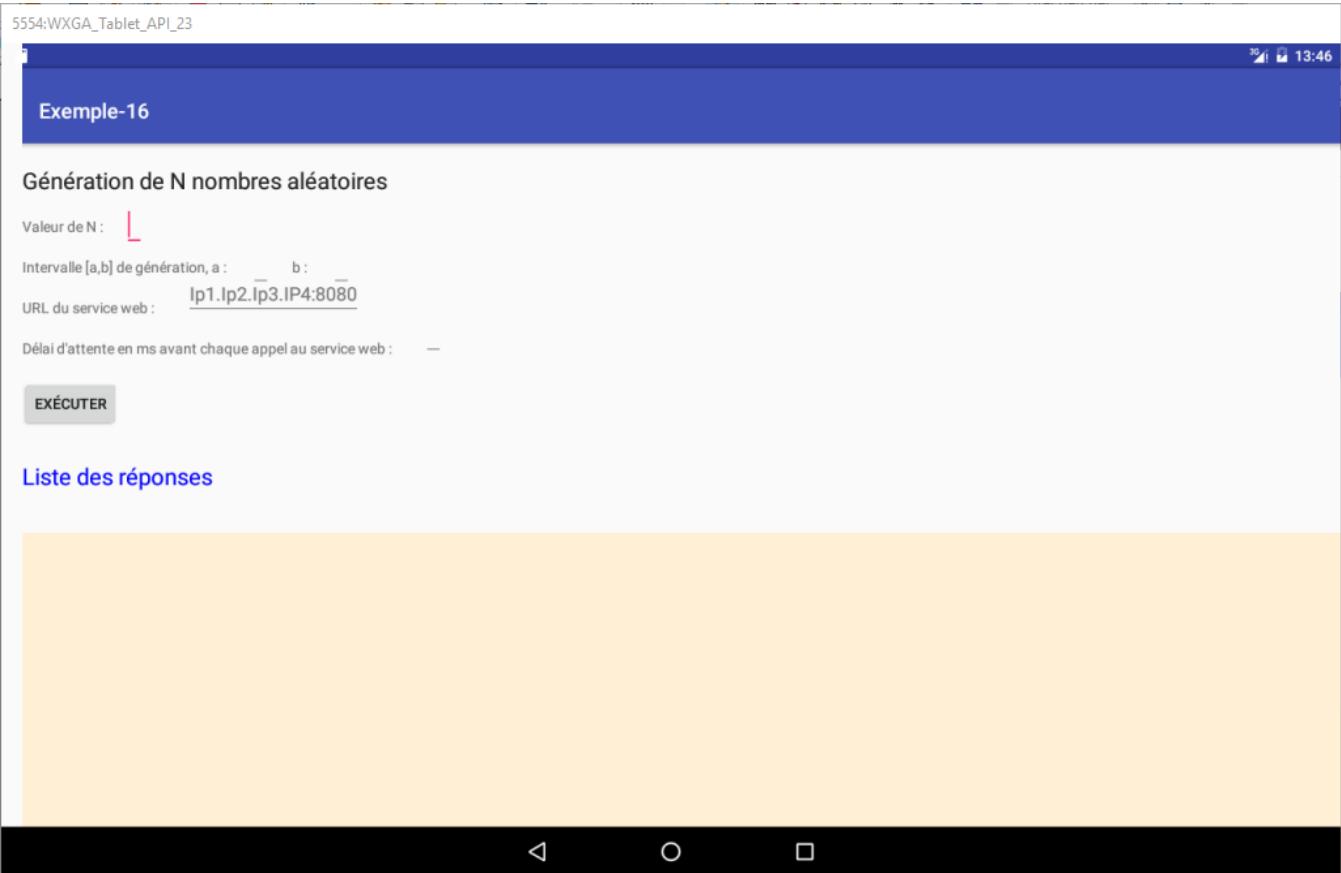
- ligne 3 : demande d'annulation ;
- ligne 4 : l'attente est annulée parce qu'une annulation a eu lieu ;
- lignes 6-10 : l'annulation des tâches provoque une exception sur chacun des threads des cinq tâches. Le type d'exception dépend des applications. L'exception est ici [java.lang.InterruptedException] parce que les tâches ont été interrompues pendant qu'elles exécutaient l'instruction [Thread.sleep(delay)] qui les fait attendre artificiellement [delay] millisecondes ;

2.8.3 Exemple-16B

Nous refactorisons ici l'exemple 16 du paragraphe 1.17, page 201. Il présente un fragment qui fait des appels asynchrones à un serveur de nombres aléatoires. Voyons comment il se comporte lors d'une rotation du périphérique :



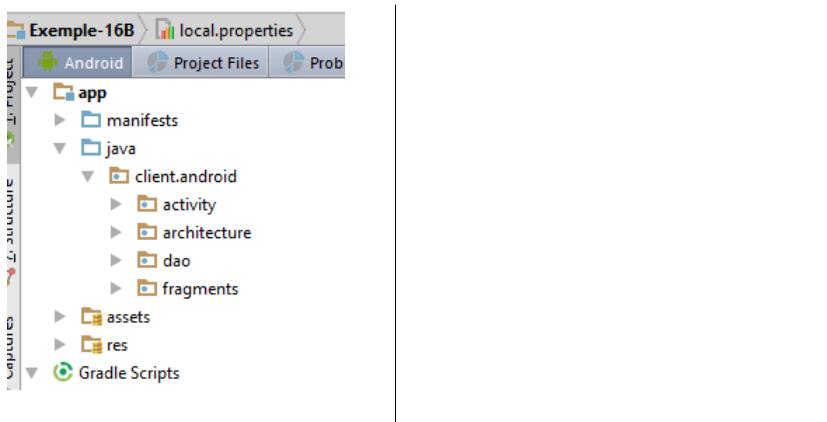
- en [1], on fait tourner le périphérique deux fois ;



On voit qu'on a perdu tous les messages d'erreur. Nous allons essayé d'améliorer cela.

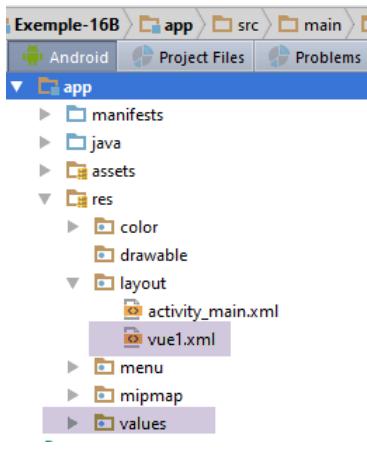
2.8.3.1 Le projet Exemple-16B

Nous copions le projet [client-android-skel] dans le projet [exemples/Exemple-16B] puis nous chargeons le nouveau projet :



Du projet initial [Exemple-16], nous copions dans [Exemple-16B] les éléments suivants :

- le fichier [res/layout/vue1.xml], le dossier [res/values] :



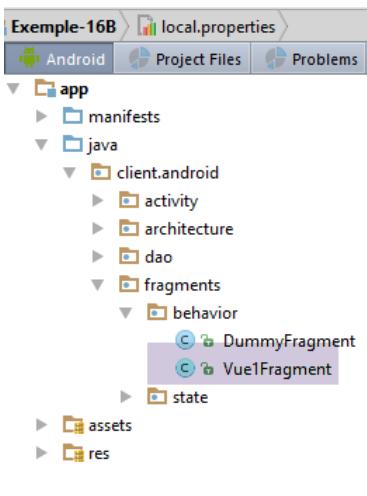
On modifiera la marge haute de la vue [vue1.xml] à 80 dp :

```

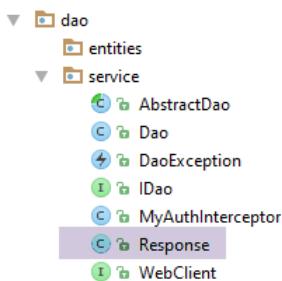
1. <TextView
2.     android:id="@+id/txt_Titre2"
3.     android:layout_width="wrap_content"
4.     android:layout_height="wrap_content"
5.     android:layout_marginTop="80dp"
6.     android:text="@string/aleas"
7.     android:textAppearance="?android:attr/textAppearanceLarge" />

```

- le fragment [Vue1Fragment] :



- la classe [dao / service / Response] :



A ce stade, on peut tenter une 1ère compilation :

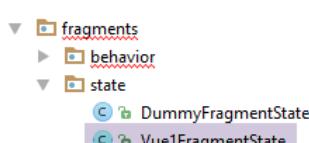
- un premier type d'erreurs est celui des *imports*. Des classes ont changé de package dans la migration vers [Exemple-16B]. On commence par corriger ce type d'erreurs ;
- un second type d'erreur est signalée sur la classe [Vue1Fragment] parce qu'elle n'implémente pas les méthodes imposées par la classe parent [AbstractParent]. On fait une génération automatique de celles-ci ;

On tente une seconde compilation :

- toutes les erreurs restantes sont désormais concentrées sur la classe [Vue1Fragment], la classe qui va subir le plus de modifications ;

2.8.3.2 Crédit d'un état pour le fragment [Vue1Fragment]

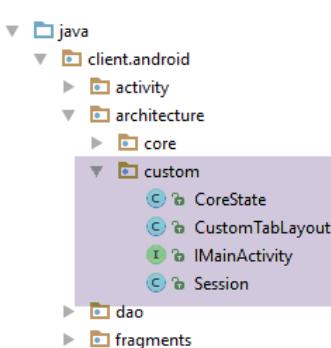
Nous avons vu que certaines informations du fragment allaient devoir être sauvegardées lors d'une rotation afin de restaurer le fragment tel qu'il était avant la rotation. Nous créons donc un état [Vue1FragmentState] vide pour le moment :



```

1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. public class Vue1FragmentState extends CoreState {
6.
7. }
```

2.8.3.3 Personnalisation du projet



L'interface [IMainActivity] permet de spécifier certaines caractéristiques du projet :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.ISession;
4. import client.android.dao.service.IDao;
5.
6. public interface IMainActivity extends IDao {
7.
8.     // accès à la session
9.     ISession getSession();
10.
11.    // changement de vue
12.    void navigateToView(int position, ISession.Action action);
13.
14.    // gestion de l'attente
15.    void beginWaiting();
16.
17.    void cancelWaiting();
18.
19.    // constantes de l'application -----
20.}
```

```

21. // mode debug
22. boolean IS_DEBUG_ENABLED = true;
23.
24. // délai maximal d'attente de la réponse du serveur
25. int TIMEOUT = 1000;
26.
27. // délai d'attente avant exécution de la requête client
28. int DELAY = 5000;
29.
30. // authentification basique
31. boolean IS_BASIC_AUTHENTICATION_NEEDED = false;
32.
33. // adjacence des fragments
34. int OFF_SCREEN_PAGE_LIMIT = 1;
35.
36. // barre d'onglets
37. boolean ARE_TABS_NEEDED = false;
38.
39. // image d'attente
40. boolean IS_WAITING_ICON_NEEDED = true;
41.
42. // nombre de fragments de l'application
43. int FRAGMENTS_COUNT = 1;
44.
45. }

```

- lignes 25, 28, 31, 40 : caractéristiques de la couche [DAO]. Il n'y a pas besoin d'authentification basique ;
- ligne 34 : adjacence des fragments. Ici cette constante n'a pas d'importance puisqu'il n'y a qu'un fragment ;
- ligne 37 : ce n'est pas une application à onglets ;
- ligne 43 : il n'y a qu'un fragment ;

La classe [CoreState] qui mémorise l'état des fragments sera la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.MenuInstanceState;
4. import client.android.fragments.state.Vue1FragmentState;
5. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
6. import com.fasterxml.jackson.annotation.JsonSubTypes;
7. import com.fasterxml.jackson.annotation.JsonProperty;
8.
9. @JsonIgnoreProperties(ignoreUnknown = true)
10. @JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
11. @JsonSubTypes({
12.     @JsonSubTypes.Type(value = Vue1FragmentState.class)
13. })
14. public class CoreState {
15.     // fragment visité ou non
16.     protected boolean hasBeenVisited = false;
17.     // état de l'éventuel menu du fragment
18.     protected MenuInstanceState[] menuOptionsState;
19.
20.     // getters et setters
21.     ...
22. }

```

- ligne 12 : nous déclarons la classe de l'état du fragment [Vue1Fragment] ;

La classe [Session] est la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.AbstractSession;
4.
5. public class Session extends AbstractSession {
6.     // les éléments qui ne peuvent être serialisés en JSON doivent avoir l'annotation @JsonIgnore
7. }

```

Elle est vide puisque dans cette application, il n'y a pas de communication inter-fragments.

2.8.3.4 La couche [DAO]



Dans la couche [DAO], trois classes doivent être personnalisées :

- l'interface **IDao** ;
- son implémentation **Dao** ;
- l'interface **WebClient** de dialogue avec le serveur web / JSON ;

La classe [Response] vient du projet [Exemple-16] qui l'utilise :

```

1. package client.android.dao.service;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.     // ----- propriétés
8.     // statut de l'opération
9.     private int status;
10.    // les éventuels messages d'erreur
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.    ...
28. }
```

L'interface [WebClient] sera la suivante :

```

1. package client.android.dao.service;
2.
3. import org.androidannotations.rest.spring.annotations.Get;
4. import org.androidannotations.rest.spring.annotations.Path;
5. import org.androidannotations.rest.spring.annotations.Rest;
6. import org.androidannotations.rest.spring.api.RestClientRootUrl;
7. import org.androidannotations.rest.spring.api.RestClientSupport;
8. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
9. import org.springframework.web.client.RestTemplate;
10.
11. @Rest(converters = {MappingJackson2HttpMessageConverter.class})
12. public interface WebClient extends RestClientRootUrl, RestClientSupport {
13.
14.     // RestTemplate
15.     void setRestTemplate(RestTemplate restTemplate);
16.
17.     // 1 nombre aléatoire dans l'intervalle [a,b]
18.     @Get("/{a}/{b}")
19.     Response<Integer> getAlea(@Path("a") int a, @Path("b") int b);
20.
21. }
```

- lignes 18-19 : l'URL du service des nombres aléatoires. On rappelle que celle-ci est relative à l'URL racine (RestClientRootUrl, ligne 12) du client. Ici cette URL racine sera [http://localhost:8080];

L'interface [IDao] sera la suivante :

```
1. package client.android.dao.service;
2.
3. import rx.Observable;
4.
5. public interface IDao {
6.     // Url du service web
7.     void setUrlServiceWebJson(String url);
8.
9.     // utilisateur
10.    void setUser(String user, String mdp);
11.
12.    // timeout du client
13.    void setTimeout(int timeout);
14.
15.    // authentification basique
16.    void setBasicAuthentification(boolean isBasicAuthentificationNeeded);
17.
18.    // mode debug
19.    void setDebugMode(boolean isDebugEnabled);
20.
21.    // délai d'attente en millisecondes du client avant requête
22.    void setDelay(int delay);
23.
24.    // service de nombres aléatoires
25.    Observable<Response<Integer>> getAlea(int a, int b);
26.
27. }
```

- on rappelle que les méthodes des lignes 6-22 sont présentes par défaut dans l'interface IDao du projet [client-android-skel] ;
- ligne 25 : la méthode [getAlea] permet d'obtenir un nombre aléatoire dans l'intervalle [a,b]. Ce nombre est obtenu dans une réponse de type [Response<Integer>] où le nombre aléatoire est dans le champ [body] de ce type ;

L'interface [IDao] est implémentée par la classe [Dao] suivante :

```
1. package client.android.dao.service;
2.
3. import android.util.Log;
4. import org.androidannotations.annotations.AfterInject;
5. import org.androidannotations.annotations.Bean;
6. import org.androidannotations.annotations.EBean;
7. import org.androidannotations.rest.spring.annotations.RestService;
8. import org.springframework.http.client.ClientHttpRequestInterceptor;
9. import org.springframework.http.client.SimpleClientHttpRequestFactory;
10. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
11. import org.springframework.web.client.RestTemplate;
12. import rx.Observable;
13.
14. import java.util.ArrayList;
15. import java.util.List;
16.
17. @EBean(scope = EBean.Scope.Singleton)
18. public class Dao extends AbstractDao implements IDao {
19.
20.     // client du service web
21.     @RestService
22.     protected WebClient webClient;
23.     // sécurité
24.     @Bean
25.     protected MyAuthInterceptor authInterceptor;
26.     // le RestTemplate
27.     private RestTemplate restTemplate;
28.     // factory du RestTemplate
29.     private SimpleClientHttpRequestFactory factory;
30.
31.     @AfterInject
32.     public void afterInject() {
33.         // log
34.         Log.d(className, "afterInject");
35.         // on construit le restTemplate
36.         factory = new SimpleClientHttpRequestFactory();
37.         restTemplate = new RestTemplate(factory);
38.         // on fixe le convertisseur JSON
39.         restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
40.         // on fixe le restTemplate du client web
41.         webClient.setRestTemplate(restTemplate);
42.     }
43.
44.     @Override
45.     public void setUrlServiceWebJson(String url) {
```

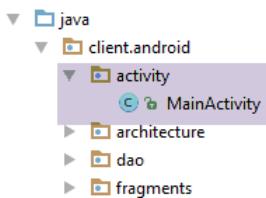
```

46.     // on fixe l'URL du service web
47.     webClient.setRootUrl(url);
48. }
49.
50. @Override
51. public void setUser(String user, String mdp) {
52.     // on enregistre l'utilisateur dans l'intercepteur
53.     authInterceptor.setUser(user, mdp);
54. }
55.
56. @Override
57. public void setTimeout(int timeout) {
58.     if (isDebugEnabled) {
59.         Log.d(className, String.format("setTimeout thread=%s, timeout=%s", Thread.currentThread().getName(), timeout));
60.     }
61.     // configuration factory
62.     factory.setReadTimeout(timeout);
63.     factory.setConnectTimeout(timeout);
64. }
65.
66. @Override
67. public void setBasicAuthentification(boolean isBasicAuthentificationNeeded) {
68.     if (isDebugEnabled) {
69.         Log.d(className, String.format("setBasicAuthentification thread=%s, isBasicAuthentificationNeeded=%s",
70.             Thread.currentThread().getName(), isBasicAuthentificationNeeded));
71.     }
72.     // intercepteur d'authentification ?
73.     if (isBasicAuthentificationNeeded) {
74.         // on ajoute l'intercepteur d'authentification
75.         List<ClientHttpRequestInterceptor> interceptors = new ArrayList<ClientHttpRequestInterceptor>();
76.         interceptors.add(authInterceptor);
77.         restTemplate.setInterceptors(interceptors);
78.     }
79.
80.     // méthodes privées -----
81.     private void log(String message) {
82.         if (isDebugEnabled) {
83.             Log.d(className, message);
84.         }
85.     }
86.
87.     // service de nombres aléatoires
88.     @Override
89.     public Observable<Response<Integer>> getAlea(final int a, final int b) {
90.         // exécution client web
91.         return getResponse(new IRequest<Response<Integer>>() {
92.             @Override
93.             public Response<Integer> getResponse() {
94.                 return webClient.getAlea(a, b);
95.             }
96.         });
97.     }
98. }
99. }

```

- on rappelle que les lignes 17-85 sont présentes par défaut dans la classe [Dao] du projet [client-android-skel]. Il faut juste ajouter les méthodes d'implémentation de l'interface [IDao] ;
- lignes 88-97 : implémentation de la méthode [getAlea]. Celle-ci est très simple et réalisée en 6 lignes, lignes 91-96 ;
- ligne 91 : la méthode [getResponse] est une méthode de la classe parent [AbstractDao]. Elle attend un paramètre de type [IRequest<T>] où T est le type de la réponse attendue, ici un type *Response<Integer>*. Le type T de [IRequest<T>] (ligne 91) doit être le type T de la méthode [Observable<T> getAlea] (ligne 89) ;
- l'interface [IRequest<T>] n'a qu'une méthode : *getResponse*. Celle-ci a pour rôle de fournir la réponse de type T que doit rendre la méthode [Observable<T> getAlea] ;
- ligne 94 : c'est l'interface [WebClient] qui fournit cette réponse. On lui passe les deux paramètres reçus ligne 89. Pour cette raison, ceux-ci doivent avoir l'attribut **final** ;

2.8.3.5 L'activité [MainActivity]

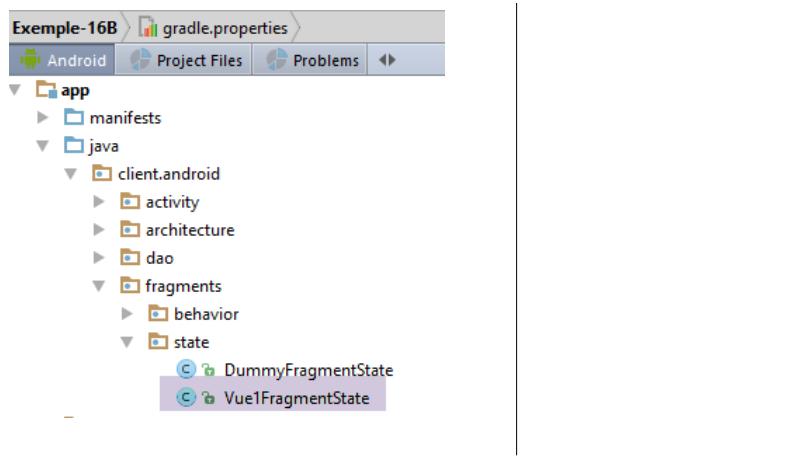


L'activité [MainActivity] est la suivante :

```
1. package client.android.activity;
2.
3. import android.util.Log;
4. import client.android.R;
5. import client.android.architecture.core.AbstractActivity;
6. import client.android.architecture.core.AbstractFragment;
7. import client.android.architecture.core.ISession;
8. import client.android.dao.Dao;
9. import client.android.dao.service.IDao;
10. import client.android.dao.service.Response;
11. import client.android.fragments.behavior.Vue1Fragment_;
12. import org.androidannotations.annotations.Bean;
13. import org.androidannotations.annotations.EActivity;
14. import org.androidannotations.annotations.OptionsMenu;
15. import rx.Observable;
16.
17. @EActivity
18. @OptionsMenu(R.menu.menu_main)
19. public class MainActivity extends AbstractActivity {
20.
21.     // couche [DAO]
22.     @Bean(Dao.class)
23.     protected IDao dao;
24.
25.     // méthodes classe parent -----
26.     @Override
27.     protected void onCreateActivity() {
28.         // log
29.         if (IS_DEBUG_ENABLED) {
30.             Log.d(className, "onCreateActivity");
31.         }
32.         // on continue les initialisations commencées par la classe parent
33.     }
34.
35.     @Override
36.     protected IDao getDao() {
37.         return dao;
38.     }
39.
40.     @Override
41.     protected AbstractFragment[] getFragments() {
42.         // définir les fragments ici
43.         return new AbstractFragment[]{new Vue1Fragment_()};
44.     }
45.
46.     @Override
47.     protected CharSequence getFragmentTitle(int position) {
48.         // définir les titres des fragments ici
49.         return null;
50.     }
51.
52.
53.     @Override
54.     protected void navigateOnTabSelected(int position) {
55.         // navigation par onglets - définir la vue à afficher
56.     }
57.
58.     @Override
59.     protected int getFirstView() {
60.         return 0;
61.     }
62.
63.     // interface IDao -----
64.     @Override
65.     public Observable<Response<Integer>> getAlea(int a, int b) {
66.         return dao.getAlea(a, b);
67.     }
68.
69. }
```

- on rappelle que les lignes 15-61 sont présentes par défaut dans le projet [client-android-skel]. Il faut juste les personnaliser ;
- lignes 40-44 : le tableau des fragments. Il n'y en a qu'un ici ;
- lignes 47-51 : pas de titres de fragments nécessaires ;
- lignes 53-56 : pas d'onglets ici ;
- lignes 58-61 : la 1ère vue à afficher est la vue n° 0, celle de [Vue1Fragment] ;
- lignes 64-67 : implémentation de l'interface [IDao]. Ici, il n'y a rien à faire d'autre qu'à déléguer le travail à la couche [DAO] de la ligne 23 ;

2.8.3.6 L'état du fragment [Vue1Fragment]



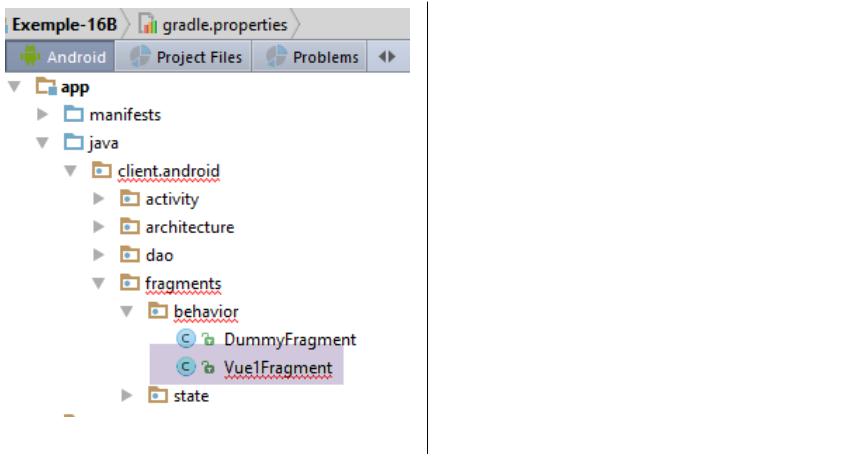
La classe [Vue1FragmentState] sera la suivante :

```

1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. import java.util.ArrayList;
6. import java.util.List;
7.
8. public class Vue1FragmentState extends CoreState {
9.
10.    // état fragment -----
11.    // liste des réponses
12.    private List<String> reponses = new ArrayList<>();
13.    // état vue -----
14.    // msg d'erreur sur le nombre de nombres aléatoires demandés
15.    private boolean txtErrorAleasVisible = false;
16.    // msg d'erreur sur l'intervalle [a,b] de génération
17.    private boolean txtErrorIntervalVisible = false;
18.    // msg d'erreur sur l'URL du service web
19.    private boolean txtMsgErreurUrlServiceWebVisible = false;
20.    // msg d'erreur sur la durée d'attente
21.    private boolean txtViewErreurDelayVisible = false;
22.    // état visible ou non du bouton Exécuter
23.    private boolean btnExecuterVisible = true;
24.
25.    // getters et setters
26.    ...
27. }
```

Pour obtenir ce qu'il fallait mémoriser dans le fragment, on a fait faire des rotations au périphérique dans diverses situations et on a regardé ce qui avait disparu à la restauration. On est arrivé à la conclusion qu'il fallait mémoriser les informations des lignes 10-23.

2.8.3.7 Le fragment [Vue1Fragment]



Actuellement la vue [Vue1Fragment] présente diverses erreurs dues au fait que la classe parent [AbstractFragment] dont elle dérive a changé. Plutôt que de décrire un à un les changements à faire, nous allons commenter directement la version finale.

Le squelette du fragment est le suivant :

```

1. package client.android.fragments.behavior;
2.
3. import android.util.Log;
4. import android.view.View;
5. import android.widget.*;
6. import client.android.R;
7. import client.android.architecture.core.AbstractFragment;
8. import client.android.architecture.custom.CoreState;
9. import client.android.dao.service.Response;
10. import client.android.fragments.state.Vue1FragmentState;
11. import com.fasterxml.jackson.core.JsonProcessingException;
12. import com.fasterxml.jackson.databind.ObjectMapper;
13. import org.androidannotations.annotations.Click;
14. import org.androidannotations.annotations.EFragment;
15. import org.androidannotations.annotations.OptionsMenu;
16. import org.androidannotations.annotations.ViewById;
17. import rx.Observable;
18. import rx.functions.Action1;
19.
20. import java.net.URI;
21. import java.net.URISyntaxException;
22. import java.util.ArrayList;
23. import java.util.List;
24.
25. @EFragment(R.layout.vue1)
26. @OptionsMenu(R.menu.menu_vue1)
27. public class Vue1Fragment extends AbstractFragment {
28.
29. ...
30. }
```

- ligne 26 on rappelle que tout fragment doit avoir un menu, même vide. C'est le cas ici.

2.8.3.7.1 Gestion du clic sur le bouton [Exécuter]

```

1.     @Click(R.id.btn_Executer)
2.     protected void doExecuter() {
3.         // on vérifie les données saisies
4.         if (!isPageValid()) {
5.             return;
6.         }
7.         // on efface les réponses précédentes
8.         reponses.clear();
9.         dataAdapterReponses.notifyDataSetChanged();
10.        // on remet à 0 le compteur de réponses
11.        nbReponses = 0;
12.        infoReponses.setText("Liste des réponses (0)");
13.        // initialisation activité
14.        mainActivity.setUrlServiceWebJson(urlServiceWebJson);
15.        mainActivity.setDelay(delay);
16.        // on prépare la tâche aléatoire
17.        beginWaiting(1);
18.        // on demande les nombres aléatoires
19.        getAleasInBackground(nbAleas, a, b);
```

```

20. }
21.
22. void getAleasInBackground(int nbAleas, int a, int b) {
23.     // on crée le processus à observer
24.     Observable<Response<Integer>> process = Observable.empty();
25.     for (int i = 0; i < nbAleas; i++) {
26.         process = process.mergeWith(mainActivity.getAlea(a, b));
27.     }
28.     // on demande les nombres aléatoires
29.     executeInBackground(process, new Action1<Response<Integer>>() {
30.
31.         @Override
32.         public void call(Response<Integer> response) {
33.             // on consomme la réponse
34.             consumeAleaResponse(response);
35.         }
36.     });
37. }
38.
39. protected void consumeAleaResponse(Response<Integer> response) {
40.     // log
41.     if (isDebugEnabled) {
42.         try {
43.             Log.d(String.format("%s", className), String.format("consumeAleaResponse(%s)", jsonMapper.writeValueAsString(response)));
44.         } catch (JsonProcessingException e) {
45.             e.printStackTrace();
46.         }
47.     }
48.     // une réponse de +
49.     nbReponses++;
50.     infoReponses.setText(String.format("Liste des réponses (%s)", nbReponses));
51.     // on analyse la réponse
52.     // erreur ?
53.     if (response.getStatus() != 0) {
54.         // affichage
55.         showAlert(response.getMessages());
56.         // annulation
57.         doAnnuler();
58.         // retour à l'Ui
59.         return;
60.     }
61.     // on ajoute l'information à la liste des reponses
62.     reponses.add(0, String.valueOf(response.getBody()));
63.     // on rafraîchit les reponses
64.     dataAdapterReponses.notifyDataSetChanged();
65. }
66.
67. // annulation -----
68. @Click(R.id.btn_Annuler)
69. protected void doAnnuler() {
70.     if (isDebugEnabled) {
71.         Log.d(className, "Annulation demandée");
72.     }
73.     // on annule les tâches asynchrones
74.     cancelRunningTasks();
75. }
76.
77. private void beginWaiting(int nbRunningTasks) {
78.     // on met le sablier
79.     beginRunningTasks(1);
80.     // le bouton [Annuler] remplace le bouton [Exécuter]
81.     btnExecuter.setVisibility(View.INVISIBLE);
82.     btnAnnuler.setVisibility(View.VISIBLE);
83. }

```

- lignes 4-6 : on vérifie d'abord que les saisies sont valides. Des messages d'erreur peuvent alors apparaître ;
- lignes 8-9 : la liste des réponses est vidée. On répercute ce changement sur le *ListView* qui les affiche ;
- lignes 11-12 : le nombre de réponses reçues est mis à zéro ;
- ligne 14 : on fixe l'URL du service des nombres aléatoires. Cette information va être transmise à la couche [DAO] ;
- ligne 15 : on fixe le délai d'attente avant de faire la requête au service des nombres aléatoires. Cette information va être transmise à la couche [DAO] ;
- ligne 17 : on se prépare à lancer 1 tâche asynchrone (et non pas N, on verra pourquoi) ;
- lignes 24-27 : des N tâches asynchrones, on en fait une par une suite d'opérations [merge] ;
- lignes 29-36 : on demande à la classe parent [AbstractParent] d'interroger le service web / JSON de nombres aléatoires ;
- lignes 29-36 : la méthode [executeInBackground] attend deux paramètres :
 - ligne 29 : le processus à observer et exécuter est celui qui a été calculé dans les lignes qui précédent ;
 - lignes 29-36 : l'instance [Action1] à exécuter lorsqu'on reçoit la réponse du service asynchrone. Le type T de [Action1<T>] doit être le type T du résultat de la méthode [getAlea], c-à-d un type [Response<Integer>] ;
- ligne 34 : lorsqu'une réponse arrive (un nombre aléatoire), on la consomme dans la méthode de la ligne 39 ;
- lignes 49-50 : on note et on signale qu'on a reçu une nouvelle réponse ;

- lignes 53-60 : le type [Response<T>] a un champ [status] qui est un code d'erreur. Si ce code est différent de zéro, alors le serveur a rencontré un problème ;
- ligne 55 : un message d'erreur est affiché. La méthode [showAlert] appartient à la classe parent ;
- ligne 57 : la méthode des lignes 68-75 est appelée. Elle va annuler les tâches encore actives (ligne 74) ;
- ligne 62 : la réponse est ajoutée à la liste des réponses qui est la source de données du *ListView* ;
- ligne 64 : le *ListView* est rafraîchi ;
- lignes 77-83 : la méthode [beginWaiting(int nbRunningTasks)] prépare la vue pour l'attente (lignes 81-82) et transmet à la classe parent que [nbRunningTasks] tâches vont bientôt s'exécuter (ligne 79) ;

2.8.3.7.2 Le cycle de vie du fragment

Le cycle de vie du fragment est assuré par les méthodes suivantes :

```

1.    // données locales
2.    private List<String> reponses;
3.    private ArrayAdapter<String> dataAdapterReponses;
4.    private int nbReponses = 0;
5.    ...
6.    // gestion du cycle de vie -----
7.    @Override
8.    public CoreState saveFragment() {
9.        // état actuel de la vue
10.       Vue1FragmentState state = new Vue1FragmentState();
11.       state.setTextViewErreurDelayVisible(textViewErreurDelay.getVisibility() == View.VISIBLE);
12.       state.setTxtErrorAleasVisible(txtErrorAleas.getVisibility() == View.VISIBLE);
13.       state.setTxtMsgErreurUrlServiceWebVisible(txtMsgErreurUrlServiceWeb.getVisibility() == View.VISIBLE);
14.       state.setTxtErrorIntervalleVisible(txtErrorIntervalle.getVisibility() == View.VISIBLE);
15.       state.setBtnExecuterVisible(btnExecuter.getVisibility() == View.VISIBLE);
16.       state.setReponses(reponses);
17.       return state;
18.    }
19.
20.    @Override
21.    protected int getNumView() {
22.        return 0;
23.    }
24.
25.    @Override
26.    protected void initFragment(CoreState previousState) {
27.        // 1ère visite ?
28.        if (previousState != null) {
29.            Vue1FragmentState state = (Vue1FragmentState) previousState;
30.            reponses = state.getReponses();
31.        } else {
32.            reponses = new ArrayList<>();
33.        }
34.        // source de données du listView
35.        dataAdapterReponses = new ArrayAdapter<>(activity, android.R.layout.simple_list_item_1, android.R.id.text1,
36.        reponses);
37.        // nre de réponses
38.        nbReponses = reponses.size();
39.    }
40.
41.    @Override
42.    protected void initView(CoreState previousState) {
43.        // lien listview / adaptateur
44.        listReponses.setAdapter(dataAdapterReponses);
45.        // 1ère visite ?
46.        if (previousState == null) {
47.            txtErrorAleas.setVisibility(View.INVISIBLE);
48.            txtErrorIntervalle.setVisibility(View.INVISIBLE);
49.            txtMsgErreurUrlServiceWeb.setVisibility(View.INVISIBLE);
50.            textViewErreurDelay.setVisibility(View.INVISIBLE);
51.            // les boutons
52.            btnAnnuler.setVisibility(View.INVISIBLE);
53.            btnExecuter.setVisibility(View.VISIBLE);
54.        }
55.    }
56.
57.    @Override
58.    protected void updateOnSubmit(CoreState previousState) {
59.    }
60.
61.
62.    @Override
63.    protected void updateOnRestore(CoreState previousState) {
64.        // état précédent de la vue
65.        Vue1FragmentState state = (Vue1FragmentState) previousState;
66.        // on montre / cache les msg d'erreur
67.        txtErrorAleas.setVisibility(state.isTxtErrorAleasVisible() ? View.VISIBLE : View.INVISIBLE);
68.        txtErrorIntervalle.setVisibility(state.isTxtErrorIntervalleVisible() ? View.VISIBLE : View.INVISIBLE);
69.        txtMsgErreurUrlServiceWeb.setVisibility(state.isTxtMsgErreurUrlServiceWebVisible() ? View.VISIBLE : View.INVISIBLE);

```

```

70.     textViewErreurDelay.setVisibility(state.isTextViewErreurDelayVisible() ? View.VISIBLE : View.INVISIBLE);
71.     // boutons
72.     btnAnnuler.setVisibility(state.isBtnExecuterVisible() ? View.INVISIBLE : View.VISIBLE);
73.     btnExecuter.setVisibility(state.isBtnExecuterVisible() ? View.VISIBLE : View.INVISIBLE);
74.     // nb de réponses
75.     infoReponses.setText(String.format("Liste des réponses (%s)", nbReponses));
76. }
77.
78. @Override
79. protected void notifyEndOfUpdates() {
80.
81. }
82.
83. @Override
84. protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
85.     // le bouton [Exécuter] remplace le bouton [Annuler]
86.     btnAnnuler.setVisibility(View.INVISIBLE);
87.     btnExecuter.setVisibility(View.VISIBLE);
88. }
89. }
```

- lignes 7-18 : assurent la sauvegarde du fragment lorsque la classe parent demande de le faire ;
- ligne 11 : visibilité du message d'erreur sur le délai d'attente ;
- ligne 12 : visibilité du message d'erreur sur le nombre de nombres aléatoires demandés ;
- ligne 13 : visibilité du message d'erreur sur l'URL du service web / JSON ;
- ligne 14 : visibilité du message d'erreur sur l'intervalle [a,b] de génération des nombres aléatoires ;
- ligne 15 : visibilité du bouton [Exécuter] ;
- ligne 16 : la liste des réponses reçues ;
- lignes 20-23 : doivent rendre le n° de la vue. Le n° du fragment est ici 0 puisqu'il n'y en a qu'un ;
- lignes 25-38 : initialisation des champs du fragment, soit sur une 1ère visite (previousState==null), soit sur une visite ultérieure ;
 - lignes 29-30 : si ce n'est pas la 1ère visite, le champ [reponses] est restauré à partir de l'état précédent du fragment ;
 - lignes 31-33 : si c'est la 1ère visite, alors le champ [reponses] est initialisé avec une liste vide ;
 - lignes 34-37 : à partir du champ [reponses] on peut construire la source de données du *ListView* du fragment (ligne 35) ainsi que le nombre de réponses (ligne 37) ;
- lignes 40-55 : exécutées pour initialiser la vue associée au fragment, soit sur une 1ère visite (previousState==null), soit sur une visite ultérieure ;
 - ligne 43 : on associe le *ListView* du fragment à la source de données qui vient d'être construit dans la méthode [initFragment] ;
 - lignes 45-54 : si c'est la 1ère visite, on prépare la vue pour son 1er affichage ;
- lignes 57-60 : exécutées lors d'une navigation inter-fragments associée à une action de type [SUBMIT]. Ici, il n'y a qu'un fragment et donc pas de navigation inter-fragments ;
- lignes 63-76 : exécutées lors d'une navigation inter-fragments associée à une action de type [NAVIGATION] ou bien lors d'un cycle sauvegarde / restauration dûe à une rotation du périphérique ou une autre raison. Ici, seul ce dernier cas peut se produire. Il faut se rappeler qu'ici, dans tous les cas, [previousState] est toujours non *null* ;
- ligne 65 : on caste l'état précédent dans le type de l'état du fragment ;
- lignes 66-75 : on utilise le contenu de l'état précédent pour restaurer la vue ;
- lignes 78-81 : appelées lorsque toutes les mises à jour précédentes ont été effectuées. Ici, il n'y a rien à faire ;
- lignes 83-89 : exécutées lorsque toutes les tâches asynchrones sont terminées. Ici on cache le bouton [Annuler] pour le remplacer par le bouton [Exécuter] ;

2.8.3.8 Les tests

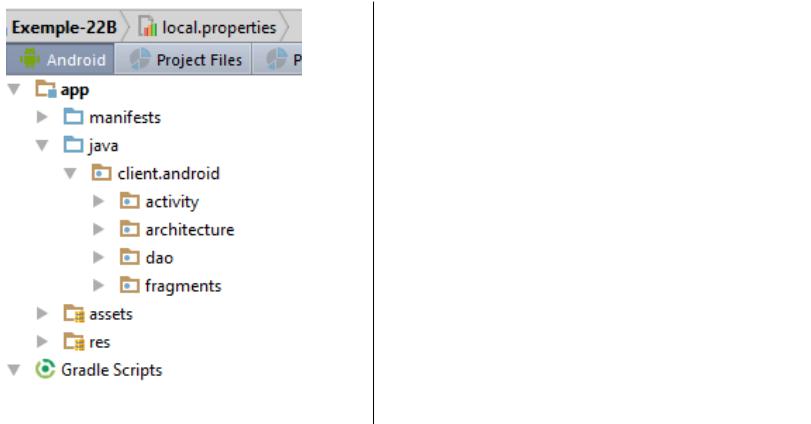
Le lecteur est invité à faire les tests suivants :

- créer des erreurs et faire tourner le périphérique : les messages d'erreur doivent être conservés ;
- obtenir des nombres aléatoires et faire tourner le périphérique : les nombres aléatoires obtenus doivent rester affichés ;
- mettre une attente de plusieurs secondes et faire tourner le périphérique pendant l'attente : les tâches doivent avoir été annulées (cela se voit dans les logs) ;

2.8.4 Exemple-22B

Nous reprenons ici l'exemple 22 pour le refactoriser selon le modèle du projet [client-android-skel]. On rappelle que le projet [Exemple-22] gère correctement le cycle sauvegarde / restauration des fragments lors d'une rotation et que c'est lui qui a servi de base au projet [client-android-skel].

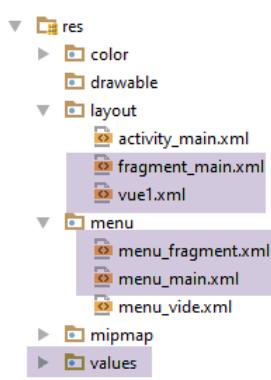
Nous dupliquons le projet [client-android-skel] dans [exemples/Exemple-22B] et nous chargeons ce dernier projet :



Puis nous copions divers éléments du projet [Exemple-22] dans le projet [Exemple-22B].

Tout d'abord, nous copions des éléments du dossier [res] :

- [layout/fragment_main.xml, layout/vue1.xml, menu/menu_fragment.xml, menu/menu_main.xml, le dossier [values] ;



On modifiera la marge haute des deux vues à 120 dp :

[vue1.xml] :

```

1. <TextView
2.     android:layout_width="wrap_content"
3.     android:layout_height="wrap_content"
4.     android:textAppearance="?android:attr/textAppearanceLarge"
5.     android:text="@string/titre_vue1"
6.     android:id="@+id/textViewTitreVue1"
7.     android:layout_marginTop="120dp"
8.     android:textSize="50sp"
9.     android:layout_gravity="center/left"
10.    android:layout_alignParentTop="true"
11.    android:layout_centerHorizontal="true"/>

```

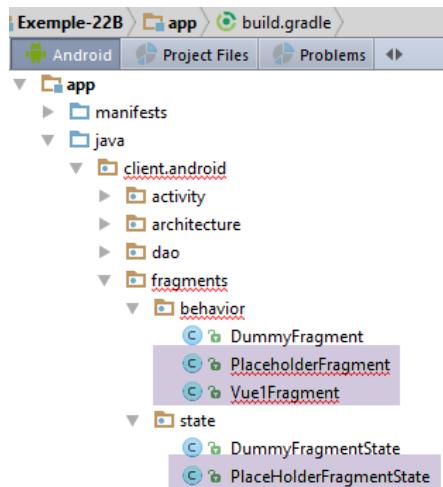
[fragment_main] :

```

1. <TextView
2.     android:id="@+id/section_label"
3.     android:layout_width="wrap_content"
4.     android:layout_height="wrap_content"
5.     android:layout_marginTop="120dp"/>

```

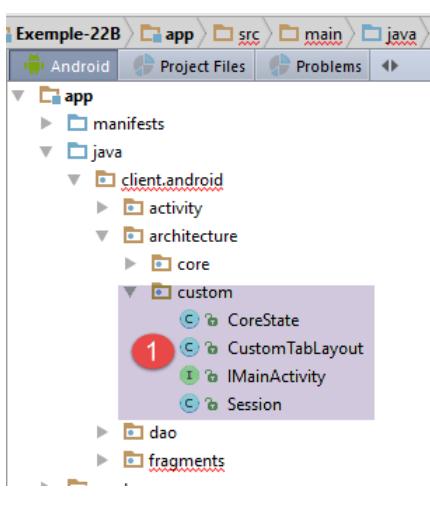
Ensuite nous copions les éléments [Vue1Fragment, PlaceHolderFragment, PlaceHolderFragmentState] :



A ce stade, nous pouvons tenter une 1ère compilation. Un premier type d'erreurs apparaît : celui des *imports* incorrects parce que des classes ont changé de package. On corrige ces *imports*. Un second type d'erreurs est dû au fait que les fragments n'implémentent pas toutes les méthodes de leur classe parent [AbstractFragment]. On corrige par (Alt+Entrée).

Les erreurs restantes proviennent des différences existantes entre l'ancienne et la nouvelle classe [AbstractFragment]. Pour l'instant, on les ignore.

2.8.4.1 Personnalisation du projet



Dans le dossier [custom] se trouvent les éléments d'architecture personnalisables par le développeur.

L'interface [IMainActivity] permet de spécifier certaines caractéristiques du projet :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.ISession;
4. import client.android.dao.service.IDao;
5.
6. public interface IMainActivity extends IDao {
7.
8.     // accès à la session
9.     ISession getSession();
10.
11.    // changement de vue
12.    void navigateToView(int position, ISession.Action action);
13.
14.    // gestion de l'attente
15.    void beginWaiting();

```

```

16.
17.     void cancelWaiting();
18.
19.     // mode debug
20.     boolean IS_DEBUG_ENABLED = true;
21.
22.     // délai maximal d'attente de la réponse du serveur
23.     int TIMEOUT = 1000;
24.
25.     // délai d'attente avant exécution de la requête client
26.     int DELAY = 0;
27.
28.     // authentification basique
29.     boolean IS_BASIC_AUTHENTICATION_NEEDED = false;
30.
31.     // adjacence des fragments
32.     int OFF_SCREEN_PAGE_LIMIT = 1;
33.
34.     // barre d'onglets
35.     boolean ARE_TABS_NEEDED = true;
36.
37.     // image d'attente
38.     boolean IS_WAITING_ICON_NEEDED = false;
39.
40.     // nombre de fragments
41.     int FRAGMENTS_COUNT = 5;
42.
43. }

```

- lignes 23, 26, 29, 38 : caractéristiques de la couche [DAO]. Il n'y en a pas ici ;
- ligne 41 : il y a ici cinq fragments ;
- ligne 32 : adjacence des fragments. Cette constante peut avoir ici une valeur dans [1,4]. Le lecteur est encouragé à faire varier cette valeur pour voir si l'application continue à fonctionner ;
- ligne 35 : c'est une application à onglets ;

La classe [CoreState] qui mémorise l'état des fragments sera la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.MenuInstanceState;
4. import client.android.fragments.state.PlaceHolderFragmentState;
5. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
6. import com.fasterxml.jackson.annotation.JsonSubTypes;
7. import com.fasterxml.jackson.annotation.JsonProperty;
8.
9. @JsonIgnoreProperties(ignoreUnknown = true)
10. @JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
11. @JsonSubTypes({
12.     @JsonSubTypes.Type(value = PlaceHolderFragmentState.class)})
13. )
14. public class CoreState {
15.     // fragment visité ou non
16.     protected boolean hasBeenVisited = false;
17.     // état de l'éventuel menu du fragment
18.     protected MenuItemState[] menuOptionsState;
19.
20.     // getters et setters
21.     ...
22. }

```

- ligne 12 : nous déclarons la classe de l'état du fragment [PlaceHolderFragment]. Le fragment [Vue1Fragment] n'a lui pas d'état ;

La classe [Session] est la suivante :

```

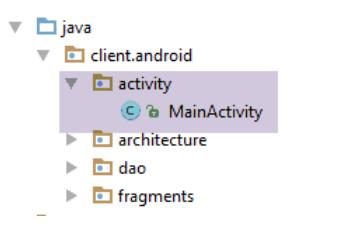
1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.AbstractSession;
4.
5. public class Session extends AbstractSession {
6.     // données à partager entre fragments eux-mêmes et entre fragments et activité
7.     // les éléments qui ne peuvent être serialisés en JSON doivent avoir l'annotation @JsonIgnore
8.     // ne pas oublier les getters et setters nécessaires pour la serialisation / déserialisation JSON
9.
10.    // nombre de fragments visités
11.    private int numVisit;
12.    // n° fragment de type [PlaceholderFragment] affiché dans second onglet
13.    private int numFragment = -1;
14.
15.    // getters et setters
16.    ...

```

17. }

C'est la session du projet [Exemple-22].

2.8.4.2 L'activité [MainActivity]



L'activité [MainActivity] est la suivante :

```
1. package client.android.activity;
2.
3. import android.os.Bundle;
4. import android.support.design.widget.TabLayout;
5. import android.util.Log;
6. import android.view.MenuItem;
7. import client.android.R;
8. import client.android.architecture.core.AbstractActivity;
9. import client.android.architecture.core.AbstractFragment;
10. import client.android.architecture.core.ISession;
11. import client.android.architecture.custom.IMainActivity;
12. import client.android.architecture.custom.Session;
13. import client.android.dao.service.Dao;
14. import client.android.dao.service.IDao;
15. import client.android.fragments.behavior.PlaceholderFragment_;
16. import client.android.fragments.behavior.Vue1Fragment_;
17. import org.androidannotations.annotations.Bean;
18. import org.androidannotations.annotations.EActivity;
19. import org.androidannotations.annotations.OptionsMenu;
20.
21. @EActivity
22. @OptionsMenu(R.menu.menu_main)
23. public class MainActivity extends AbstractActivity {
24.
25.     // couche [DAO]
26.     @Bean(Dao.class)
27.     protected IDao dao;
28.     // session
29.     private Session session;
30.
31.     // gestion du menu-----
32.     @Override
33.     public boolean onOptionsItemSelected(MenuItem item) {
34.         ...
35.     }
36.
37.     private void showFragment(int i) {
38.         ...
39.     }
40.
41.     // implémentation méthodes de la classe parent -----
42.     ...
43. }
```

Ici, la classe [MainActivity] est plus conséquente que celle des exemples précédents pour deux raisons :

- il y a des onglets à gérer ;
- il y a un menu à gérer ;

2.8.4.2.1 Implémentation des méthodes de la classe parent

```
1. // méthodes classe parent -----
2. @Override
3. protected void onCreateActivity() {
4.     // log
5.     if (IS_DEBUG_ENABLED) {
6.         Log.d(class.getName(), "onCreateActivity");
7.     }
8.     // on continue les initialisations commencées par la classe parent
9.     // session
```

```

10.     this.session = (Session) super.session;
11.     ...
12. }
13.
14. @Override
15. protected IDao getDao() {
16.     return dao;
17. }
18.
19. @Override
20. protected AbstractFragment[] getFragments() {
21.     // n° de fragment
22.     final String ARG_SECTION_NUMBER = "section_number";
23.     // initialisation du tableau des fragments
24.     AbstractFragment[] fragments = new AbstractFragment[FRAGMENTS_COUNT];
25.     int i;
26.     for (i = 0; i < fragments.length - 1; i++) {
27.         // on crée un fragment
28.         fragments[i] = new PlaceholderFragment_();
29.         // on peut passer des arguments au fragment
30.         Bundle args = new Bundle();
31.         args.putInt(ARG_SECTION_NUMBER, i + 1);
32.         fragments[i].setArguments(args);
33.     }
34.     // un fragment de +
35.     fragments[i] = new Vue1Fragment_();
36.     // résultat
37.     return fragments;
38. }
39.
40.
41. @Override
42. protected CharSequence getFragmentTitle(int position) {
43.     // pas de titres ici
44.     return null;
45. }
46.
47. @Override
48. protected void navigateOnTabSelected(int position) {
49. ...
50. }
51.
52. @Override
53. protected int getFirstView() {
54.     return IMainActivity.FRAGMENTS_COUNT - 1;
55. }

```

- lignes 2-12 : la méthode [onCreateActivity] est appelée par la classe parent [AbstractActivity] lorsque l'activité est créée la 1ère fois ou recréée lors d'un cycle sauvegarde / restauration. Lorsque cette méthode est appelée, la classe parent a déjà restauré la session ;
- ligne 10 : on récupère une référence locale de la session. Le changement de type est dû au fait que la session de la classe parent est de type [AbstractSession] ;
- lignes 19-38 : la méthode [getFragments] doit rendre à la classe parent le tableau des fragments gérés par l'application. Ici il y en a [FRAGMENTS_COUNT], nombre défini dans [IMainActivity]. Les [FRAGMENTS_COUNT-1] premiers fragments sont de type [PlaceHolderFragment] et le dernier de type [Vue1Fragment] ;
- lignes 41-45 : la méthode [getFragmentTitle] doit rendre les titres des fragments lorsque cette information peut être utile. Ce n'est pas le cas ici ;
- lignes 47-50 : cette méthode est appelée par la classe parent lorsque l'utilisateur clique sur un onglet. Nous allons y revenir dans le paragraphe suivant ;
- lignes 52-55 : rend le n° de la 1ère vue à afficher lorsque l'application démarre. Ici c'est le fragment [Vue1Fragment] qui doit être affiché en premier. En écrivant ces lignes, je me dis que la méthode [getFirstView] pourrait être avantageusement remplacée par une constante dans [IMainActivity] ;

2.8.4.2.2 Gestion des onglets

Les onglets sont gérés par les méthodes suivantes :

```

1. @Override
2. protected void onCreateActivity() {
3.     // log
4.     if (IS_DEBUG_ENABLED) {
5.         Log.d(className, "onCreateActivity");
6.     }
7.     // on continue les initialisations commencées par la classe parent
8.     // session
9.     this.session = (Session) super.session;
10.    // 1er onglet
11.    TabLayout.Tab tab = tabLayout.newTab();
12.    tab.setText("Vue 1");
13.    tabLayout.addTab(tab);

```

```

14.     // 2ième onglet ?
15.     int numFragment = session.getNumFragment();
16.     if (numFragment != -1) {
17.         TabLayout.Tab tab2 = tabLayout.newTab();
18.         tab2.setText(String.format("Fragment n° %s", (numFragment + 1)));
19.         tabLayout.addTab(tab2);
20.     }
21. }
22.
23. @Override
24. protected void navigateOnTabSelected(int position) {
25.     // n° du fragment à afficher
26.     int numFragment;
27.     switch (position) {
28.         case 0:
29.             // n° fragment [Vue1Fragment]
30.             numFragment = getView();
31.             break;
32.         default:
33.             // n° fragment [PlaceholderFragment]
34.             numFragment = session.getNumFragment();
35.     }
36.     // affichage fragment
37.     if (numFragment != mViewPager.getCurrentItem()) {
38.         navigateToView(numFragment, ISession.Action.SUBMIT);
39.     }
40. }
41. }
```

- lignes 1-20 : la méthode [onCreateActivity] est appelée par la classe parent [AbstractActivity] lorsque l'activité est créée la 1ère fois ou recréée lors d'un cycle sauvegarde / restauration. Lorsque cette méthode est appelée, la classe parent a déjà restauré la session ;
- ligne 9 : on récupère une référence locale de la session. Le changement de type est dû au fait que la session de la classe parent est de type [AbstractSession] ;
- lignes 11-13 : on crée le 1er onglet ;
- lignes 15-20 : on crée le second onglet si un n° de fragment est enregistré dans la session (ligne 15). Ce n° vaut initialement -1 lors de la 1ère construction de l'activité ;
- lignes 23-39 : cette méthode est appelée par la classe parent lorsque l'utilisateur clique sur un onglet ;
- lignes 28-31 : si c'est l'onglet 0 qui est cliqué, alors on doit faire afficher [Vue1Fragment]. On sait que c'est la 1ère vue qui a été affichée au démarrage de l'application ;
- lignes 32-35 : si c'est l'onglet 1 qui est cliqué, alors on doit faire afficher le fragment dont le n° est enregistré dans la session ;
- lignes 37-39 : on navigue vers le fragment choisi. L'action associée est [SUBMIT]. Aurait-ce pu être [NAVIGATION] ? Dans ce document, on utilise [NAVIGATION] uniquement lorsque l'affichage du nouveau fragment ne nécessite de connaître que son état précédent. Ici, ce n'est pas le cas puisque l'affichage du fragment affiché doit changer par rapport à son état précédent pour afficher une visite de plus ;

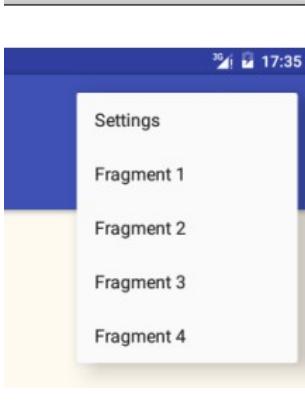
2.8.4.2.3 Gestion du menu

L'activité est associée au menu [menu_main.xml] suivant :

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.       xmlns:app="http://schemas.android.com/apk/res-auto"
3.       xmlns:tools="http://schemas.android.com/tools"
4.       tools:context="exemples.android.MainActivity">
5.     <item android:id="@+id/action_settings"
6.           android:title="@string/action_settings"
7.           android:orderInCategory="100"
8.           app:showAsAction="never"/>
9.     <item android:id="@+id/fragment1"
10.        android:title="@string/fragment1"
11.        android:orderInCategory="100"
12.        app:showAsAction="never"/>
13.     <item android:id="@+id/fragment2"
14.        android:title="@string/fragment2"
15.        android:orderInCategory="100"
16.        app:showAsAction="never"/>
17.     <item android:id="@+id/fragment3"
18.        android:title="@string/fragment3"
19.        android:orderInCategory="100"
20.        app:showAsAction="never"/>
21.     <item android:id="@+id/fragment4"
22.        android:title="@string/fragment4"
23.        android:orderInCategory="100"
24.        app:showAsAction="never"/>
25.   </menu>
```

qui affiche la chose suivante :



La gestion du menu est assuré par les méthodes suivantes :

```

1.  @Override
2.  public boolean onOptionsItemSelected(MenuItem item) {
3.      // log
4.      if (IS_DEBUG_ENABLED) {
5.          Log.d(className, "onOptionsItemSelected");
6.      }
7.      // traitement des options de menu
8.      int id = item.getItemId();
9.      switch (id) {
10.         case R.id.action_settings: {
11.             if (IS_DEBUG_ENABLED) {
12.                 Log.d(className, "action_settings selected");
13.             }
14.             break;
15.         }
16.         case R.id.fragment1: {
17.             showFragment(0);
18.             break;
19.         }
20.         case R.id.fragment2: {
21.             showFragment(1);
22.             break;
23.         }
24.         case R.id.fragment3: {
25.             showFragment(2);
26.             break;
27.         }
28.         case R.id.fragment4: {
29.             showFragment(3);
30.             break;
31.         }
32.     }
33.     // item traité
34.     return true;
35. }
36.
37. private void showFragment(int i) {
38.     if (i < FRAGMENTS_COUNT && mViewPager.getCurrentItem() != i) {
39.         // pas de navigation sur sélection logicielle d'un onglet
40.         session.setNavigationOnTabSelectionNeeded(false);
41.         // on recrée les deux onglets pour une histoire de police de caractères des titres
42.         tabLayout.removeAllTabs();
43.         tabLayout.addTab(tabLayout.newTab().setText("Vue1"), false);
44.         tabLayout.addTab(tabLayout.newTab().setText(String.format("Fragment n° %s", (i + 1))), false);
45.         // le n° du fragment à afficher est mis en session
46.         session.setNumFragment(i);
47.         // on sélectionne l'onglet n° 2 avec navigation
48.         session.setNavigationOnTabSelectionNeeded(true);
49.         tabLayout.getTabAt(1).select();
50.     }
51. }

```

- lignes 16-31 : gestion du clic sur une option de menu de type [Fragment*i*] ;
- lignes 37-50 : affichent le fragment n° *i* (il s'agit des fragments de type PlaceHolderFragment) dans l'onglet n° 1 (2ième onglet) ;
- lignes 42-44 : on décide de supprimer les onglets existants pour en recréer deux nouveaux. Cette décision a été prise pour contourner le problème suivant : lorsqu'on se contente d'afficher le fragment dans l'onglet 1 existant (sans donc le supprimer), curieusement son titre a une allure (police de caractères, taille) différente de celle du titre de l'onglet 0 ;

- lignes 43-44 : les deux onglets sont créés mais pas sélectionnés (dernier paramètre à *false*) ;
- ligne 40 : les opérations des lignes 42-44 sont susceptibles de faire des opérations [select] sur les onglets ce qui va appeler le gestionnaire [onTabSelected]. Si on ne fait rien, il y aura alors navigation vers un fragment. On évite cela en mettant le booléen [navigationOnTabSelectionNeeded] à *faux* dans la session. Ce booléen est automatiquement remis à *vrai* par la classe [AbstractFragment] lorsqu'un fragment devient visible ;
- ligne 46 : on note le n° du fragment à afficher dans la session ;
- lignes 48-50 : on sélectionne l'onglet n° 2 avec navigation (ligne 48). Cela va déclencher la procédure [onTabSelected] qui va :
 - faire afficher le fragment dont le n° a été mis en session ;
 - mémoriser en session le n° de l'onglet sélectionné ;

2.8.4.3 Le fragment [Vue1Fragment]

Nous donnons ici la version finale du fragment :

```

1. package client.android.fragments.behavior;
2.
3. import android.widget.EditText;
4. import android.widget.Toast;
5. import client.android.R;
6. import client.android.architecture.core.AbstractFragment;
7. import client.android.architecture.custom.CoreState;
8. import client.android.architecture.custom.IMainActivity;
9. import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.OptionsMenu;
12. import org.androidannotations.annotations.ViewById;
13.
14. @EFragment(R.layout.vue1)
15. @OptionsMenu(R.menu.menu_fragment)
16. public class Vue1Fragment extends AbstractFragment {
17.
18.     // les éléments de l'interface visuelle
19.     @ViewById(R.id.editTextNom)
20.     protected EditText editTextNom;
21.
22.     // gestionnaire d'évt
23.     @Click(R.id.buttonValider)
24.     protected void doValider() {
25.         // on affiche le nom saisi
26.         Toast.makeText(activity, String.format("Bonjour %s", editTextNom.getText().toString()), Toast.LENGTH_LONG).show();
27.     }
28.
29.     // cycle de vie du fragment -----
30.     private void initFragment() {
31.         // rien à faire
32.     }
33.
34.     // sauvegarde état fragment
35.     @Override
36.     public CoreState saveFragment() {
37.         // état de la vue - rien à sauvegarder
38.         return new CoreState();
39.     }
40.
41.     @Override
42.     protected int getNumView() {
43.         return IMainActivity.FRAGMENTS_COUNT - 1;
44.     }
45.
46.     @Override
47.     protected void initFragment(CoreState previousState) {
48.         // rien à faire
49.     }
50.
51.     @Override
52.     protected void initView(CoreState previousState) {
53.         // 1ère visite ?
54.         if (previousState == null) {
55.             // on affiche le n° de la visite
56.             showNumVisit();
57.         }
58.     }
59. }
60.
61. @Override
62. protected void updateOnSubmit(CoreState previousState) {
63.     // on affiche le n° de la visite
64.     showNumVisit();
65. }
```

```

67.
68.     @Override
69.     protected void updateOnRestore(CoreState previousState) {
70.
71. }
72.
73.     @Override
74.     protected void notifyEndOfUpdates() {
75.
76. }
77.
78.     @Override
79.     protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
80.
81. }
82.
83. // méthodes privées -----
84. // affichage n° de visite
85. private void showNumVisit() {
86.     // incrément n° de visite
87.     int numVisit = session.getNumVisit();
88.     numVisit++;
89.     session.setNumVisit(numVisit);
90.     // on affiche le n° de la visite
91.     Toast.makeText(activity, String.format("Visite n° %s", numVisit), Toast.LENGTH_SHORT).show();
92. }
93. }
```

La classe est quasi vide.

- lignes 35-39 : appelées par la classe parent lorsque le fragment doit sauver son état. Le fragment [Vue1Fragment] n'a pas d'état à sauver. On rend simplement une instance de la classe de base [CoreState] (rappel : on ne peut pas rendre `null`) ;
- lignes 41-44 : doivent rendre le n° du fragment. Le fragment [Vue1Fragment] a par construction le n° [FRAGMENTS_COUNT-1] ;
- lignes 51-59 : appelées par la classe parent lorsque le fragment est construit pour la 1ère fois (`previousState==null`) ou les fois suivantes (`previousState!=null`) ;
 - lignes 54-57 : si c'est la 1ère visite, on incrémente le n° de visite et on l'affiche (lignes 85-92) ;
- lignes 61-65 : appelées lorsque le fragment va être affiché associé à une action [SUBMIT]. On incrémente le n° de visite et on l'affiche. Ici, il n'est pas possible que le n° de visite soit incrémenté deux fois dans le cycle de vie. En effet, la 1ère visite au fragment [Vue1Fragment] est faite au démarrage de l'application lorsque l'action vaut [NONE] par construction dans la session. Ceci assure que la méthode [updateOnSubmit] ne va pas être appelée. Ensuite, ce ne sera plus jamais la 1ère visite et la méthode [initView] ne fera rien ;
- lignes 68-71 : appelées dans un cycle sauvegarde / restauration. Le fragment n'ayant pas d'état, il n'y a ici rien à restaurer ;
- lignes 73-76 : appelées lorsque toutes les mises à jour précédentes ont été effectuées. Ici, il n'y a rien de plus à faire ;
- lignes 78-81 : appelées lorsque les tâches asynchrones lancées sont toutes terminées. Ici, il n'y a pas de tâches asynchrones ;

2.8.4.4 L'état [PlaceHolderFragmentState]

L'état du fragment [PlaceHolderFragment] sera le suivant :

```

1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. public class PlaceHolderFragmentState extends CoreState {
6.     // texte
7.     private String text;
8.
9.     // constructeurs
10.    public PlaceHolderFragmentState() {
11.
12.    }
13.
14.    public PlaceHolderFragmentState(String text) {
15.        super();
16.        this.text = text;
17.    }
18.
19.    // getters et setters
20.    ...
21. }
```

- lorsqu'il faudra sauver l'état du fragment, on sauvera le texte qu'il affichait (ligne 7) ;

2.8.4.5 Le fragment [PlaceHolderFragment]

Le fragment [PlaceHolderFragment] sera le suivant :

```
1. package client.android.fragments.behavior;
2.
3. import android.util.Log;
4. import android.widget.TextView;
5. import client.android.R;
6. import client.android.architecture.core.AbstractFragment;
7. import client.android.architecture.custom.CoreState;
8. import client.android.fragments.state.PlaceHolderFragmentState;
9. import org.androidannotations.annotations.EFragment;
10. import org.androidannotations.annotations.OptionsMenu;
11. import org.androidannotations.annotations.ViewById;
12.
13. @EFragment(R.layout.fragment_main)
14. @OptionsMenu(R.menu.menu_fragment)
15. public class PlaceholderFragment extends AbstractFragment {
16.
17.     // composants de l'interface visuelle
18.     @ViewById(R.id.section_label)
19.     protected TextView textViewInfo;
20.     @ViewById(R.id.textView1)
21.     protected TextView textView1;
22.
23.     // data
24.     private String text;
25.
26.     // n° de fragment
27.     private static final String ARG_SECTION_NUMBER = "section_number";
28.
29.     // implémentation méthodes de la classe parent -----
30.     @Override
31.     public CoreState saveFragment() {
32.         // on sauvegarde l'état du fragment
33.         PlaceHolderFragmentState placeHolderFragmentState = new PlaceHolderFragmentState();
34.         placeHolderFragmentState.setText(textViewInfo.getText().toString());
35.         return placeHolderFragmentState;
36.     }
37.
38.     @Override
39.     protected int getNumView() {
40.         return getArguments().getInt(ARG_SECTION_NUMBER) - 1;
41.     }
42.
43.     @Override
44.     protected void initFragment(CoreState previousState) {
45.         // texte original
46.         text = getString(R.string.section_format, getArguments().getInt(ARG_SECTION_NUMBER));
47.     }
48.
49.     @Override
50.     protected void initView(CoreState previousState) {
51.     }
52.
53.     @Override
54.     protected void updateOnSubmit(CoreState previousState) {
55.         // on met à jour le texte affiché
56.         // incrément n° de visite
57.         int numVisit = session.getNumVisit();
58.         numVisit++;
59.         session.setNumVisit(numVisit);
60.         // texte modifié
61.         textViewInfo.setText(String.format("%s, visite %s", text, numVisit));
62.         // log
63.         if (isDebugEnabled) {
64.             Log.d(className, String.format("updateForSubmit, numvisit=%s, texte affiché=%s, visibility=%s", numVisit,
65.                 textViewInfo.getText().toString(), textViewInfo.getVisibility()));
66.         }
67.
68.     @Override
69.     protected void updateOnRestore(CoreState previousState) {
70.         // on restaure le texte affiché
71.         PlaceHolderFragmentState state = (PlaceHolderFragmentState) previousState;
72.         textViewInfo.setText(state.getText());
73.
74.     }
75.
76.     @Override
77.     protected void notifyEndOfUpdates() {
78.
79.     }
80.
81.     @Override
```

```

82.     protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
83.
84. }
85.
86. }
```

- lignes 30-36 : lorsque la classe parent demande au fragment de sauver son état, on sauve le texte affiché par le fragment (ligne 34) ;
- lignes 38-41 : rendent le n° du fragment. Celui-ci dépend du n° de section qu'on lui a passé en argument lors de sa création ;
- lignes 43-47 : appelées lors de la 1ère construction du fragment (previousState==null) ou lors des suivantes (previousState!=null) ;
 - ligne 46 : ici, on n'exploite pas l'état précédent. Le texte initial [text] (ligne 24) affiché lors de la 1ère visite est recalculé à chaque fois. C'est discutable. On aurait pu choisir de mettre également cette information dans l'état du fragment ;
- lignes 49-51 : appelées lors de la 1ère construction de la vue associée au fragment (previousState==null) ou lors des suivantes (previousState!=null). Il n'y a rien à faire ;
- lignes 53-56 : appelées lorsque le fragment va être affiché associé à une action [SUBMIT]. C'est toujours le cas sauf pour le cycle sauvegarde / restauration où l'action est [RESTORE]. On incrémente donc le n° de la visite et on l'affiche ;
- lignes 68-74 : appelées dans un cycle sauvegarde / restauration. On restaure le texte qui avait été sauvegardé dans l'état du fragment ;
- lignes 76-79 : appelées lorsque toutes les mises à jour précédentes ont été effectuées. Ici, il n'y a rien de plus à faire ;
- lignes 82-83 : appelées lorsque les tâches asynchrones lancées sont toutes terminées. Ici, il n'y a pas de tâches asynchrones ;

2.8.4.6 Tests

Le lecteur est invité à tester l'application en faisant tourner le périphérique pour vérifier que le fragment affiché ne perd pas son état. On regardera également les logs.

2.9 Conclusion

A l'issue de ce chapitre, nous disposons d'un projet modèle [client-android-skel] de client Android communiquant avec un service web / JSON avec les caractéristiques suivantes :

- la communication asynchrone avec le serveur web / JSON se fait avec la bibliothèque RxJava ;
- le cycle de vie d'un fragment (update, save, restore) est géré par sa classe parent [AbstractFragment] qui appelle à des moments précis certaines méthodes de ses classes filles. Le fragment fille n'a ainsi pas à se soucier des étapes du cycle de vie mais seulement d'implémenter certaines méthodes imposées par sa classe parent ;
- le cycle de vie de l'activité (save / restore) est géré par une classe abstraite [AbstractActivity] qui elle aussi impose à l'activité fille d'implémenter certaines méthodes ;
- la classe [AbstractActivity] est capable de gérer une application avec ou sans onglets, avec ou sans image d'attente, avec ou sans authentification basique auprès du serveur web / JSON. La présence ou non de ces éléments se fait par configuration ;

Nous allons maintenant présenter une étude de cas plus complexe que les exemples qui ont précédé. La nouvelle application s'appuiera sur le projet modèle [client-android-skel].

3 Étude de cas - Gestion de rendez-vous

3.1 Le projet

Dans le document [[Tutoriel AngularJS / Spring 4](#)], a été développée une application client / serveur pour gérer des rendez-vous de médecins. Par la suite nous référencerons ce document [[rdvmedecins-angular](#)]. L'application avait deux types de client :

- un client HTML / CSS / JS ;
- un client Android ;

Le client Android était obtenu de façon automatique à partir de la version HTML du client avec l'outil [[Cordova](#)]. Le projet sera ici de recréer ce client Android à la main en utilisant les connaissances acquises dans les chapitres précédents.

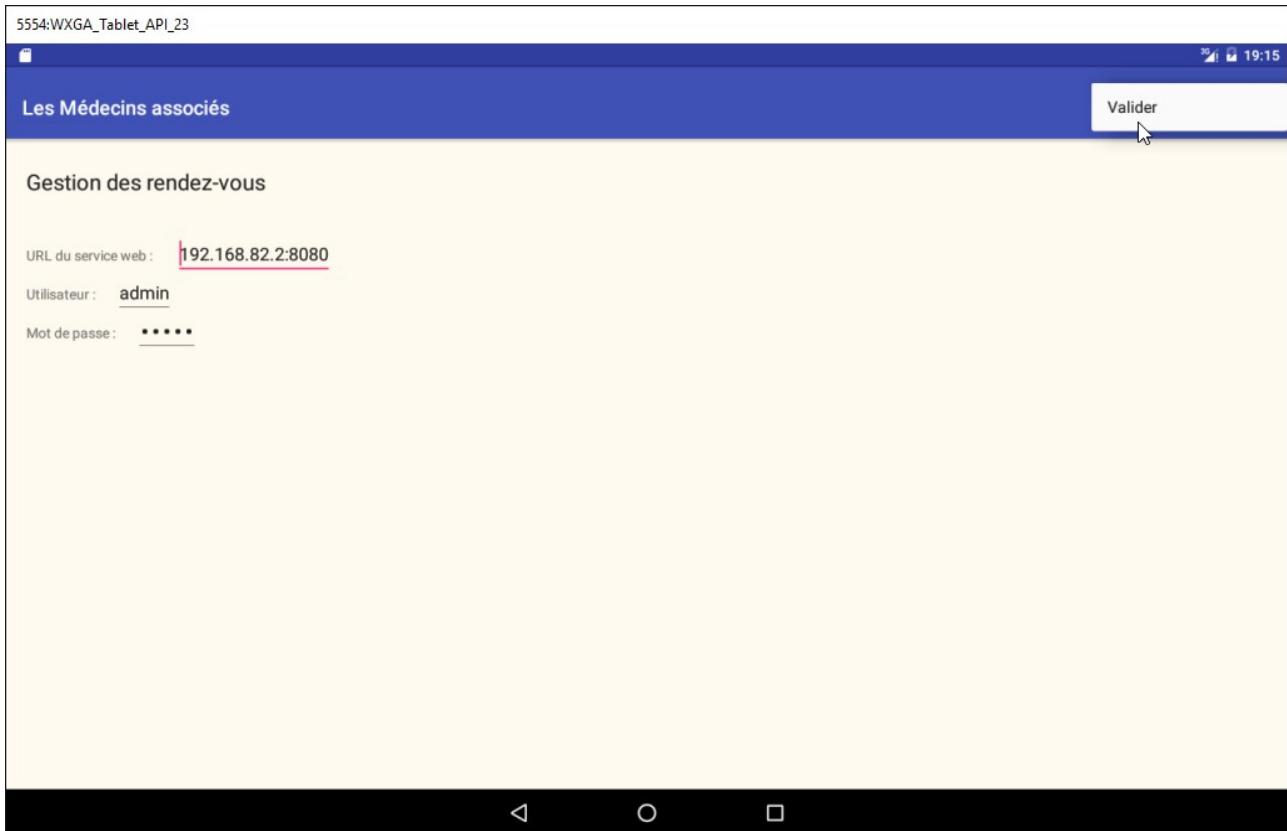
On notera une différence importante entre les deux solutions :

- celle que nous allons créer ne sera utilisable que sur les tablettes Android ;
- dans la version [[rdvmedecins-angular](#)], le client web mobile (HTML / CSS / JS) est utilisable sur n'importe quelle plate-forme (Android, IoS, Windows) ;

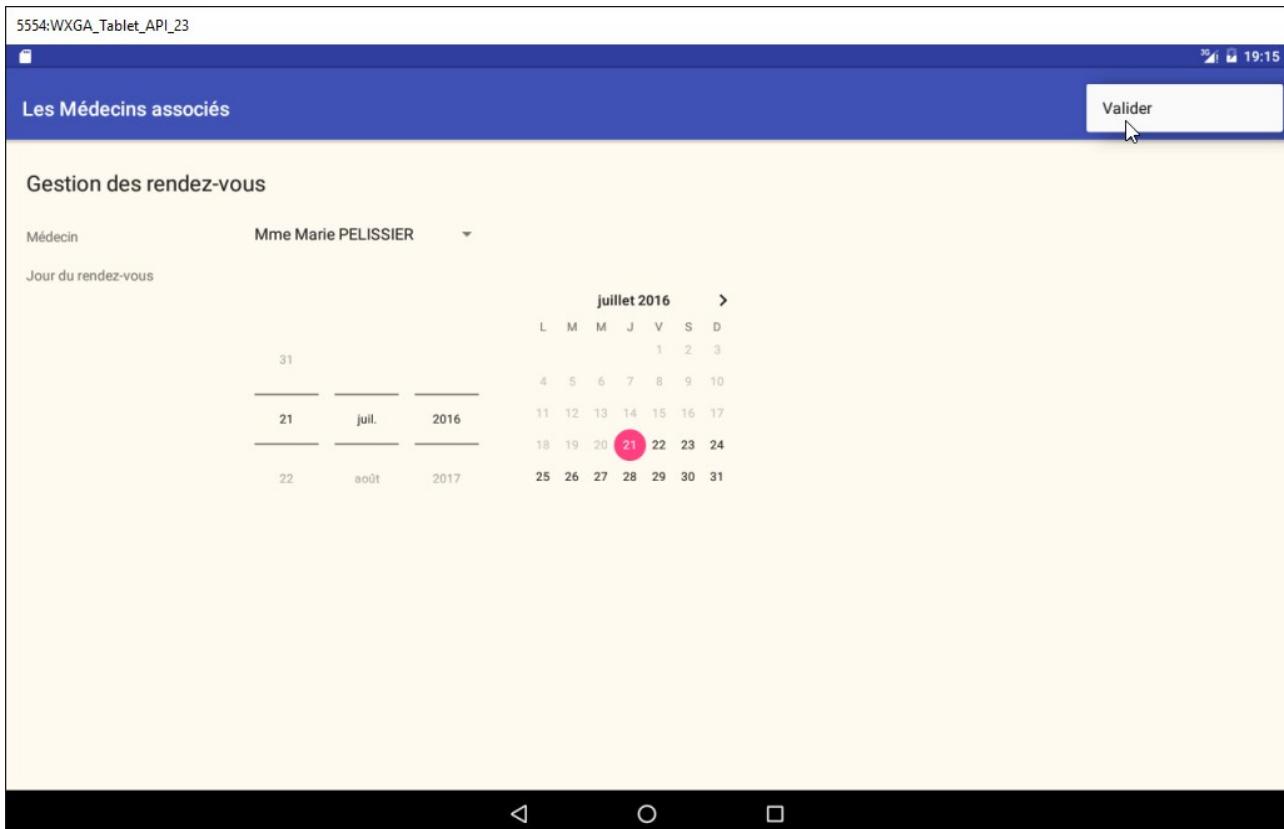
3.2 Les vues du client Android

Il y a quatre vues.

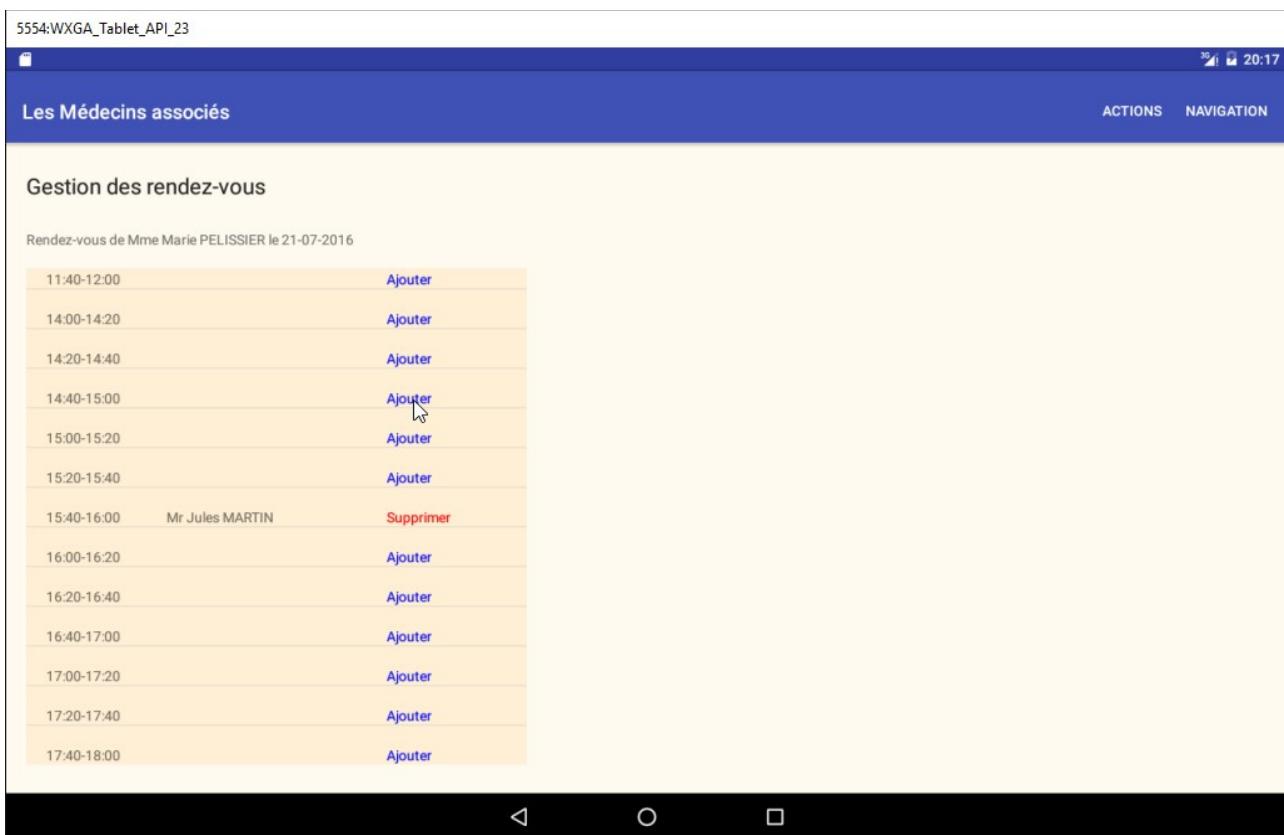
Vue de configuration



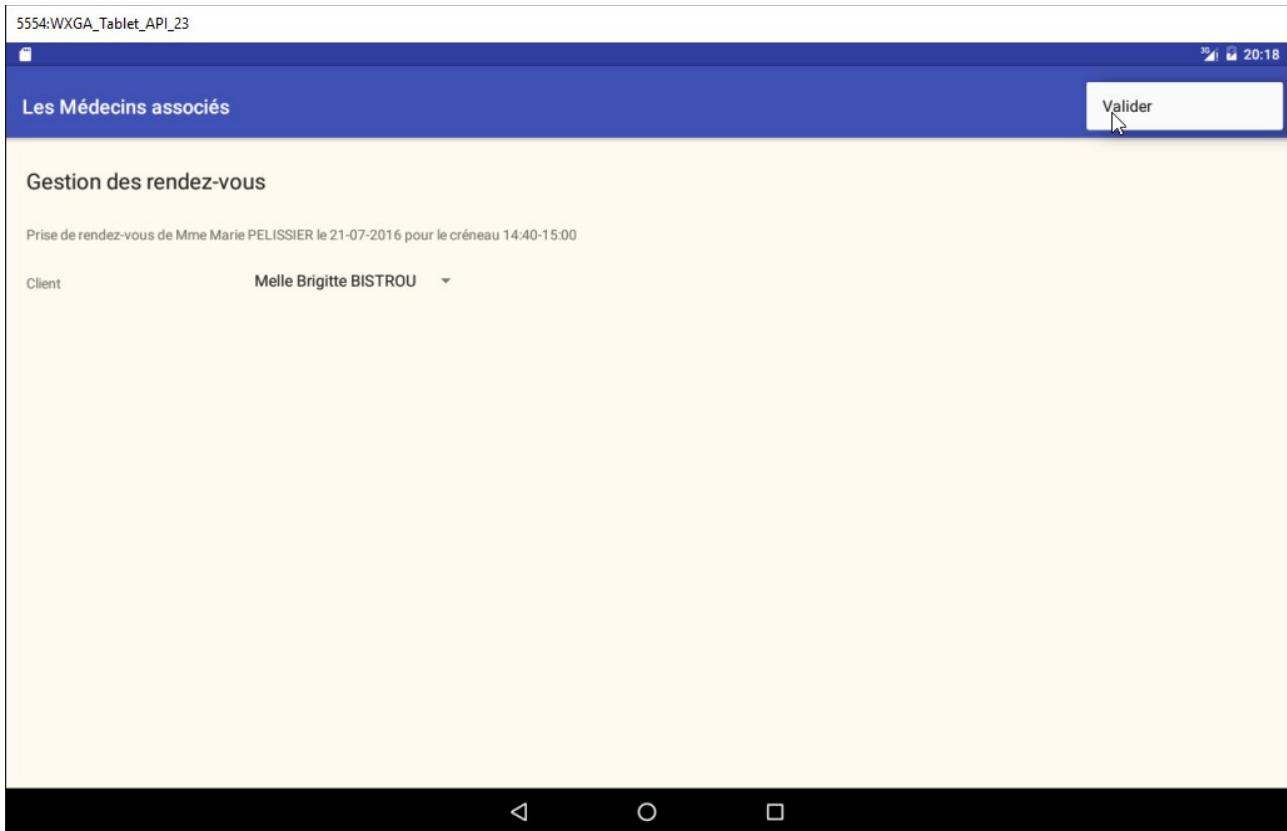
Vue du choix du médecin et de la date du rendez-vous



Vue du choix du créneau horaire du rendez-vous

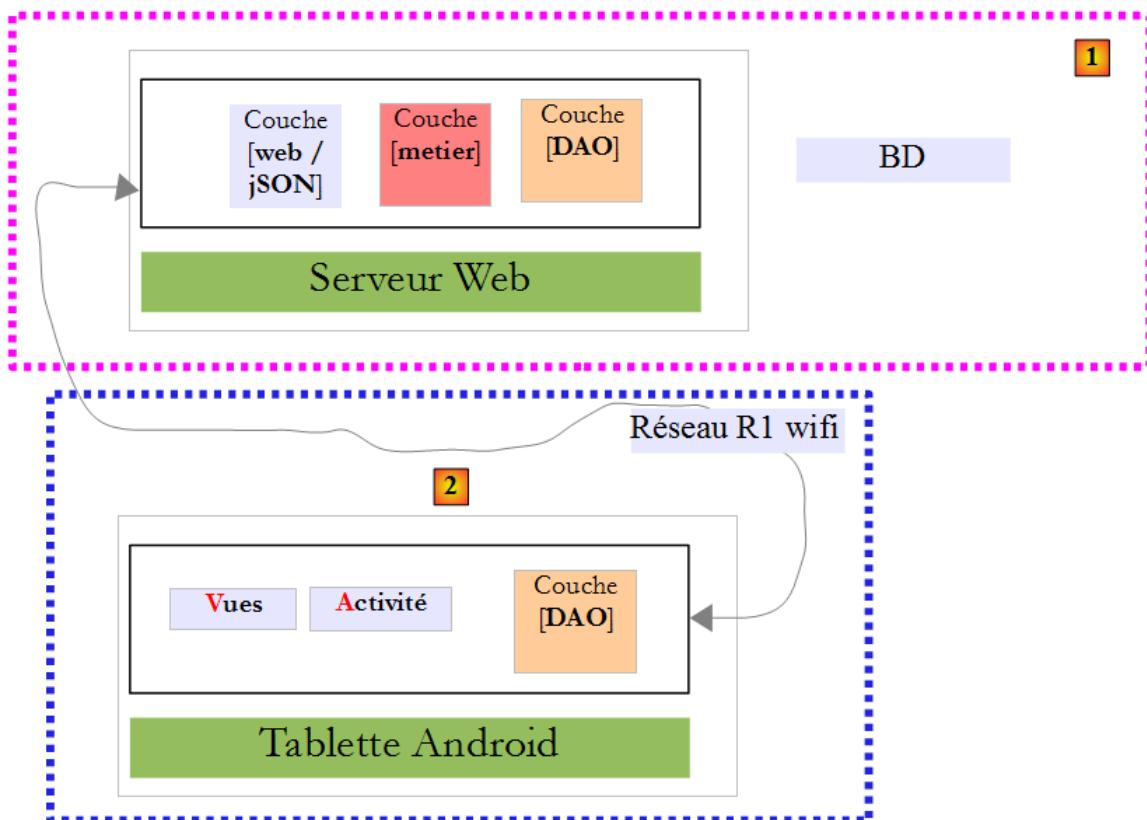


Vue du choix du client du rendez-vous



3.3 L'architecture du projet

On aura une architecture client / serveur analogue à celle de l'exemple [Exemple-15] (cf paragraphe 1.16, page 157) de ce document :



Les échanges asynchrones entre le client et le serveur seront gérés avec la bibliothèque RxAndroid.

3.4 La base de données

Elle ne joue pas un rôle fondamental dans ce document. Nous la donnons à titre d'information. On l'appellera [dbrdvmedecins]. C'est une base de données MySQL5 avec quatre tables :

The screenshot shows the MySQL Workbench interface with the database "dbrdvmedecins" selected. Under the "Tables (4)" section, four tables are listed: clients, creneaux, medecins, and rv. Each table has its own icon and a small preview of its structure.

3.4.1 La table [MEDECINS]

Elle contient des informations sur les médecins gérés par l'application [RdvMedecins].

Fields				
Field Name	Field Type	Size	Precision	Not Null
ID	BIGINT	20	0	✓
VERSION	INTEGER	11	0	✓
TITRE	VARCHAR	5	0	✓
NOM	VARCHAR	30	0	✓
PRENOM	VARCHAR	30	0	✓

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mme	PELISSIER	Marie
2	1	Mr	BROMARD	Jacques
3	1	Mr	JANDOT	Philippe
4	1	Melle	JACQUEMOT	Justine

- ID : n° identifiant le médecin - clé primaire de la table

- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du médecin
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

3.4.2 La table [CLIENTS]

Les clients des différents médecins sont enregistrés dans la table [CLIENTS] :

The screenshot shows the MySQL Workbench interface with two panes. The left pane displays the table structure with columns: ID, VERSION, TITRE, NOM, and PRENOM. The right pane shows the data with four rows:

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mr	MARTIN	Jules
2	1	Mme	GERMAN	Christine
3	1	Mr	JACQUARD	Jules
4	1	Melle	BISTRONU	Brigitte

- ID : n° identifiant le client - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du client
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

3.4.3 La table [CRENEAUX]

Elle liste les créneaux horaires où les RV sont possibles :

The screenshot shows the MySQL Workbench interface with two panes. The left pane displays the table structure with columns: ID, VERSION, HDEBUT, MDEBUT, HFIN, MFIN, and ID_MEDECIN. The right pane shows the data with 31 rows:

ID	VERSION	HDEBUT	MDEBUT	HFIN	MFIN	ID_MEDECIN
1	1	8	0	8	40	
2	1	8	20	8	40	
3	1	8	40	9	0	
4	1	9	0	9	20	
5	1	9	20	9	40	
6	1	9	40	10	0	
7	1	10	0	10	20	
8	1	10	20	10	40	
9	1	10	40	11	0	
10	1	11	0	11	20	
11	1	11	20	11	40	
12	1	11	40	12	0	
13	1	14	0	14	20	
14	1	14	20	14	40	
15	1	14	40	15	0	
16	1	1	15	0	15	20
17	1	1	15	20	15	40
18	1	1	15	40	16	0
19	1	1	16	0	16	20
20	1	1	16	20	16	40
21	1	1	16	40	17	0
22	1	1	17	0	17	20
23	1	1	17	20	17	40
24	1	1	17	40	18	0
25	1	2	8	0	8	20
26	1	2	8	20	8	40
27	1	2	8	40	9	0
28	1	2	9	0	9	20
29	1	2	9	20	9	40
30	1	2	9	40	10	0
31	1	2	10	0	10	20
32	1	2	10	20	10	40
33	1	2	10	40	12	0
34	1	2	12	0	12	20
35	1	2	12	20	12	40
36	1	2	12	40	12	0
37	1	3	8	0	8	20
38	1	3	8	20	8	40
39	1	3	8	40	9	0
40	1	3	9	0	9	20
41	1	3	9	20	9	40
42	1	3	9	40	10	0
43	1	3	10	0	10	20
44	1	3	10	20	10	40
45	1	3	10	40	12	0
46	1	3	12	0	12	20

The screenshot shows the MySQL Workbench interface with two panes. The left pane displays the table structure with columns: ID, VERSION, ID_MEDECIN, HDEBUT, MDEBUT, HFIN, and MFIN. The right pane shows the data with 46 rows:

ID	VERSION	ID_MEDECIN	HDEBUT	MDEBUT	HFIN	MFIN
1	1	1	8	0	8	40
2	1	1	8	20	8	40
3	1	1	8	40	9	0
4	1	1	9	0	9	20
5	1	1	9	20	9	40
6	1	1	9	40	10	0
7	1	1	10	0	10	20
8	1	1	10	20	10	40
9	1	1	10	40	11	0
10	1	1	11	0	11	20
11	1	1	11	20	11	40
12	1	1	11	40	12	0
13	1	1	14	0	14	20
14	1	1	14	20	14	40
15	1	1	14	40	15	0
16	1	1	15	0	15	20
17	1	1	15	20	15	40
18	1	1	15	40	16	0
19	1	1	16	0	16	20
20	1	1	16	20	16	40
21	1	1	16	40	17	0
22	1	1	17	0	17	20
23	1	1	17	20	17	40
24	1	1	17	40	18	0
25	1	2	8	0	8	20
26	1	2	8	20	8	40
27	1	2	8	40	9	0
28	1	2	9	0	9	20
29	1	2	9	20	9	40
30	1	2	9	40	10	0
31	1	2	10	0	10	20
32	1	2	10	20	10	40
33	1	2	10	40	12	0
34	1	2	12	0	12	20
35	1	2	12	20	12	40
36	1	2	12	40	12	0
37	1	3	8	0	8	20
38	1	3	8	20	8	40
39	1	3	8	40	9	0
40	1	3	9	0	9	20
41	1	3	9	20	9	40
42	1	3	9	40	10	0
43	1	3	10	0	10	20
44	1	3	10	20	10	40
45	1	3	10	40	12	0
46	1	3	12	0	12	20

- ID : n° identifiant le créneau horaire - clé primaire de la table (ligne 8)
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- ID_MEDECIN : n° identifiant le médecin auquel appartient ce créneau – clé étrangère sur la colonne MEDECINS(ID).
- HDEBUT : heure début créneau
- MDEBUT : minutes début créneau
- HFIN : heure fin créneau
- MFIN : minutes fin créneau

La seconde ligne de la table [CRENEAUX] (cf [1] ci-dessus) indique, par exemple, que le créneau n° 2 commence à 8 h 20 et se termine à 8 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER).

3.4.4 La table [RV]

Elle liste les RV pris pour chaque médecin :

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null
JOUR	DATE	10	0	<input checked="" type="checkbox"/>	
ID_CLIENT	BIGINT	20	0	<input checked="" type="checkbox"/>	
ID_CRENEAU	BIGINT	20	0	<input checked="" type="checkbox"/>	

ID	JOUR	ID_CLIENT	ID_CRENEAU
1	22/08/2006	2	1
3	23/08/2006	4	20
4	10/09/2006	2	10
6	23/08/2006	3	7
9	23/08/2006	2	10

- ID : n° identifiant le RV de façon unique – clé primaire
- JOUR : jour du RV
- ID_CRENEAU : créneau horaire du RV - clé étrangère sur le champ [ID] de la table [CRENEAUX] – fixe à la fois le créneau horaire et le médecin concerné.
- ID_CLIENT : n° du client pour qui est faite la réservation – clé étrangère sur le champ [ID] de la table [CLIENTS]

Cette table a une contrainte d'unicité sur les valeurs des colonnes jointes (JOUR, ID_CRENEAU) :

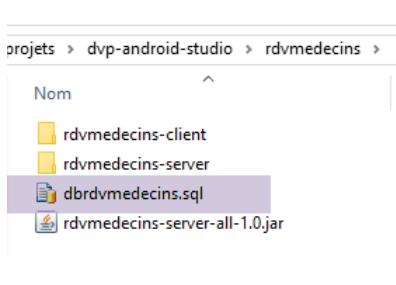
```
|ALTER TABLE RV ADD CONSTRAINT UNQ1_RV UNIQUE (JOUR, ID_CRENEAU);|
```

Si une ligne de la table[RV] a la valeur (JOUR1, ID_CRENEAU1) pour les colonnes (JOUR, ID_CRENEAU), cette valeur ne peut se retrouver nulle part ailleurs. Sinon, cela signifierait que deux RV ont été pris au même moment pour le même médecin. D'un point de vue programmation Java, le pilote JDBC de la base lance une *SQLException* lorsque ce cas se produit.

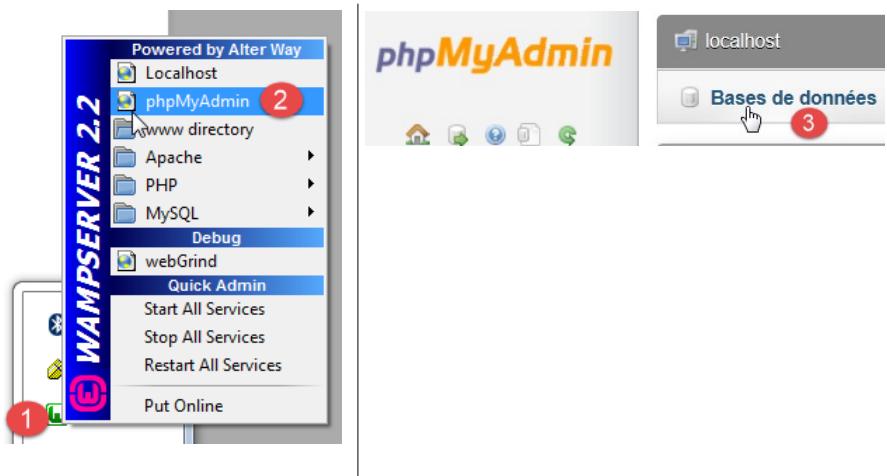
La ligne d'*id* égal à 3 (cf [1] ci-dessus) signifie qu'un RV a été pris pour le créneau n° 20 et le client n° 4 le 23/08/2006. La table [CRENEAUX] nous apprend que le créneau n° 20 correspond au créneau horaire 16 h 20 - 16 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER). La table [CLIENTS] nous apprend que le client n° 4 est Melle Brigitte BISTROU.

3.4.5 Génération de la base

Pour créer les tables et les remplir on pourra utiliser le script [dbrdvmmedecins.sql] qu'on trouvera dans l'archive des exemples [<http://tahe.developpez.com/tutoriels-cours/programmation-android-avec-android-studio-debutant/documents/dvp-android-studio.rar>].



Avec [WampServer] (cf paragraphe 6.15, page 504), on pourra procéder comme suit :



- en [1], on clique sur l'icône de [WampServer] et on choisit l'option [PhpMyAdmin] [2],
- en [3], dans la fenêtre qui s'est ouverte, on sélectionne le lien [Bases de données],

- en [4-6], on importe un fichier SQL,

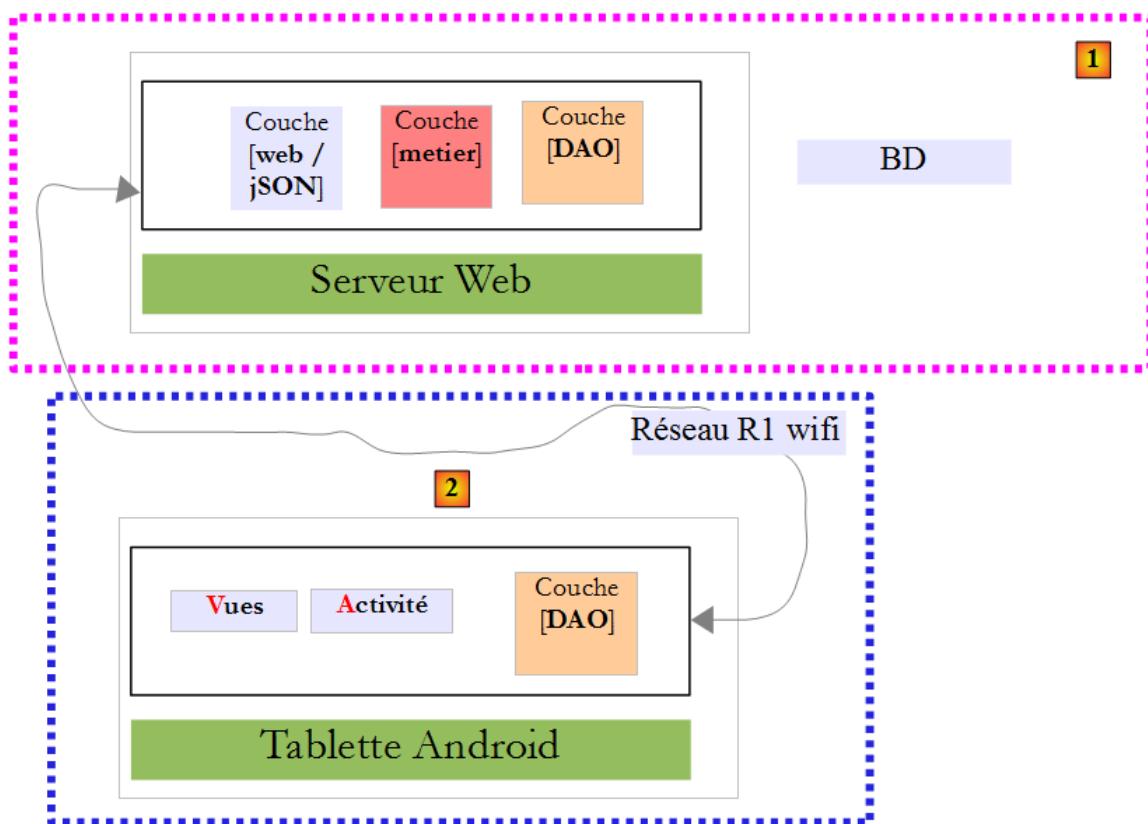
- en [7], on sélectionne le script SQL et en [8] on l'exécute,
- en [9], les tables de la base ont été créées. On suit l'un des liens,

	ID	TITRE	NOM	VERSION	PRENOM
<input type="checkbox"/> Modifier	<input type="checkbox"/> Éditer en place	Copier	Effacer	1	Mr MARTIN
<input type="checkbox"/> Modifier	<input type="checkbox"/> Éditer en place	Copier	Effacer	2	Mme GERMAN
<input type="checkbox"/> Modifier	<input type="checkbox"/> Éditer en place	Copier	Effacer	3	Mr JACQUARD
<input type="checkbox"/> Modifier	<input type="checkbox"/> Éditer en place	Copier	Effacer	4	Melle BISTROU

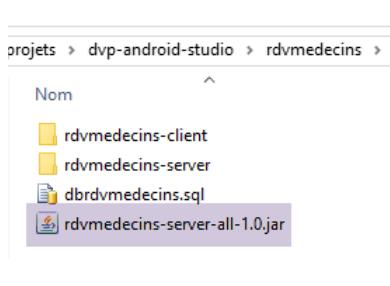
- en [10], le contenu de la table.

Par la suite, nous ne reviendrons plus sur cette base. Mais le lecteur est invité à suivre son évolution au fil des tests surtout lorsque l'application ne marche pas.

3.5 Le serveur web / JSON



Nous nous intéressons ici au serveur [1]. Nous n'allons pas le développer. Celui-ci a été détaillé dans le document [[Spring MVC et Thymeleaf par l'exemple](#)]. Le lecteur intéressé pourra s'y reporter. Il a été développé comme celui du serveur de l'exemple 15. Son code source est livré dans les exemples. Nous allons utiliser ici son binaire :



- [rdvmedecins-server-all-1.0.jar] est le binaire du serveur ;

3.5.1 Mise en oeuvre

Dans une fenêtre de commandes, on se place dans le dossier contenant le binaire du serveur :

```
1. ...\\rdvmedecins>dir
2. Le volume dans le lecteur D s'appelle Données
3. Le numéro de série du volume est 7A34-AE5F
4.
5. Répertoire de D:\\data\\istia-1516\\projets\\dvp-android-studio\\rdvmedecins
6.
7. 09/06/2016 10:50    <DIR>        .
8. 09/06/2016 10:50    <DIR>        ..
9. 06/07/2014 16:36           7 631 dbrdvmedecins.sql
10. 08/06/2016 16:31   <DIR>      rdvmedecins-client
11. 08/06/2016 16:22   <DIR>      rdvmedecins-server
12. 08/06/2016 16:23  29 618 709 rdvmedecins-server-all-1.0.jar
```

puis pour lancer le serveur on tape la commande suivante (le SGBD MySQL doit être déjà lancé) :

```

34. 10:55:56.257 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped
    "[[/ajouterRv],methods=[POST],consumes=[application/json;charset=UTF-8],produces=[application/json;charset=UTF-8]}" onto
    public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.ajouterRv(rdvmedecins.models.PostAjouterRv,javax.servlet.http.HttpServletRes-
    onse,java.lang.String) throws com.fasterxml.jackson.core.JsonProcessingException
35. 10:55:56.259 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped
    "[[/getAllClients],methods=[GET],produces=[application/json;charset=UTF-8]}" onto
    public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.getAllClients(javax.servlet.http.HttpServletResponse,java.lang.String) throws
    com.fasterxml.jackson.core.JsonProcessingException
36. 10:55:56.261 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped "[[/getClientById/
    {id}],methods=[GET],produces=[application/json;charset=UTF-8]}"
37. 10:55:56.261 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped "[[/getClientById/
    {id}],methods=[GET],produces=[application/json;charset=UTF-8]}"
38. onto public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.getClientById(long,javax.servlet.http.HttpServletResponse,java.lang.String)
    throws com.fasterxml.jackson.core.JsonProcessingException
39. 10:55:56.264 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped "[[/getMedecinById/
    {id}],methods=[GET],produces=[application/json;charset=UTF-8]}" onto public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.getMedecinById(long,javax.servlet.http.HttpServletResponse,java.lang.String)
    throws com.fasterxml.jackson.core.JsonProcessingException
40. 10:55:56.266 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped "[[/getRvById/
    {id}],methods=[GET],produces=[application/json;charset=UTF-8]}" onto public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.getRvById(long,javax.servlet.http.HttpServletResponse,java.lang.String)
    throws com.fasterxml.jackson.core.JsonProcessingException
41. 10:55:56.268 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped
    "[[/getAllMedecins],methods=[GET],produces=[application/json;charset=UTF-8]}" onto public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.getAllMedecins(javax.servlet.http.HttpServletResponse,java.lang.String)
    throws com.fasterxml.jackson.core.JsonProcessingException
42. 10:55:56.270 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped
    "[[/supprimerRv],methods=[POST],consumes=[application/json;charset=UTF-8],produces=[application/json;charset=UTF-8]}" onto
    public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.supprimerRv(rdvmedecins.models.PostSupprimerRv,javax.servlet.http.HttpServlet
    Response,java.lang.String) throws com.fasterxml.jackson.core.JsonProcessingException
43. 10:55:56.273 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped
    "[[/authenticate],methods=[GET],produces=[application/json;charset=UTF-8]}" onto public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.authenticate(javax.servlet.http.HttpServletResponse,java.lang.String) throws
    com.fasterxml.jackson.core.JsonProcessingException
44. 10:55:56.276 [main] INFO o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped "[[/getAgendaMedecinJour/{idMedecin}/
    {jour}],methods=[GET],produces=[application/json;charset=UTF-8]}" onto public java.lang.String
    rdvmedecins.controllers.RdvMedecinsController.getAgendaMedecinJour(long,java.lang.String,javax.servlet.http.HttpServletRes-
    onse,java.lang.String) throws com.fasterxml.jackson.core.JsonProcessingException
45. ...
46. 10:55:56.681 [main] INFO o.s.b.c.e.t.TomcatEmbeddedServletContainer - Tomcat started on port(s): 8080 (http)
47. 10:55:56.686 [main] INFO rdvmedecins.boot.Boot - Started Boot in 8.231 seconds

```

Le serveur affiche de nombreux logs. Nous n'avons retenu ci-dessus que ceux utiles à comprendre :

- lignes 14-18 : un serveur Tomcat embarqué est lancé sur le port 8080 de la machine. C'est ce serveur qui exécute l'application web de gestion des rendez-vous. Cette application est en fait un service web / JSON : elle est interrogée via des URL et elle répond en envoyant une chaîne JSON ;
- ligne 24 : le service web est sécurisé avec le framework [Spring Security]. On accède aux URL du service web en s'authentifiant ;
- lignes 29-44 : les URL exposées par le service web ;

Nous allons détailler ces dernières.

3.5.2 Sécurisation du service web

Les URL exposées par le service web sont sécurisées. Le serveur attend dans la requête HTTP du client l'entête suivant :

Authorization: Basic code

Le code attendu est le codage en base64 [<http://fr.wikipedia.org/wiki/Base64>] de la chaîne 'utilisateur:motdepasse'. Le service web n'accepte dans son état initial qu'un utilisateur 'admin' avec le mot de passe 'admin'. L'entête ci-dessus devient pour cet utilisateur la ligne suivante :

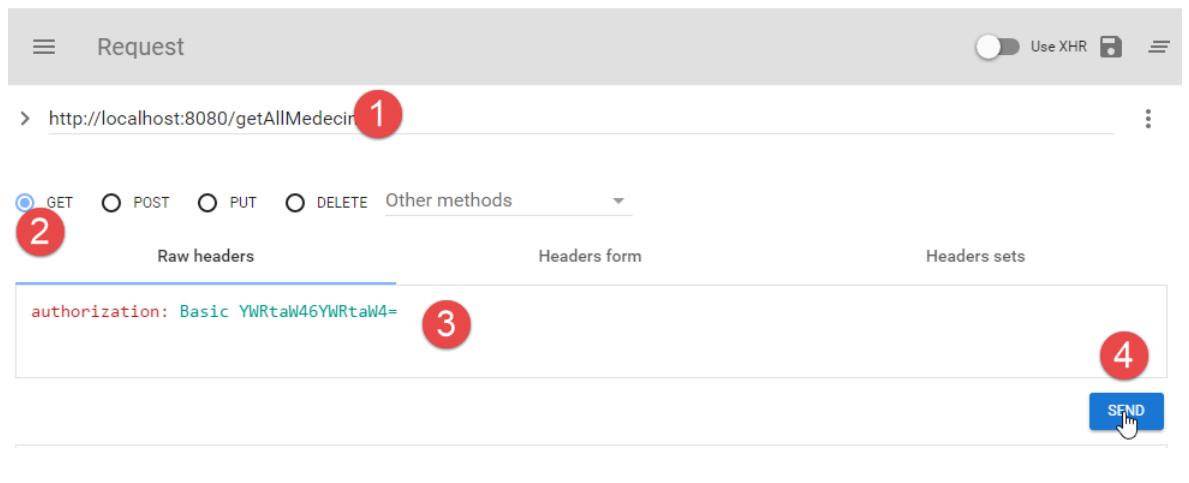
Authorization: Basic YWRtaW46YWRtaW4=

Afin de pouvoir envoyer cet entête HTTP, nous utilisons le client HTTP [[Advanced Rest Client](#)] qui est un plugin du navigateur Chrome (cf paragraphe 6.13, page 501). Nous allons tester à la main les différentes URL exposées par le service web afin de comprendre :

- les paramètres attendus par l'URL ;
- la nature exacte de sa réponse ;

3.5.3 Liste des médecins

L'URL [/getAllMedecins] permet d'obtenir la liste des médecins :



- en [1], l'URL interrogée ;
- en [2], la méthode HTTP utilisée pour cette interrogation ;
- en [3], l'en-tête HTTP de sécurité de l'utilisateur (admin, admin) ;
- en [4], on envoie la requête HTTP ;

La réponse du serveur est la suivante :

```
{
  "status": 0,
  "messages": null
- "body": [4]
  - 0: {
    "id": 1
    "version": 1
    "titre": "Mme"
    "nom": "PELISSIER"
    "prenom": "Marie"
  }
  - 1: {
    "id": 2
    "version": 1
    "titre": "Mr"
    "nom": "BROMARD"
    "prenom": "Jacques"
  }
  - 2: {
    "id": 3
    "version": 1
    "titre": "Mr"
    "nom": "JANDOT"
  }
  - 3: {
    "id": 4
    "version": 1
    "titre": "Melle"
    "nom": "JACQUEMOT"
    "prenom": "Justine"
  }
}
```

- en [5], la réponse JSON du serveur, mise en forme ;

```
{"status":0,"messages":null,"body":[{"id":1,"version":1,"titre":"Mme","nom":"PELISSIER","prenom":"Marie"}, {"id":2,"version":1,"titre":"Mr","nom":"BROMARD","prenom":"Jacques"}, {"id":3,"version":1,"titre":"Mr","nom":"JANDOT","prenom":"Philippe"}, {"id":4,"version":1,"titre":"Melle","nom":"JACQUEMOT","prenom":"Justine"}]}
```

- en [6], la même réponse à l'état brut ;

La forme [5] permet de mieux voir la structure de la réponse. Toutes les réponses du service web sont une instance de la classe [Response] suivante :

```

1. package rdvmedecins.android.dao.service;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.     // ----- propriétés
8.     // statut de l'opération
9.     private int status;
10.    // les éventuels messages d'erreur
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.    ...
28. }
```

- ligne 9 : le statut de la réponse. La valeur 0 veut dire qu'il n'y a pas eu d'erreur, sinon il y a eu erreur ;
- ligne 11 : une liste de messages d'erreur s'il y a eu erreur ;
- ligne 13 : la réponse réellement attendue par le client ;

La réponse à l'URL [/getAllMedecins] est la chaîne JSON d'un objet de type [Response<List<Medecin>>]. La classe [Medecin] est la suivante :

```

1. package rdvmedecins.android.dao.entities;
2.
3. public class Medecin extends Personne {
4.
5.     // constructeur par défaut
6.     public Medecin() {
7.     }
8.
9.     // constructeur avec paramètres
10.    public Medecin(String titre, String nom, String prenom) {
11.        super(titre, nom, prenom);
12.    }
13.
14.    public String toString() {
15.        return String.format("Medecin[%s]", super.toString());
16.    }
17.
18. }
```

Ligne 3, la classe [Medecin] étend la classe [Personne] suivante :

```

1. package rdvmedecins.android.dao.entities;
2.
3. public class Personne extends AbstractEntity {
4.     // attributs d'une personne
5.     private String titre;
6.     private String nom;
7.     private String prenom;
8.
9.     // constructeur par défaut
10.    public Personne() {
11.    }
12.
13.    // constructeur avec paramètres
14.    public Personne(String titre, String nom, String prenom) {
15.        this.titre = titre;
16.        this.nom = nom;
17.        this.prenom = prenom;
18.    }
19. }
```

```

20.     // toString
21.     public String toString() {
22.         return String.format("Personne[%s, %s, %s, %s, %s]", id, version, titre, nom, prenom);
23.     }
24.
25.     // getters et setters
26.     ...
27. }
```

Ligne 3, la classe [Personne] étend la classe [AbstractEntity] suivante :

```

1. package rdvmedecins.android.dao.entities;
2.
3. import java.io.Serializable;
4.
5. public class AbstractEntity implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     protected Long id;
9.     protected Long version;
10.
11.    @Override
12.    public int hashCode() {
13.        int hash = 0;
14.        hash += (id != null ? id.hashCode() : 0);
15.        return hash;
16.    }
17.
18.    // initialisation
19.    public AbstractEntity build(Long id, Long version) {
20.        this.id = id;
21.        this.version = version;
22.        return this;
23.    }
24.
25.    @Override
26.    public boolean equals(Object entity) {
27.        String class1 = this.getClass().getName();
28.        String class2 = entity.getClass().getName();
29.        if (!class2.equals(class1)) {
30.            return false;
31.        }
32.        AbstractEntity other = (AbstractEntity) entity;
33.        return this.id == other.id;
34.    }
35.
36.    // getters et setters
37.    ...
38. }
```

Au final, la structure d'un objet [Medecin] est la suivante :

```
[Long id; Long version; String titre; String nom; String prenom;]
```

et celle de [Response<List<Medecin>>] la suivante :

```
[int status; List<String> messages; List<Medecin> medecins]
```

Par la suite, nous utiliserons ces définitions raccourcies pour caractériser la réponse du serveur. Par ailleurs, pendant un certain temps, nous ne montrerons plus de copies d'écran. Il suffit de répéter ce que nous venons de voir. Nous reviendrons aux copies d'écran lorsqu'il faudra faire une requête POST. Nous présenterons également un exemple d'exécution sous la forme suivante :

URL	/getAllMedecins
Réponse	{"status":0,"messages":null,"medecins":[{"id":1,"version":1,"titre":"Mme","nom":"PELISSIER","prenom":"Marie"}, {"id":2,"version":1,"titre":"Mr","nom":"BROMARD","prenom":"Jacques"}, {"id":3,"version":1,"titre":"Mr","nom":"JANDOT","prenom":"Philippe"}, {"id":4,"version":1,"titre":"Melle","nom":"JACQUEMOT","prenom":"Justine"}]}

3.5.4 Liste des clients

URL	/getAllClients
Réponse	Response<List<Client>> :[int status; List<String> messages; List<Client> clients] Client : [Long id; Long version; String titre; String nom; String prenom;]

Exemple :

URL	/getAllClients
Réponse	{"status":0,"messages":null,"clients":[{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}, {"id":2,"version":1,"titre":"Mme","nom":"GERMAN","prenom":"Christine"}, {"id":3,"version":1,"titre":"Mr","nom":"JACQUARD","prenom":"Jules"}, {"id":4,"version":1,"titre":"Melle","nom":"BISTROU","prenom":"Brigitte"}]}

3.5.5 Liste des créneaux d'un médecin

URL	/getAllCreneaux/{idMedecin}
Réponse	Response<List<Creneau>>:[int status ; List<String> messages ; List<Creneau> creneaux] Creneau : [int hdebut ; int mdebut ; int hfin ; int mfin ;]

- [idMedecin] : identifiant du médecin dont on veut les créneaux horaires de consultation ;
- [hdebut] : heure de début de la consultation ;
- [mdebut] : minutes de début de la consultation ;
- [hfin] : heure de fin de la consultation ;
- [mfin] : minutes de fin de la consultation ;

Pour un créneau entre 10h20 et 10h40 on aura [hdebut, mdebut, hfin, mfin]=[10, 20, 10, 40].

Exemple :

URL	/getAllCreneaux/1
Réponse	{"status":0,"messages":null,"creneaux":[{"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1}, {"id":2,"version":1,"hdebut":8,"mdebut":20,"hfin":8,"mfin":40,"idMedecin":1}, {"id":3,"version":1,"hdebut":8,"mdebut":40,"hfin":9,"mfin":0,"idMedecin":1}, {"id":4,"version":1,"hdebut":9,"mdebut":0,"hfin":9,"mfin":20,"idMedecin":1}, {"id":5,"version":1,"hdebut":9,"mdebut":20,"hfin":9,"mfin":40,"idMedecin":1}, {"id":6,"version":1,"hdebut":9,"mdebut":40,"hfin":10,"mfin":0,"idMedecin":1}, {"id":7,"version":1,"hdebut":10,"mdebut":0,"hfin":10,"mfin":20,"idMedecin":1}, {"id":8,"version":1,"hdebut":10,"mdebut":20,"hfin":10,"mfin":40,"idMedecin":1}, {"id":9,"version":1,"hdebut":10,"mdebut":40,"hfin":11,"mfin":0,"idMedecin":1}, {"id":10,"version":1,"hdebut":11,"mdebut":0,"hfin":11,"mfin":20,"idMedecin":1}, {"id":11,"version":1,"hdebut":11,"mdebut":20,"hfin":11,"mfin":40,"idMedecin":1}, {"id":12,"version":1,"hdebut":11,"mdebut":40,"hfin":12,"mfin":0,"idMedecin":1}, {"id":13,"version":1,"hdebut":14,"mdebut":0,"hfin":14,"mfin":20,"idMedecin":1}, {"id":14,"version":1,"hdebut":14,"mdebut":20,"hfin":14,"mfin":40,"idMedecin":1}, {"id":15,"version":1,"hdebut":14,"mdebut":40,"hfin":15,"mfin":0,"idMedecin":1}, {"id":16,"version":1,"hdebut":15,"mdebut":0,"hfin":15,"mfin":20,"idMedecin":1}, {"id":17,"version":1,"hdebut":15,"mdebut":20,"hfin":15,"mfin":40,"idMedecin":1}, {"id":18,"version":1,"hdebut":15,"mdebut":40,"hfin":16,"mfin":0,"idMedecin":1}, {"id":19,"version":1,"hdebut":16,"mdebut":0,"hfin":16,"mfin":20,"idMedecin":1}, {"id":20,"version":1,"hdebut":16,"mdebut":20,"hfin":16,"mfin":40,"idMedecin":1}, {"id":21,"version":1,"hdebut":16,"mdebut":40,"hfin":17,"mfin":0,"idMedecin":1}, {"id":22,"version":1,"hdebut":17,"mdebut":0,"hfin":17,"mfin":20,"idMedecin":1}, {"id":23,"version":1,"hdebut":17,"mdebut":20,"hfin":17,"mfin":40,"idMedecin":1}, {"id":24,"version":1,"hdebut":17,"mdebut":40,"hfin":18,"mfin":0,"idMedecin":1}]]

3.5.6 Liste des rendez-vous d'un médecin

URL	/getRvMedecinJour/{idMedecin}/{jour}
Réponse	Response<List<Rv>>:[int status ; List<String> messages ; List<Rv> rvs] Rv : [Date jour ; Client client ; Creneau creneau ; long idClient ; long idCreneau]

- [idMedecin] : identifiant du médecin dont on veut les rendez-vous ;
- URL [jour] : jour des rendez-vous sous la forme 'aaaa-mm-jj' ;
- Réponse [jour] : idem mais sous la forme d'une date Java ;
- [client] : le client du rendez-vous. Sa structure a été décrite précédemment ;
- [idClient] : l'identifiant du client ;
- [creneau] : le créneau du rendez-vous. Sa structure a été décrite précédemment ;
- [idCreneau] : l'identifiant du créneau ;

Exemple :

URL	/getRvMedecinJour/1/2014-07-08
Réponse	{"status":0,"messages":null,"rvs":[{"id":45,"version":0,"jour":"2014-07-08","client":{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"},"creneau":{"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1,"idClient":1,"idCreneau":1}]}]

3.5.7 L'agenda d'un médecin

URL	/getAgendaMedecinJour/{idMedecin}/{jour}
Réponse	Response<AgendaMedecinJour>:[int status ; List<String> messages ; AgendaMedecinJour agenda] AgendaMedecinJour : [Medecin medecin ; Date jour ; CreneauMedecinJour[] creneauxMedecinJour] CreneauMedecinJour : [Creneau creneau ; Rv rv]

- [idMedecin] : identifiant du médecin dont on veut les rendez-vous ;
- URL [jour] : jour des rendez-vous sous la forme 'aaaa-mm-jj' ;
- [agenda] : agenda du médecin ;
- [medecin] : le médecin concerné. Sa structure a été définie précédemment ;
- Réponse [jour] : le jour de l'agenda sous la forme d'une date Java ;
- [creneauxMedecinJour] : un tableau d'éléments de type [CreneauMedecinJour] ;
- [creneau] : un créneau. Sa structure a été décrite précédemment ;
- [rv] : un rendez-vous. Sa structure a été décrite précédemment ;

Exemple :

URL	/getAgendaMedecinJour/1/2014-07-08
Réponse	{"status":0,"messages":null,"agenda":{"medecin": {"id":1,"version":1,"titre":"Mme","nom":"PELISSIER","prenom":"Marie"},"jour":1404770400000,"creneauxMedecinJour": [{"creneau":{"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1}, "rv": [{"id":45,"version":0,"jour":"2014-07-08","client": {"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}, "creneau": {"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1}, "idClient":1,"idCreneau":1}, {"creneau": {"id":2,"version":1,"hdebut":8,"mdebut":20,"hfin":8,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":3,"version":1,"hdebut":8,"mdebut":40,"hfin":9,"mfin":0,"idMedecin":1}, "rv":null}, {"creneau": {"id":4,"version":1,"hdebut":9,"mdebut":0,"hfin":9,"mfin":20,"idMedecin":1}, "rv":null}, {"creneau": {"id":5,"version":1,"hdebut":9,"mdebut":9,"hfin":9,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":6,"version":1,"hdebut":9,"mdebut":40,"hfin":10,"mfin":0,"idMedecin":1}, "rv":null}, {"creneau": {"id":7,"version":1,"hdebut":10,"mdebut":0,"hfin":10,"mfin":20,"idMedecin":1}, "rv":null}, {"creneau": {"id":8,"version":1,"hdebut":10,"mdebut":20,"hfin":10,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":9,"version":1,"hdebut":10,"mdebut":40,"hfin":11,"mfin":0,"idMedecin":1}, "rv":null}, {"creneau": {"id":10,"version":1,"hdebut":11,"mdebut":0,"hfin":11,"mfin":20,"idMedecin":1}, "rv":null}, {"creneau": {"id":11,"version":1,"hdebut":11,"mdebut":20,"hfin":11,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":12,"version":1,"hdebut":11,"mdebut":40,"hfin":12,"mfin":0,"idMedecin":1}, "rv":null}, {"creneau": {"id":13,"version":1,"hdebut":14,"mdebut":0,"hfin":14,"mfin":20,"idMedecin":1}, "rv":null}, {"creneau": {"id":14,"version":1,"hdebut":14,"mdebut":20,"hfin":14,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":15,"version":1,"hdebut":14,"mdebut":40,"hfin":15,"mfin":0,"idMedecin":1}, "rv":null}, {"creneau": {"id":16,"version":1,"hdebut":15,"mdebut":0,"hfin":15,"mfin":20,"idMedecin":1}, "rv":null}, {"creneau": {"id":17,"version":1,"hdebut":15,"mdebut":20,"hfin":15,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":18,"version":1,"hdebut":15,"mdebut":40,"hfin":16,"mfin":0,"idMedecin":1}, "rv":null}, {"creneau": {"id":19,"version":1,"hdebut":16,"mdebut":0,"hfin":16,"mfin":20,"idMedecin":1}, "rv":null}, {"creneau": {"id":20,"version":1,"hdebut":16,"mdebut":20,"hfin":16,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":21,"version":1,"hdebut":16,"mdebut":40,"hfin":17,"mfin":0,"idMedecin":1}, "rv":null}, {"creneau": {"id":22,"version":1,"hdebut":17,"mdebut":0,"hfin":17,"mfin":20,"idMedecin":1}, "rv":null}, {"creneau": {"id":23,"version":1,"hdebut":17,"mdebut":20,"hfin":17,"mfin":40,"idMedecin":1}, "rv":null}, {"creneau": {"id":24,"version":1,"hdebut":17,"mdebut":40,"hfin":18,"mfin":0,"idMedecin":1}, "rv":null}]}]}

On a mis en exergue le cas où il y a un rendez-vous dans le créneau et le cas où il n'y en a pas.

3.5.8 Obtenir un médecin par son identifiant

URL	/getMedecinById/{idMedecin}
Réponse	Response<Medecin> :[int status ; List<String> messages ; Medecin medecin]

- [idMedecin] : l'identifiant du médecin ;

Exemple 1 :

URL	/getMedecinById/1
Réponse	{"status":0,"messages":null,"medecin": {"id":1,"version":1,"titre":"Mme","nom":"PELISSIER","prenom":"Marie"}}

Exemple 2 :

URL	/getMedecinById/100
Réponse	{"status":2,"messages":["Médecin [100] inexistant"],"medecin":null}

3.5.9 Obtenir un client par son identifiant

URL	/getClientById/{idClient}
Réponse	Response<Client> :[int status ; List<String> messages ; Client client]

- [idClient] : l'identifiant du client ;

Exemple 1 :

URL	/getClientById/1
Réponse	{"status":0,"messages":null,"client":{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}}

Exemple 2 :

URL	/getClientById/100
Réponse	{"status":2,"messages":["Client [100] inexistant"],"client":null}

3.5.10 Obtenir un créneau par son identifiant

URL	/getCreneauById/{idCreneau}
Réponse	Response<Creneau> :[int status ; List<String> messages ; Creneau creneau]

- [idCreneau] : l'identifiant du créneau ;

Exemple 1 :

URL	/getCreneauById/10
Réponse	{"status":0,"messages":null,"creneau":{"id":10,"version":1,"hdebut":11,"mdebut":0,"hfin":11,"mfin":20,"idMedecin":1}}

On remarquera que dans la réponse, il n'y a pas le médecin propriétaire du créneau mais seulement son identifiant.

Exemple 2 :

URL	/getCreneauById/100
Réponse	{"status":2,"messages":["Créneau [100] inexistant"],"creneau":null}

3.5.11 Obtenir un rendez-vous par son identifiant

URL	/getRvById/{idRv}
Réponse	Response<Rv> :[int status ; List<String> messages ; Rv rv]

- [idRv] : l'identifiant du rendez-vous ;

Exemple 1 :

URL	/getRvById/45
Réponse	{"status":0,"messages":null,"rv":{"id":45,"version":0,"jour":"2014-07-08","idClient":1,"idCreneau":1}}

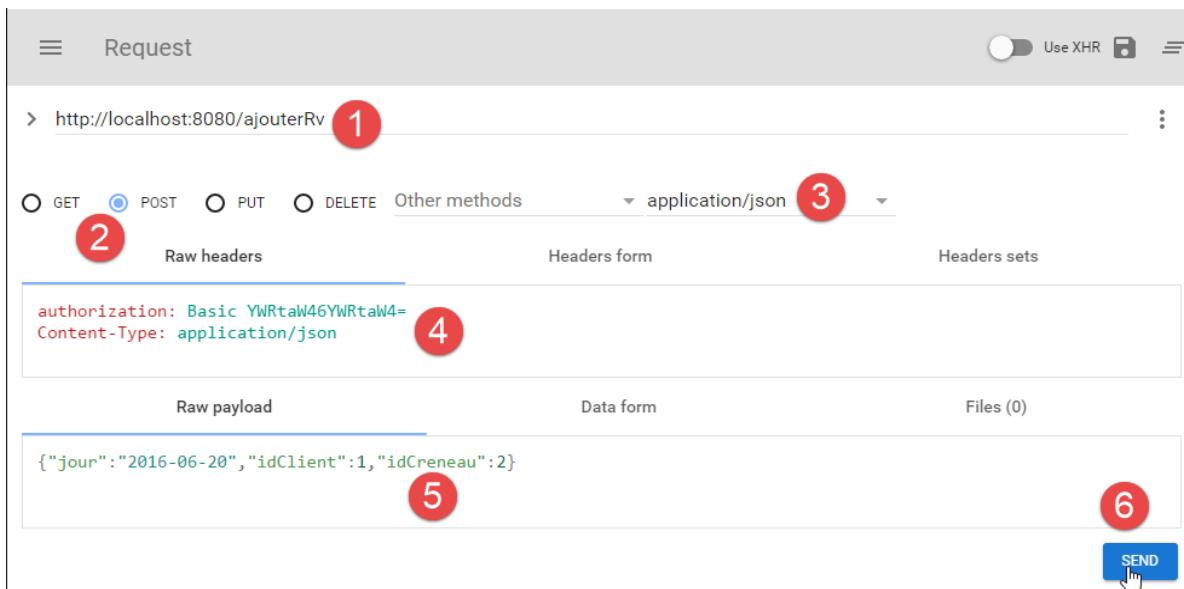
On remarquera que dans la réponse, il n'y a ni le client, ni le créneau du rendez-vous mais seulement leurs identifiants.

Exemple 2 :

URL	/getRvById/455
Réponse	{"status":2,"messages":["Rv [455] inexistant"],"rv":null}

3.5.12 Ajouter un rendez-vous

L'URL [/ajouterRv] permet d'ajouter un rendez-vous. Les informations nécessaires à cet ajout (le jour, le créneau et le client) sont transmises via une requête HTTP POST. Nous montrons comment réaliser cette requête avec l'outil [Advanced Rest Client].



- en [1], l'URL interrogée ;
- en [2], elle est interrogée par un POST ;
- en [3-4], on précise au serveur que les valeurs qui lui sont postées le sont sous la forme d'une chaîne JSON ;
- en [4], l'en-tête HTTP de l'authentification ;
- en [5], les informations transmises par le POST. C'est une chaîne JSON contenant :
 - [jour] : le jour du rendez-vous sous la forme 'aaaa-mm-jj',
 - [idClient] : l'identifiant du client pour lequel le rendez-vous est pris,
 - [idCreneau] : l'identifiant du créneau horaire du rendez-vous. Comme un créneau horaire appartient à un médecin précis, on désigne par là également le médecin ;
- en [6], on envoie la requête ;

La chaîne JSON qui est postée est celle de l'objet de type [PostAjouterRv] suivant :

```

1. public class PostAjouterRv {
2.
3.     // données du post
4.     private String jour;
5.     private long idClient;
6.     private long idCreneau;
7.
8.     // constructeurs
9.     public PostAjouterRv() {
10.
11. }
12.
13.     public PostAjouterRv(String jour, long idCreneau, long idClient) {
14.         this.jour = jour;
15.         this.idClient = idClient;
16.         this.idCreneau = idCreneau;
17.     }
18.
19.     // getters et setters
20.     ...
21. }
```

La réponse du serveur est de type [Response<Rv>] [int status; List<String> messages; Rv rv] où [rv] est le rendez-vous ajouté.

La réponse du serveur à la requête plus haut est la suivante :

Raw

```
{
    "status": 0,
    "messages": null,
    "body": {
        "id": 209,
        "version": 0,
        "jour": 1466373600000,
        "client": {
            "id": 1,
            "version": 1,
            "titre": "Mr",
            "nom": "MARTIN",
            "prenom": "Jules"
        },
        "creneau": {
            "id": 2,
            "version": 1,
            "hdebut": 8,
            "mdebut": 20,
            "hfin": 8,
            "mfin": 40,
            "idMedecin": 1
        }
    },
    "idClient": 0,
    "idCreneau": 0
}
```

On notera ci-dessus que certaines informations ne sont pas renseignées [idClient, idCreneau] mais on les trouve dans les champs [client] et [creneau]. L'information importante est l'identifiant du rendez-vous ajouté (209). Le service web aurait pu se contenter de renvoyer cette seule information.

3.5.13 Supprimer un rendez-vous

Cette opération se fait également par un POST :

URL	/supprimerRv
POST	{"idRv":idRv}
Réponse	Response<RV> :[int status ; List<String> messages ; Rv rv]

La valeur postée est la chaîne JSON d'un objet de type [PostSupprimerRv] suivant :

```

1. public class PostSupprimerRv {
2.
3.     // données du post
4.     private long idRv;
5.
6.     // constructeurs
7.     public PostSupprimerRv() {
8.
9.     }
10.
11.    public PostSupprimerRv(long idRv) {
12.        this.idRv = idRv;
13.    }
14.
15.    // getters et setters
16.    ...
17. }
```

- ligne 4, [idRv] est l'identifiant du rendez-vous à supprimer.

Exemple 1 :

URL	/supprimerRv
POST	{"idRv":209}

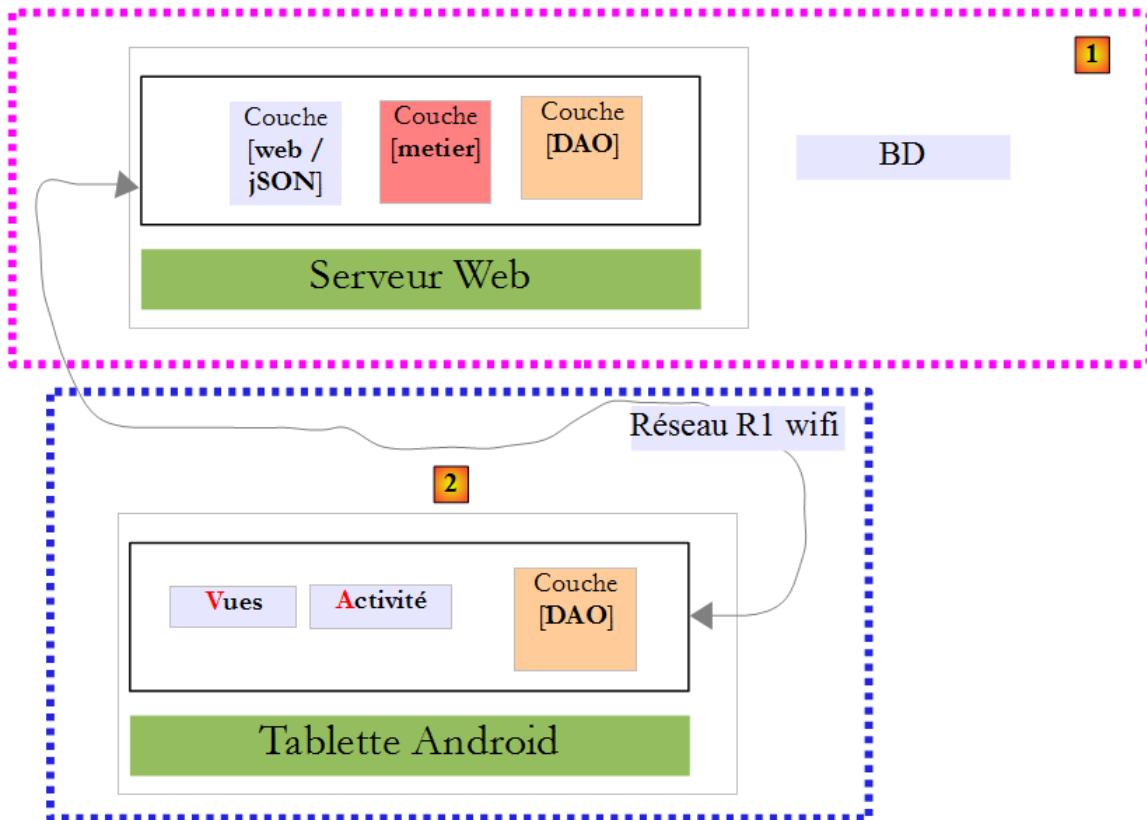
Réponse | {"status":0,"messages":null,"rv":null}

Le rendez-vous de n° 209 a bien été supprimé car [status=0].

Exemple 2 :

URL	/supprimerRv
POST	{"idRv":650}
Réponse	{"status":2,"messages":["Rv [650] inexistant"],"rv":null}

3.6 Le client Android



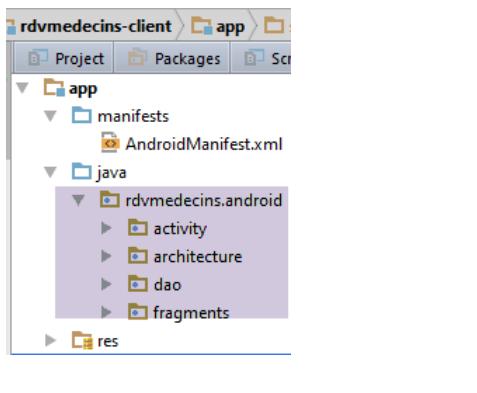
Maintenant que le serveur [1] a été détaillé et est opérationnel, nous allons étudier le client Android [2].

3.6.1 Architecture du projet Android Studio

Le projet reprend l'architecture du projet [client-android-skel] (cf paragraphe 1.17, page 201). Dans l'architecture ci-dessus du client Android, on distingue trois blocs :

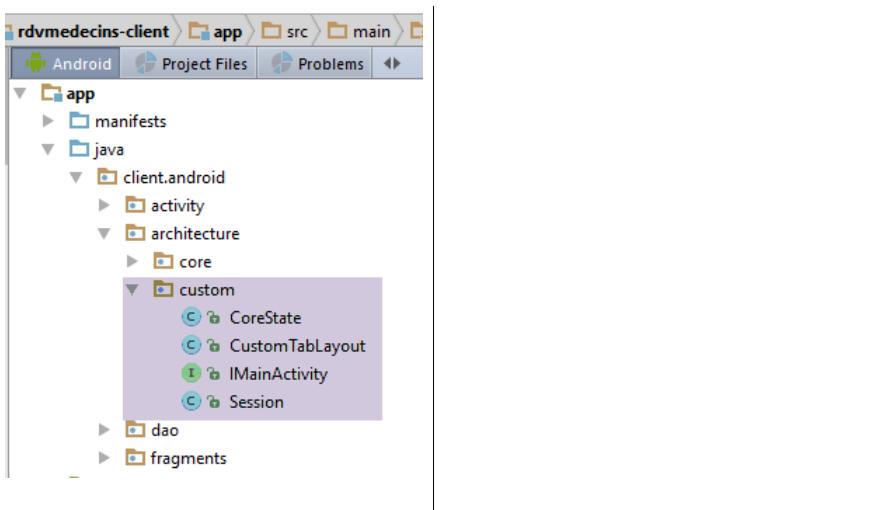
- la couche [DAO] chargée de la communication avec le service web ;
- les [vues] chargées de la communication avec l'utilisateur ;
- l'[activité] qui fait le lien entre les deux blocs précédents. Les vues n'ont pas connaissance de la couche [DAO]. Elles ne communiquent qu'avec l'activité.

Cette architecture est reflétée dans celle du projet Android Studio du client Android :



- le package [activity] implémente l'activité ;
- le package [architecture] reprend les éléments d'architecture que nous avons développée précédemment ;
- le package [dao] implémente la couche [DAO] ;
- le package [fragments] implémente les [vues] ;

3.6.2 Personnalisation du projet



Le dossier [architecture / custom] contient les éléments personnalisables de l'architecture.

L'interface [IMainActivity] est la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.ISession;
4. import client.android.dao.service.IDao;
5.
6. public interface IMainActivity extends IDao {
7.
8.     // accès à la session
9.     ISession getSession();
10.
11.    // changement de vue
12.    void navigateToView(int position, ISession.Action action);
13.
14.    // gestion de l'attente
15.    void beginWaiting();
16.
17.    void cancelWaiting();
18.
19.    // constantes de l'application -----
20.
21.    // mode debug
22.    boolean IS_DEBUG_ENABLED = true;
23.
24.    // délai maximal d'attente de la réponse du serveur
25.    int TIMEOUT = 1000;

```

```

26.
27.    // délai d'attente avant exécution de la requête client
28.    int DELAY = 000;
29.
30.    // authentification basique
31.    boolean IS_BASIC_AUTHENTICATION_NEEDED = true;
32.
33.    // adjacence des fragments
34.    int OFF_SCREEN_PAGE_LIMIT = 1;
35.
36.    // barre d'onglets
37.    boolean ARE_TABS_NEEDED = false;
38.
39.    // image d'attente
40.    boolean IS_WAITING_ICON_NEEDED = true;
41.
42.    // nombre de fragments de l'application
43.    int FRAGMENTS_COUNT = 4;
44.
45.    // n°s de vue
46.    int VUE_CONFIG = 0;
47.    int VUE_ACCUEIL = 1;
48.    int VUE_AGENDA = 2;
49.    int VUE_AJOUT_RV = 3;
50. }

```

- lignes 25, 28 : personnalisation de la couche [DAO] ;
- ligne 31 : cette application fait des accès authentifiés au serveur ;
- ligne 40 : on a besoin d'une image d'attente ;
- ligne 43 : l'application a quatre fragments ;
- lignes 46-49 : les n°s des quatre fragments ;
- ligne 37 : il n'y a pas d'onglets ;

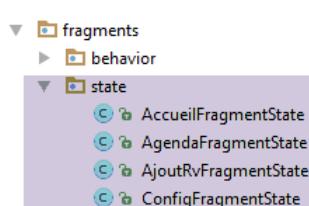
La classe de base des états [CoreState] sera la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.MenuInstanceState;
4. import client.android.fragments.state.AccueilFragmentState;
5. import client.android.fragments.state.AgendaFragmentState;
6. import client.android.fragments.state.AjoutRvFragmentState;
7. import client.android.fragments.state.ConfigFragmentState;
8. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
9. import com.fasterxml.jackson.annotation.JsonSubTypes;
10. import com.fasterxml.jackson.annotation.JsonProperty;
11.
12. @JsonIgnoreProperties(ignoreUnknown = true)
13. @JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
14. @JsonSubTypes({
15.     @JsonSubTypes.Type(value = AccueilFragmentState.class),
16.     @JsonSubTypes.Type(value = AgendaFragmentState.class),
17.     @JsonSubTypes.Type(value = AjoutRvFragmentState.class),
18.     @JsonSubTypes.Type(value = ConfigFragmentState.class)
19. })
20. )
21. public class CoreState {
22.     // fragment visité ou non
23.     protected boolean hasBeenVisited = false;
24.     // état de l'éventuel menu du fragment
25.     protected MenuItemState[] menuOptionsState;
26.
27.     // getters et setters
28.     ...
29. }

```

- lignes 15-18 : les quatre fragments ont un état :

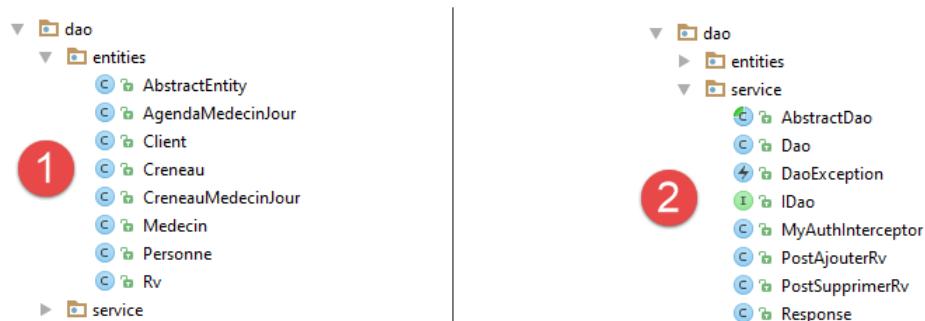
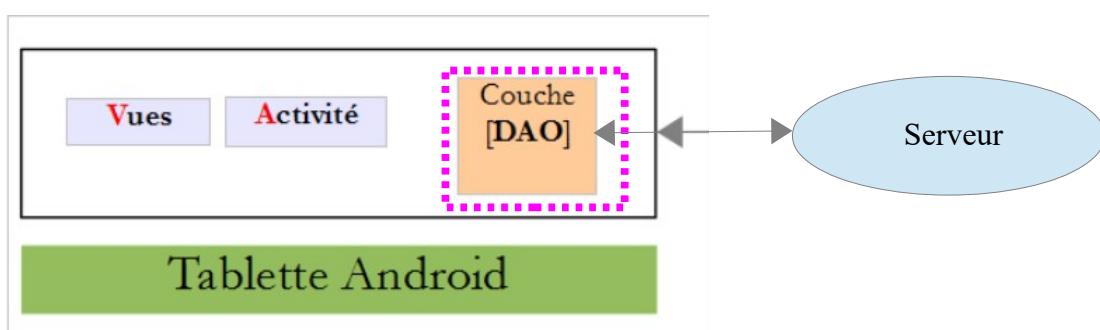


Enfin la session contient les données partagées entre fragments :

```
1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.AbstractSession;
4. import client.android.dao.entities.AgendaMedecinJour;
5. import client.android.dao.entities.Client;
6. import client.android.dao.entities.Medecin;
7. import client.android.fragments.state.AccueilFragmentState;
8. import client.android.fragments.state.AgendaFragmentState;
9. import client.android.fragments.state.AjoutRvFragmentState;
10. import client.android.fragments.state.ConfigFragmentState;
11.
12. import java.util.List;
13.
14. public class Session extends AbstractSession {
15.     // les éléments qui ne peuvent être sérialisés en JSON doivent avoir l'annotation @JsonIgnore
16.
17.     // liste des médecins
18.     private List<Medecin> medecins;
19.     // liste des clients
20.     private List<Client> clients;
21.     // agenda
22.     private AgendaMedecinJour agenda;
23.     // position de l'élément cliqué dans l'agenda
24.     private int position;
25.     // jour du Rv en notation anglaise "yyyy-MM-dd"
26.     private String dayRv;
27.     // jour du Rv en notation française "dd-MM-yyyy"
28.     private String jourRv;
29.
30.     // getters et setters
31.     ...
32. }
```

- lignes 17-28 : la session mémorise six informations. Nous expliquerons le rôle de celles-ci au fil des explications.

3.6.3 La couche [DAO]



- en [1], les entités encapsulées dans les réponses du serveur. Elles ont été présentées au paragraphe 3.5, page 348 ;
- en [2], les éléments du client gérant les échanges avec le serveur ;

Nous n'allons pas revenir sur les éléments [1]. Ils ont déjà été présentés. Le lecteur est invité à revenir au paragraphe 3.5, page 348 si besoin est. Nous allons étudier l'implémentation du package [service]. Cela nous amènera à parler également de l'implémentation des échanges sécurisés entre le client et le serveur.

3.6.3.1 Implémentation des échanges client / serveur



La classe [WebClient] est un composant AA qui décrit :

- les URL exposées par le service web ;
- leurs paramètres ;
- leurs réponses ;

```
1. package rdvmedecins.android.dao.service;
2.
3. import rdvmedecins.android.dao.entities.*;
4. import org.androidannotations.rest.spring.annotations.*;
5. import org.androidannotations.rest.spring.api.RestClientRootUrl;
6. import org.androidannotations.rest.spring.api.RestClientSupport;
7. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
8. import org.springframework.web.client.RestTemplate;
9.
10. import java.util.List;
11.
12. @Rest(converters = {MappingJackson2HttpMessageConverter.class})
13. public interface WebClient extends RestClientRootUrl, RestClientSupport {
14.
15.     // RestTemplate
16.     public void setRestTemplate(RestTemplate restTemplate);
17.
18.     // liste des médecins
19.     @Get("/getAllMedecins")
20.     public Response<List<Medecin>> getAllMedecins();
21.
22.     // liste des clients
23.     @Get("/getAllClients")
24.     public Response<List<Client>> getAllClients();
25.
26.     // liste des créneaux d'un médecin
27.     @Get("/getAllCreneaux/{idMedecin}")
28.     public Response<List<Creneau>> getAllCreneaux(@Path long idMedecin);
29.
30.     // liste des rendez-vous d'un médecin
31.     @Get("/getRvMedecinJour/{idMedecin}/{jour}")
32.     public Response<List<Rv>> getRvMedecinJour(@Path long idMedecin, @Path String jour);
33.
34.     // Client
35.     @Get("/getClientById/{id}")
36.     public Response<Client> getClientById(@Path long id);
37.
38.     // Médecin
39.     @Get("/getMedecinById/{id}")
40.     public Response<Medecin> getMedecinById(@Path long id);
41.
42.     // Rv
43.     @Get("/getRvById/{id}")
44.     public Response<Rv> getRvById(@Path long id);
45.
46.     // Créneau
47.     @Get("/getCreneauById/{id}")
48.     public Response<Creneau> getCreneauById(@Path long id);
49.
50.     // ajouter un RV
51.     @Post("/ajouterRv")
52.     public Response<Rv> ajouterRv(@Body PostAjouterRv post);
53.
```

```

54.    // supprimer un Rv
55.    @Post("/supprimerRv")
56.    public Response<Rv> supprimerRv(@Body PostSupprimerRv post);
57.
58.    // obtenir l'agenda d'un médecin
59.    @Get(value = "/getAgendaMedecinJour/{idMedecin}/{jour}")
60.    public Response<AgendaMedecinJour> getAgendaMedecinJour(@Path long idMedecin, @Path String jour);
61.
62. }

```

- lignes 19-60 : on retrouve toutes les URL étudiées au paragraphe 3.5, page 348 ;
- ligne 16 : le composant [RestTemplate] de [Spring Android] sur lequel repose la communication client / serveur ;

3.6.3.2 L'interface [IDao]



L'interface [IDao] de la couche [DAO] est la suivante :

```

1. package rdvmedecins.android.dao.service;
2.
3. import rdvmedecins.android.dao.entities.*;
4. import rx.Observable;
5.
6. import java.util.List;
7.
8. public interface IDao {
9.     // Url du service web
10.    public void setUrlServiceWebJson(String url);
11.
12.    // utilisateur
13.    public void setUser(String user, String mdp);
14.
15.    // timeout du client
16.    public void setTimeout(int timeout);
17.
18.    // liste des clients
19.    public Observable<List<Client>> getAllClients();
20.
21.    // liste des Médecins
22.    public Observable<List<Medecin>> getAllMedecins();
23.
24.    // liste des créneaux horaires d'un médecin
25.    public Observable<List<Creneau>> getAllCreneaux(long idMedecin);
26.
27.    // liste des Rv d'un médecin, un jour donné
28.    public Observable<List<Rv>> getRvMedecinJour(long idMedecin, String jour);
29.
30.    // trouver un client identifié par son id
31.    public Observable<Client> getClientById(long id);
32.
33.    // trouver un médecin identifié par son id
34.    public Observable<Medecin> getMedecinById(long id);
35.
36.    // trouver un Rv identifié par son id
37.    public Observable<Rv> getRvById(long id);
38.
39.    // trouver un créneau horaire identifié par son id
40.    public Observable<Creneau> getCreneauById(long id);
41.
42.    // ajouter un RV
43.    public Observable<Rv> ajouterRv(String jour, long idCreneau, long idClient);
44.
45.    // supprimer un RV
46.    public Observable<Rv> supprimerRv(long idRv);
47.
48.    // metier
49.    public Observable<AgendaMedecinJour> getAgendaMedecinJour(long idMedecin, String jour);

```

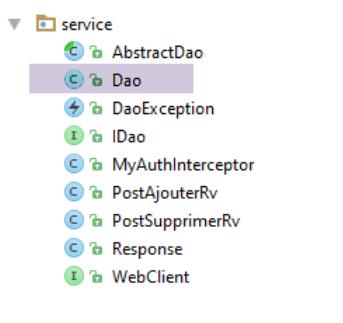
```

50.
51.    // mode debug
52.    void setDebugMode(boolean isEnabled);
53. }

```

- ligne 10 : pour fixer l'URL du service web / JSON ;
- ligne 13 : pour fixer l'utilisateur de la communication client / serveur. [user] est l'identifiant de l'utilisateur, [mdp] son mot de passe ;
- ligne 16 : pour fixer un délai d'attente maximal de la réponse du serveur ;
- lignes 18-49 : à chaque URL exposée par le service web, correspond une méthode. Elles reprennent la signature des méthodes de mêmes noms du composant AA [WebClient] ;
- ligne 52 : pour contrôler le mode *debug* de la couche [DAO] ;

3.6.3.3 La classe [Dao]



L'implémentation [DAO] de l'interface [IDao] précédente est la suivante :

```

1. package client.android.dao.service;
2.
3. import android.util.Log;
4. import client.android.dao.entities.*;
5. import org.androidannotations.annotations.AfterInject;
6. import org.androidannotations.annotations.Bean;
7. import org.androidannotations.annotations.EBean;
8. import org.androidannotations.rest.spring.annotations.RestService;
9. import org.springframework.http.client.ClientHttpRequestInterceptor;
10. import org.springframework.http.client.SimpleClientHttpRequestFactory;
11. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
12. import org.springframework.web.client.RestTemplate;
13. import rx.Observable;
14.
15. import java.util.ArrayList;
16. import java.util.List;
17.
18. @EBean(scope = EBean.Scope.Singleton)
19. public class Dao extends AbstractDao implements IDao {
20.
21.     // client du service web
22.     @RestService
23.     protected WebClient webClient;
24.     // sécurité
25.     @Bean
26.     protected MyAuthInterceptor authInterceptor;
27.     // le RestTemplate
28.     private RestTemplate restTemplate;
29.     // factory du RestTemplate
30.     private SimpleClientHttpRequestFactory factory;
31.
32.     @AfterInject
33.     public void afterInject() {
34.         ...
35.     }
36.
37.     @Override
38.     public void setUrlServiceWebJson(String url) {
39.         ...
40.     }
41.
42.     @Override
43.     public void setUser(String user, String mdp) {
44.         ...
45.     }
46.
47.     @Override

```

```

48.     public void setTimeout(int timeout) {
49.     ...
50.   }
51.
52.   @Override
53.   public void setBasicAuthentication(boolean isBasicAuthenticationNeeded) {
54.     if (isDebugEnabled) {
55.       Log.d(className, String.format("setBasicAuthentication thread=%s, isBasicAuthenticationNeeded=%s",
56.           Thread.currentThread().getName(), isBasicAuthenticationNeeded));
57.     }
58.     // intercepteur d'authentification ?
59.     if (isBasicAuthenticationNeeded) {
60.       // on ajoute l'intercepteur d'authentification
61.       List<ClientHttpRequestInterceptor> interceptors = new ArrayList<ClientHttpRequestInterceptor>();
62.       interceptors.add(authInterceptor);
63.       restTemplate.setInterceptors(interceptors);
64.     }
65.   }
66.
67.   // méthodes privées -----
68.   private void log(String message) {
69.     if (isDebugEnabled) {
70.       Log.d(className, message);
71.     }
72.   }
73.
74.   // implémentation de l'interface IDao -----
75.   @Override
76.   public Observable<Response<List<Client>>> getAllClients() {
77.     // log
78.     log("getAllClients");
79.     // résultat
80.     return getResponse(new IRequest<Response<List<Client>>>() {
81.       @Override
82.       public Response<List<Client>> getResponse() {
83.         return webClient.getAllClients();
84.       }
85.     });
86.   }
87.
88.   @Override
89.   public Observable<Response<List<Medecin>>> getAllMedecins() {
90.     // log
91.     log("getAllMedecins");
92.     // résultat
93.     return getResponse(new IRequest<Response<List<Medecin>>>() {
94.       @Override
95.       public Response<List<Medecin>> getResponse() {
96.         return webClient.getAllMedecins();
97.       }
98.     });
99.   }
100.
101.  @Override
102.  public Observable<Response<List<Creneau>>> getAllCreneaux(final long idMedecin) {
103.    // log
104.    log("getAllCreneaux");
105.    // résultat
106.    return getResponse(new IRequest<Response<List<Creneau>>>() {
107.      @Override
108.      public Response<List<Creneau>> getResponse() {
109.        return webClient.getAllCreneaux(idMedecin);
110.      }
111.    });
112.  }
113.
114.  @Override
115.  public Observable<Response<List<Rv>>> getRvMedecinJour(final long idMedecin, final String jour) {
116.    // log
117.    log("getRvMedecinJour");
118.    // résultat
119.    return getResponse(new IRequest<Response<List<Rv>>>() {
120.      @Override
121.      public Response<List<Rv>> getResponse() {
122.        return webClient.getRvMedecinJour(idMedecin, jour);
123.      }
124.    });
125.  }
126.
127.  @Override
128.  public Observable<Response<Client>> getClientById(final long id) {
129.    // log
130.    log("getClientById");
131.    // résultat
132.    return getResponse(new IRequest<Response<Client>>() {
133.      @Override

```

```

134.     public Response<Client> getResponse() {
135.         return webClient.getClientById(id);
136.     }
137. );
138. }
139.
140. @Override
141. public Observable<Response<Medecin>> getMedecinById(final long id) {
142.     // log
143.     log("getMedecinById");
144.     // résultat
145.     return getResponse(new IRequest<Response<Medecin>>() {
146.         @Override
147.         public Response<Medecin> getResponse() {
148.             return webClient.getMedecinById(id);
149.         }
150.     });
151. }
152.
153. @Override
154. public Observable<Response<Rv>> getRvById(final long id) {
155.     // log
156.     log("getRvById");
157.     // résultat
158.     return getResponse(new IRequest<Response<Rv>>() {
159.         @Override
160.         public Response<Rv> getResponse() {
161.             return webClient.getRvById(id);
162.         }
163.     });
164. }
165.
166. @Override
167. public Observable<Response<Creneau>> getCreneauById(final long id) {
168.     // log
169.     log("getCreneauById");
170.     // résultat
171.     return getResponse(new IRequest<Response<Creneau>>() {
172.         @Override
173.         public Response<Creneau> getResponse() {
174.             return webClient.getCreneauById(id);
175.         }
176.     });
177. }
178.
179. @Override
180. public Observable<Response<Rv>> ajouterRv(final String jour, final long idCreneau, final long idClient) {
181.     // log
182.     log("ajouterRv");
183.     // résultat
184.     return getResponse(new IRequest<Response<Rv>>() {
185.         @Override
186.         public Response<Rv> getResponse() {
187.             return webClient.ajouterRv(new PostAjouterRv(jour, idCreneau, idClient));
188.         }
189.     });
190. }
191.
192. @Override
193. public Observable<Response<Rv>> supprimerRv(final long idRv) {
194.     // log
195.     log("supprimerRv");
196.     // résultat
197.     return getResponse(new IRequest<Response<Rv>>() {
198.         @Override
199.         public Response<Rv> getResponse() {
200.             return webClient.supprimerRv(new PostSupprimerRv(idRv));
201.         }
202.     });
203. }
204.
205. @Override
206. public Observable<Response<AgendaMedecinJour>> getAgendaMedecinJour(final long idMedecin, final String jour) {
207.     // log
208.     log("getAgendaMedecinJour");
209.     // résultat
210.     return getResponse(new IRequest<Response<AgendaMedecinJour>>() {
211.         @Override
212.         public Response<AgendaMedecinJour> getResponse() {
213.             return webClient.getAgendaMedecinJour(idMedecin, jour);
214.         }
215.     });
216. }
217.
218. }
```

- lignes 18-72 : sont celles présentes de base dans la classe [Dao] du projet [client-android-skel] ;
- lignes 74-216 : implémentation de l'interface [IDao]. Les méthodes qui interrogent les URL exposées par le service web déléguent cette interrogation au composant AA [WebClient] (lignes 22-23) ;
- lignes 58-63 : si les échanges client / serveur sont authentifiés par une autorisation de type basique, on ajoute un intercepteur au composant [RestTemplate]. Ceci va avoir pour effet que toute requête HTTP émise par le composant [RestTemplate] va être interceptée par la classe [MyAuthInterceptor] (lignes 25-26) ;

La classe [MyAuthInterceptor] est la suivante :

```

1. package rdvmedecins.android.dao.security;
2.
3. import org.androidannotations.annotations.Bean;
4. import org.androidannotations.annotations.EBean;
5. import org.springframework.http.HttpAuthentication;
6. import org.springframework.http.HttpBasicAuthentication;
7. import org.springframework.http.HttpHeaders;
8. import org.springframework.http.HttpRequest;
9. import org.springframework.http.client.ClientHttpRequestExecution;
10. import org.springframework.http.client.ClientHttpRequestInterceptor;
11. import org.springframework.http.client.ClientHttpResponse;
12.
13. import java.io.IOException;
14.
15. @EBean(scope = EBean.Scope.Singleton)
16. public class MyAuthInterceptor implements ClientHttpRequestInterceptor {
17.
18.     // utilisateur
19.     private String user;
20.     private String mdp;
21.
22.     public ClientHttpResponse intercept(HttpRequest request, byte[] body, ClientHttpRequestExecution execution) throws
IOException {
23.         HttpHeaders headers = request.getHeaders();
24.         HttpBasicAuthentication auth = new HttpBasicAuthentication(user, mdp);
25.         headers.setAuthorization(auth);
26.         return execution.execute(request, body);
27.     }
28.
29.     public void setUser(String user, String mdp) {
30.         this.user = user;
31.         this.mdp = mdp;
32.     }
33. }
```

- ligne 15 : la classe [MyAuthInterceptor] est un composant AA de type [singleton] ;
- ligne 16 : la classe [MyAuthInterceptor] étend l'interface Spring [ClientHttpRequestInterceptor]. Cette interface a une méthode, la méthode [intercept] de la ligne 22. On étend cette interface pour intercepter toute requête HTTP du client. La méthode [intercept] reçoit trois paramètres ;
 - [HttpRequest request] : la requête HTTP interceptée,
 - [byte[] body] : son corps si elle en a un (des valeurs postées par exemple),
 - [ClientHttpRequestExecution execution] : le composant Spring qui exécute la requête ;

Nous interceptons toutes les requêtes HTTP du client Android pour lui ajouter l'entête HTTP d'authentification présenté au paragraphe 3.5, page 348.

- ligne 23 : nous récupérons les entêtes HTTP de la requête interceptée ;
- ligne 24 : nous créons l'entête HTTP d'authentification. Le mode d'authentification utilisé (codage base64 de la chaîne 'user:mdp') est fourni par la classe Spring [HttpBasicAuthentication] ;
- ligne 25 : l'entête d'authentification que nous venons de créer est ajouté aux entêtes actuels de la requête interceptée ;
- ligne 26 : on poursuit l'exécution de la requête interceptée. Si nous résumons, la requête interceptée a été enrichie de l'entête d'authentification ;

Les implémentations des méthodes de l'interface [IDao] sont toutes faites sur le même modèle. Prenons l'exemple de la méthode [getAgendaMedecinJour] :

```

1.     @Override
2.     public Observable<Response<AgendaMedecinJour>> getAgendaMedecinJour(final long idMedecin, final String jour) {
3.         // log
4.         log("getAgendaMedecinJour");
5.         // résultat
6.         return getResponse(new IRequest<Response<AgendaMedecinJour>>() {
7.             @Override
8.             public Response<AgendaMedecinJour> getResponse() {
9.                 return webClient.getAgendaMedecinJour(idMedecin, jour);
10.            }
```

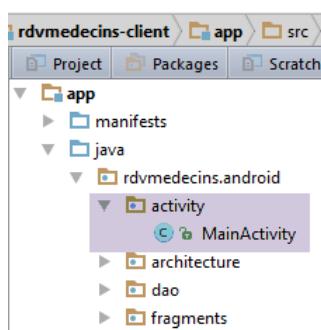
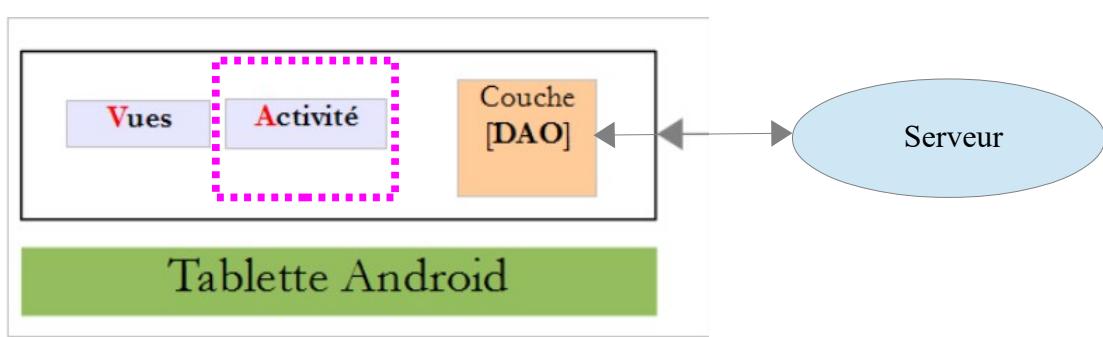
```
11.     });
12. }
```

- ligne 2 : la méthode attend deux paramètres :
 - [idMedecin] : l'identifiant du médecin dont on veut l'agenda ;
 - [jour] : le jour pour lequel on veut l'agenda ;
- ligne 6 : on appelle la méthode [getResponse] de la classe parent [AbstractDao]. Cette méthode attend un paramètre de type [IRequest<T>] où T est le type rendu par la méthode [getAgendaMedecinJour] ligne 2, ici [Response<AgendaMedecinJour>]. L'interface [IRequest] n'a qu'une méthode : [getResponse] (ligne 8) ;
- lignes 8-10 : implémentation de la méthode [IRequest.getResponse]. Cette méthode doit rendre le résultat attendu par la méthode [getAgendaMedecinJour] ligne 2 ;
- ligne 9 : la réponse est en fait rendue par la méthode [webClient.getAgendaMedecinJour] :

```
a) // obtenir l'agenda d'un médecin
b) @Get(value = "/getAgendaMedecinJour/{idMedecin}/{jour}")
c) Response<AgendaMedecinJour> getAgendaMedecinJour(@Path long idMedecin, @Path String jour);
```

Les paramètres utilisés ligne 9 sont ceux passés à la méthode [getAgendaMedecinJour] ligne 2. Pour cette raison, ces paramètres doivent avoir l'attribut *final* ;

3.6.4 L'activité [MainActivity]



La classe [MainActivity] est la suivante :

```
1. package client.android.activity;
2.
3. import android.util.Log;
4. import client.android.architecture.core.AbstractActivity;
5. import client.android.architecture.core.AbstractFragment;
6. import client.android.architecture.custom.IMainActivity;
7. import client.android.dao.entities.*;
8. import client.android.dao.service.Dao;
9. import client.android.dao.service.IDao;
10. import client.android.dao.service.Response;
11. import client.android.fragments.behavior.AccueilFragment_;
12. import client.android.fragments.behavior.AgendaFragment_;
13. import client.android.fragments.behavior.AjoutRvFragment_;
14. import client.android.fragments.behavior.ConfigFragment_;
15. import org.androidannotations.annotations.Bean;
16. import org.androidannotations.annotations.EActivity;
```

```

17. import rx.Observable;
18.
19. import java.util.List;
20.
21. @EActivity
22. public class MainActivity extends AbstractActivity {
23.
24.     // couche [DAO]
25.     @Bean(Dao.class)
26.     protected IDao dao;
27.
28.     // classe parent -----
29.     @Override
30.     protected void onCreateActivity() {
31.         // log
32.         if (IS_DEBUG_ENABLED) {
33.             Log.d(className, "onCreateActivity");
34.         }
35.     }
36.
37.     @Override
38.     protected IDao getDao() {
39.         return dao;
40.     }
41.
42.     @Override
43.     protected AbstractFragment[] getFragments() {
44.         AbstractFragment[] fragments= new AbstractFragment[]{new ConfigFragment_(), new AccueilFragment_(), new
        AgendaFragment_(), new AjoutRvFragment_()};
45.         return fragments;
46.     }
47.
48.     @Override
49.     protected CharSequence getFragmentTitle(int position) {
50.         return null;
51.     }
52.
53.     @Override
54.     protected void navigateOnTabSelected(int position) {
55.     }
56.
57.     @Override
58.     protected int getFirstView() {
59.         return IMainActivity.VUE_CONFIG;
60.     }
61.
62.
63.     // interface IDao -----
64. ...
65.
66.     @Override
67.     public Observable<Response<List<Client>>> getAllClients() {
68.         return dao.getAllClients();
69.     }
70.
71.     @Override
72.     public Observable<Response<List<Medecin>>> getAllMedecins() {
73.         return dao.getAllMedecins();
74.     }
75.
76.     @Override
77.     public Observable<Response<List<Creneau>>> getAllCreneaux(long idMedecin) {
78.         return dao.getAllCreneaux(idMedecin);
79.     }
80.
81.     @Override
82.     public Observable<Response<List<Rv>>> getRvMedecinJour(long idMedecin, String jour) {
83.         return dao.getRvMedecinJour(idMedecin, jour);
84.     }
85.
86.     @Override
87.     public Observable<Response<Client>> getClientById(long id) {
88.         return dao.getClientById(id);
89.     }
90.
91.     @Override
92.     public Observable<Response<Medecin>> getMedecinById(long id) {
93.         return dao.getMedecinById(id);
94.     }
95.
96.     @Override
97.     public Observable<Response<Rv>> getRvById(long id) {
98.         return dao.getRvById(id);
99.     }
100.    @Override
101.    public Observable<Response<Creneau>> getCreneauById(long id) {

```

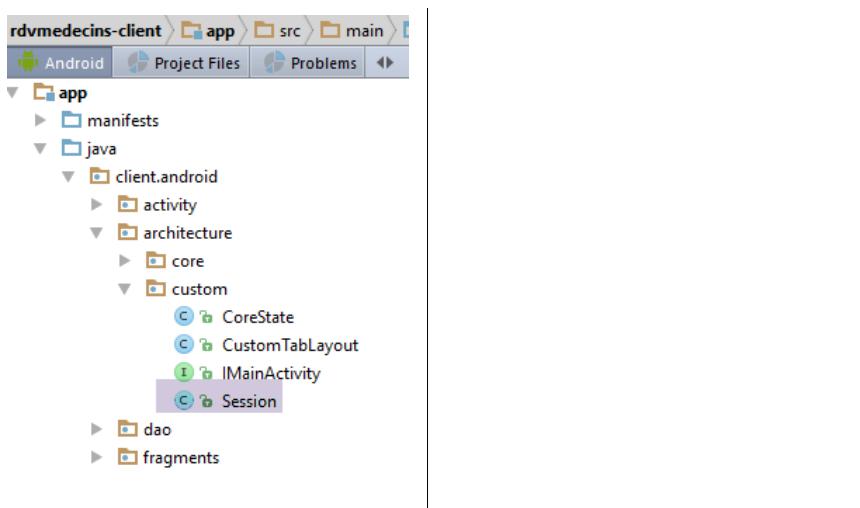
```

103.     return dao.getCreneauById(id);
104. }
105.
106. @Override
107. public Observable<Response<Rv>> ajouterRv(String jour, long idCreneau, long idClient) {
108.     return dao.ajouterRv(jour, idCreneau, idClient);
109. }
110.
111. @Override
112. public Observable<Response<Rv>> supprimerRv(long idRv) {
113.     return dao.supprimerRv(idRv);
114. }
115.
116. @Override
117. public Observable<Response<AgendaMedecinJour>> getAgendaMedecinJour(long idMedecin, String jour) {
118.     return dao.getAgendaMedecinJour(idMedecin, jour);
119. }
120. }

```

- lignes 21-66 : ces lignes sont fournies de base dans le modèle [client-android-skel] ;
- lignes 66-119 : implémentation de l'interface [IDao]. Toutes les méthodes déléguent le travail à la couche [DAO] de la ligne 26 ;
- lignes 42-46 : la méthode [getFragments] rend le tableau des quatre fragments de l'application ;
- lignes 58-61 : la vue de configuration est la 1ère vue à afficher lorsque l'application démarre ;

3.6.5 La session



La classe [Session] sert à mémoriser les informations qui doivent être transmises entre fragments. Elle est la suivante :

```

1. package rdvmedecins.android.architecture;
2.
3. import rdvmedecins.android.dao.entities.AgendaMedecinJour;
4. import rdvmedecins.android.dao.entities.Client;
5. import rdvmedecins.android.dao.entities.Medecin;
6. import org.androidannotations.annotations.EBean;
7.
8. import java.util.List;
9.
10. @EBean(scope = EBean.Scope.Singleton)
11. public class Session {
12.     // liste des médecins
13.     private List<Medecin> medecins;
14.     // liste des clients
15.     private List<Client> clients;
16.     // agenda
17.     private AgendaMedecinJour agenda;
18.     // position de l'élément cliqué dans l'agenda
19.     private int position;
20.     // jour du Rv en notation anglaise "yyyy-MM-dd"
21.     private String dayRv;
22.     // jour du Rv en notation française "dd-MM-yyyy"
23.     private String jourRv;
24.
25.
26.     // getters et setters
27. ...

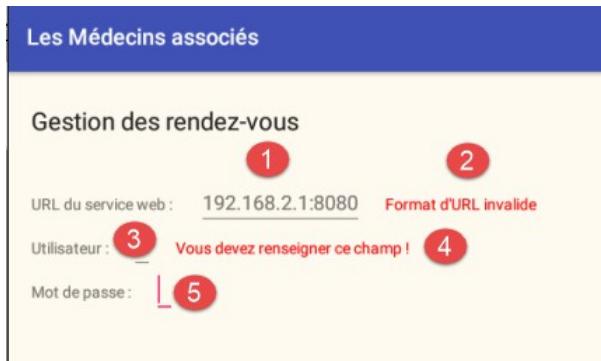
```

- ligne 10 : la classe [Session] est un composant AA instancié en un unique exemplaire ;
- lignes 12-15 : on supposera dans cette étude de cas, que les listes de médecins et de clients ne changent pas. On les demandera au démarrage de l'application et on les stockera dans la session pour que les fragments puissent les utiliser ;
- lignes 20-23 : le jour souhaité pour un rendez-vous. Il est manipulé sous deux formes, en notation française (ligne 23) au sein du client Android, en notation anglaise (ligne 21) pour les échanges avec le serveur ;
- ligne 19 : la position de l'élément cliqué (ajouter / supprimer) sur l'agenda ;

3.6.6 Gestion de la vue de configuration

3.6.6.1 La vue

La vue de configuration est la vue affichée au démarrage de l'application :

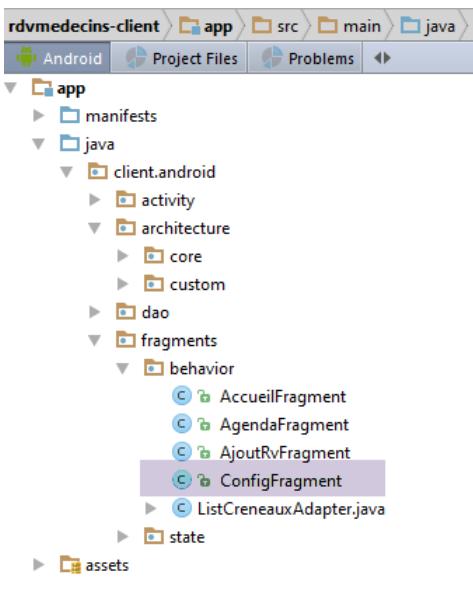


Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	EditText	edtUrlServiceRest
3	EditText	edtUtilisateur
5	EditText	edtMdp
2	TextView	txtErrorUrlServiceRest
3	TextView	txtErrorUtilisateur

3.6.6.2 Le fragment

La vue de configuration est gérée par le fragment [ConfigFragment] suivant :

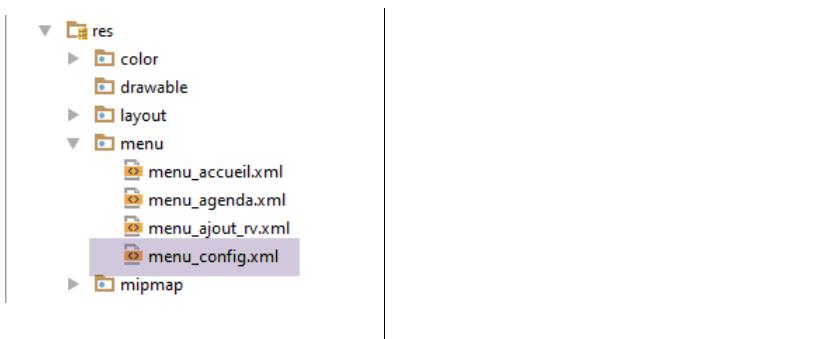


```

1. package client.android.fragments.behavior;
2.
3. import android.util.Log;
4. import android.view.View;
5. import android.widget.Button;
6. import android.widget.EditText;
7. import android.widget.TextView;
8. import client.android.R;
9. import client.android.architecture.core.AbstractFragment;
10. import client.android.architecture.core.ISession;
11. import client.android.architecture.core.MenuItemState;
12. import client.android.architecture.custom.CoreState;
13. import client.android.architecture.custom.IMainActivity;
14. import client.android.dao.entities.Client;
15. import client.android.dao.entities.Medecin;
16. import client.android.dao.service.Response;
17. import client.android.fragments.state.ConfigFragmentState;
18. import org.androidannotations.annotations.*;
19. import rx.functions.Action1;
20.
21. import java.net.URI;
22. import java.util.List;
23.
24. @EFragment(R.layout.config)
25. @OptionsMenu(R.menu.menu_config)
26. public class ConfigFragment extends AbstractFragment {
27.
28.     // les éléments de l'interface visuelle
29.     @ViewById(R.id.edt_urlServiceRest)
30.     protected EditText edtUrlServiceRest;
31.     @ViewById(R.id.txt_errorUrlServiceRest)
32.     protected TextView txtErrorUrlServiceRest;
33.     @ViewById(R.id.txt_errorUtilisateur)
34.     protected TextView txtErrorUtilisateur;
35.     @ViewById(R.id.edt_utilisateur)
36.     protected EditText edtUtilisateur;
37.     @ViewById(R.id.edt_mdp)
38.     protected EditText edtMdp;
39.
40.     // les saisies
41.     private String urlServiceRest;
42.     private String utilisateur;
43.     private String mdp;
44.
45.     // validation de la page
46.     @OptionsItemSelected(R.id.actionValider)
47.     protected void doValider() {
48.         ...
49.     }
50. ..
51.     // implémentation méthodes classe parent -----
52.     ...
53. }

```

- ligne 25 : le fragment est associé au menu [menu_config] suivant :



```
a) <menu xmlns:android="http://schemas.android.com/apk/res/android"
b)   xmlns:app="http://schemas.android.com/apk/res-auto"
c)   xmlns:tools="http://schemas.android.com/tools"
d)   tools:context=".activity.MainActivity1"
e)   <item
f)     android:id="@+id/menuActions"
g)     app:showAsAction="ifRoom"
h)     android:title="@string/menuActions"
i)   <menu>
j)     <item
k)       android:id="@+id/actionValider"
l)       android:title="@string/actionValider"/>
m)   <item
n)     android:id="@+id/actionAnnuler"
o)     android:title="@string/actionAnnuler"/>
p)   </menu>
q) </item>
r) </menu>
s) </menu>
```

- lignes 28-38 : les éléments de l'interface visuelle ;
- lignes 41-43 : les trois saisies du formulaire ;

Le clic sur l'option de menu [Valider] est géré par la méthode [doValider] :

```
1. // validation de la page
2. @OptionsItem(R.id.actionValider)
3. protected void doValider() {
4.   // on cache les éventuels msg d'erreur précédents
5.   txtErrorUrlServiceRest.setVisibility(View.INVISIBLE);
6.   txtErrorUtilisateur.setVisibility(View.INVISIBLE);
7.   // on teste la validité des saisies
8.   if (!isPageValid()) {
9.     return;
10.   }
11.   // on renseigne l'URL du service web
12.   mainActivity.setUrlServiceWebJson(urlServiceRest);
13.   // on renseigne l'utilisateur
14.   mainActivity.setUser(utilisateur, mdp);
15.   // début de l'attente
16.   beginWaiting(2);
17.   // médecins
18.   executeInBackground(mainActivity.getAllMedecins(), new Action1<Response<List<Medecin>>>() {
19.     @Override
20.     public void call(Response<List<Medecin>> responseMedecins) {
21.       // on consomme la réponse
22.       consumeMedecins(responseMedecins);
23.     }
24.   });
25.   // clients
26.   executeInBackground(mainActivity.getAllClients(), new Action1<Response<List<Client>>>() {
27.     @Override
28.     public void call(Response<List<Client>> responseClients) {
29.       // on consomme la réponse
30.       consumeClients(responseClients);
31.     }
32.   });
33. }
34.
35.
36. private void consumeMedecins(Response<List<Medecin>> responseMedecins) {
37.   // log
38.   if (isDebugEnabled) {
39.     Log.d(className, "consume médecins");
```

```

40.    }
41.    // erreur ?
42.    if (responseMedecins.getStatus() != 0) {
43.        // message
44.        showAlert(responseMedecins.getMessages());
45.        // annulation
46.        doAnnuler();
47.        // retour à l'UI
48.        return;
49.    }
50.    // on mémorise les médecins dans la session
51.    session.setMedecins(responseMedecins.getBody());
52. }
53.
54. private void consumeClients(Response<List<Client>> responseClients) {
55.     // log
56.     if (isDebugEnabled) {
57.         Log.d(className, "consume clients");
58.     }
59.     // erreur ?
60.     if (responseClients.getStatus() != 0) {
61.         // message
62.         showAlert(responseClients.getMessages());
63.         // annulation
64.         doAnnuler();
65.         // retour à l'UI
66.         return;
67.     }
68.     // on mémorise les clients dans la session
69.     session.setClients(responseClients.getBody());
70. }

```

- lignes 8-10 : la validité des trois saisies du formulaire est testée. Si le formulaire est invalide, on ne va pas plus loin ;
- lignes 11-14 : les saisies nécessaires à la couche [DAO] sont passées à l'activité ;
- ligne 16 : on indique à la classe parent qu'on va lancer deux tâches asynchrones et on prépare l'attente ;
- lignes 17-24 : la liste des médecins est demandée ;
- ligne 18 : la méthode [executeInBackground] attend deux paramètres :
 - ligne 18 : le processus à exécuter et observer est fourni par la méthode [mainActivity.getAllMedecins()] ;
 - lignes 18-24 : le second paramètre est une instance de type [Action1<T>] où T est le type rendu par le processus observé, ici [Response<List<Medecin>>]
- ligne 22 : lorsqu'on reçoit la réponse, on la passe à la méthode [consumeMedecins] de la ligne 36 ;
- lignes 25-33 : après avoir lancé une 1ère tâche asynchrone, on en lance une seconde pour demander la liste des clients. On va donc avoir deux tâches s'exécutant en parallèle ;
- lignes 36-52 : on a reçu la réponse de la tâche des médecins. On l'exploite ;
- lignes 42-49 : on regarde d'abord si le serveur a signalé une erreur dans le champ [status] de la réponse ;
- ligne 44 : si erreur il y a, on affiche les messages que le serveur a mis dans le champ [messages] de la réponse ;
- ligne 46 : on annule toutes les tâches ;
- ligne 48 : on revient à l'Ui ;
- ligne 51 : s'il n'y a pas eu d'erreur, la liste des médecins est mise en session ;

La validité des saisies (ligne 8) est vérifiée avec la méthode suivante :

```

1. private boolean isPageValid() {
2.     // on vérifie la validité des données saisies
3.     boolean erreur;
4.     URI service;
5.     // validité de l'URL du service REST
6.     urlServiceRest = String.format("http://%", edtUrlServiceRest.getText().toString().trim());
7.     try {
8.         service = new URI(urlServiceRest);
9.         erreur = service.getHost() == null || service.getPort() == -1;
10.    } catch (Exception ex) {
11.        // on note l'erreur
12.        erreur = true;
13.    }
14.    if (erreur) {
15.        // affichage erreur
16.        txtErrorUrlServiceRest.setVisibility(View.VISIBLE);
17.    }
18.    // utilisateur
19.    utilisateur = edtUtilisateur.getText().toString().trim();
20.    if (utilisateur.length() == 0) {
21.        // on affiche l'erreur
22.        txtErrorUtilisateur.setVisibility(View.VISIBLE);
23.        // on note l'erreur
24.        erreur = true;
25.    }
26.    // mot de passe
27.    mdp = edtMdp.getText().toString().trim();

```

```

28.     // retour
29.     return !erreur;
30. }

```

La méthode [beginWaiting] (ligne 16) est la suivante :

```

1.     // début de l'attente
2.     protected void beginWaiting(int numberOfRunningTasks) {
3.         // on prépare le lancement des tâches
4.         beginRunningTasks(numberOfRunningTasks);
5.         // état des boutons et menus
6.         setAllMenuOptionsStates(false);
7.         setMenuOptionsStates(new MenuItemState[] {new MenuItemState(R.id.menuActions, true),new
    MenuItemState(R.id.actionAnnuler, true)});
8.
9.     }

```

- ligne 4 : on indique à la tâche parent qu'on va lancer [numberOfRunningTasks] tâches ;
- ligne 6 : on rend invisibles toutes les options du menu ;
- ligne 7 : pour ensuite rendre visible l'option [Actions/Annuler] ;

Le clic sur l'option de menu [Annuler] est géré par la méthode [doAnnuler] :

```

1.     @OptionsItem(R.id.actionAnnuler)
2.     protected void doAnnuler() {
3.         if (isDebugEnabled) {
4.             Log.d(className, "Annulation demandée");
5.         }
6.         // on annule les tâches asynchrones
7.         cancelRunningTasks();
8.     }

```

- ligne 8 : on demande à la classe parent d'annuler les tâches asynchrones ;

3.6.6.3 Gestion du cycle de vie du fragment

Le fragment a l'état [ConfigFragmentState] suivant :

```

1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. public class ConfigFragmentState extends CoreState {
6.
7.     // la visibilité des deux messages d'erreur
8.     private boolean txtErrorUrlServiceRestVisible;
9.     private boolean txtErrorUtilisateurVisible;
10.
11.    // getters et setters
12. ...
13. }

```

- lorsque la classe parent le lui demandera, le fragment sauvera la visibilité de ses deux messages d'erreur ;

Le cycle de vie du fragment est implémenté de la façon suivante :

```

1. // implémentation méthodes classe parent -----
2. @Override
3. public CoreState saveFragment() {
4.     // sauvegarde état fragment
5.     ConfigFragmentState state = new ConfigFragmentState();
6.     state.setTxtErrorUrlServiceRestVisible(txtErrorUrlServiceRest.getVisibility() == View.VISIBLE);
7.     state.setTxtErrorUtilisateurVisible(txtErrorUtilisateur.getVisibility() == View.VISIBLE);
8.     return state;
9. }
10.
11. @Override
12. protected int getNumView() {
13.     return IMainActivity.VUE_CONFIG;
14. }
15.
16. @Override
17. protected void initFragment(CoreState previousState) {
18.
19. }
20.
21. @Override
22. protected void initView(CoreState previousState) {

```

```

23.     if (previousState == null) {
24.         // 1ère visite
25.         // on cache les messages d'erreur
26.         txtErrorUtilisateur.setVisibility(View.INVISIBLE);
27.         txtErrorUrlServiceRest.setVisibility(View.INVISIBLE);
28.         // menu
29.         initMenu();
30.     }
31. }
32.
33. @Override
34. protected void updateOnSubmit(CoreState previousState) {
35. }
36.
37. @Override
38. protected void updateOnRestore(CoreState previousState) {
39.     // restauration de la visibilité des msg d'erreur
40.     ConfigFragmentState state = (ConfigFragmentState) previousState;
41.     // pas la 1ère visite - on restitue les msg d'erreur
42.     txtErrorUtilisateur.setVisibility(state.isTxtErrorUtilisateurVisible() ? View.VISIBLE : View.INVISIBLE);
43.     txtErrorUrlServiceRest.setVisibility(state.isTxtErrorUrlServiceRestVisible() ? View.VISIBLE : View.INVISIBLE);
44. }
45.
46.
47. @Override
48. protected void notifyEndOfUpdates() {
49. }
50.
51. @Override
52. protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
53.     // menu
54.     initMenu();
55.     // vue suivante ?
56.     if (!runningTasksHaveBeenCalled) {
57.         mainActivity.navigateToView(IMainActivity.VUE_ACCUEIL, ISession.Action.SUBMIT);
58.     }
59. }
60.
61. // méthodes privées -----
62. private void initMenu(){
63.     // état menu
64.     setAllMenuOptionsStates(true);
65.     setMenuOptionsStates(new MenuItemState[]{new MenuItemState(R.id.actionAnnuler, false)});
66. }

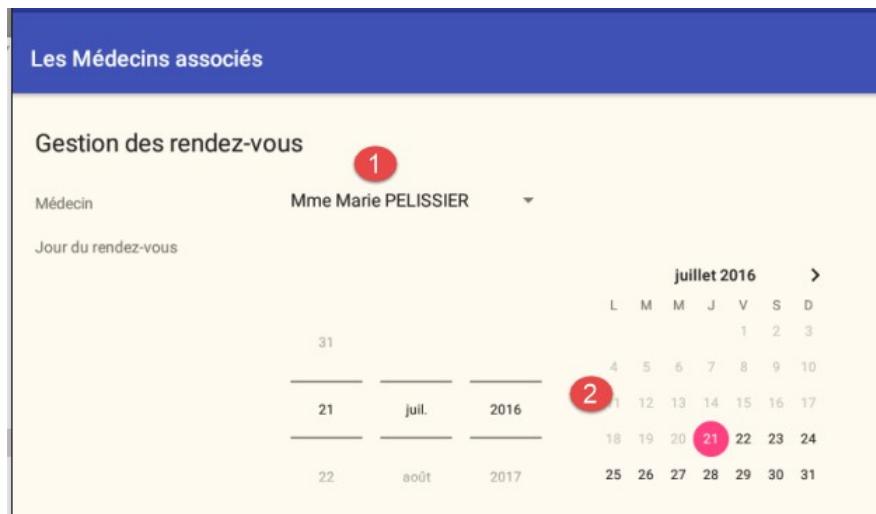
```

- lignes 2-9 : lorsque sa classe parent le lui demande, le fragment sauve l'état de ses deux messages d'erreur ;
- lignes 11-14 : le n° du fragment est [IMainActivity.VUE_CONFIG] ;
- lignes 16-19 : exécutées lorsque le fragment est généré la 1ère fois (previousState==null) ou régénéré les fois suivantes (previousState !=null). Ici, il n'y a rien à faire ;
- lignes 21-31 : exécutées lorsque la vue associée au fragment est construite la 1ère fois (previousState==null) ou reconstruite les fois suivantes (previousState !=null) ;
 - lignes 24-29 : pour la 1ère visite, on cache les messages d'erreur et on affiche le menu sans l'action [Annuler] (lignes 62-66) ;
- lignes 39-41 : exécutées lorsqu'on arrive au fragment par une opération [SUBMIT]. Ca n'arrive jamais ici ;
- lignes 43-45 : exécutées lorsqu'on arrive au fragment par une opération [NAVIGATION] ou [RESTORE] ;
- lignes 37-44 : on restitue l'état des messages d'erreur à partir de l'état précédent ;
- lignes 47-49 : exécutées lorsque toutes les mises à jour précédentes ont été faites. Il n'y a rien de plus à faire ;
- lignes 51-59 : exécutées lorsque toutes les tâches asynchrones sont terminées ;
 - lignes 53-54 : on remet le menu dans son état par défaut ;
 - lignes 56-58 : si les tâches se sont terminées normalement, alors on passe à la vue suivante, sinon on reste sur la même vue ;

3.6.7 Gestion de la vue d'accueil

3.6.7.1 La vue

La vue d'accueil est la suivante :

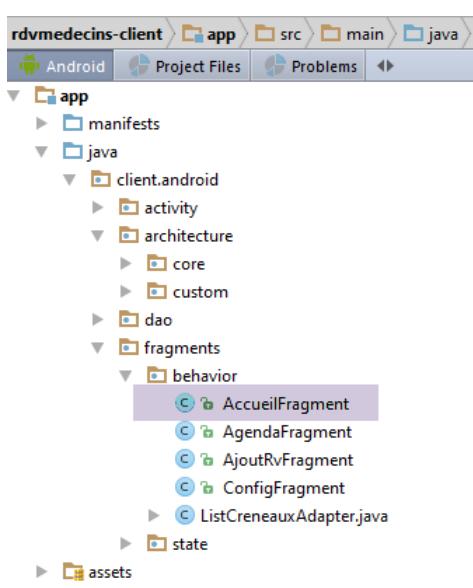


Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	Spinner	spinnerMedecins
2	DatePicker	edtJourRv

3.6.7.2 Le fragment

La vue d'accueil est gérée par le fragment [AccueilFragment] suivant :



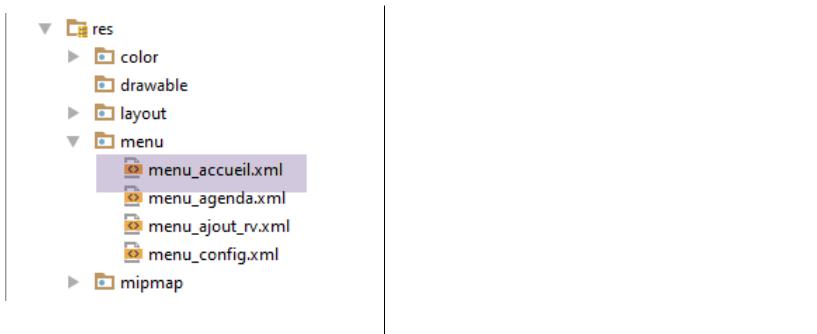
```

1. package client.android.fragments.behavior;
2.
3. import android.util.Log;
4. import android.view.View;
5. import android.widget.ArrayAdapter;
6. import android.widget.Button;
7. import android.widget.DatePicker;
8. import android.widget.Spinner;
9. import client.android.R;
10. import client.android.architecture.core.AbstractFragment;
11. import client.android.architecture.core.ISession;
12. import client.android.architecture.core.MenuItemState;
13. import client.android.architecture.custom.CoreState;
14. import client.android.architecture.custom.IMainActivity;
```

```

15. import client.android.dao.entities.AgendaMedecinJour;
16. import client.android.dao.entities.Medecin;
17. import client.android.dao.service.Response;
18. import client.android.fragments.state.AccueilFragmentState;
19. import org.androidannotations.annotations.*;
20. import rx.functions.Action1;
21.
22. import java.util.Calendar;
23. import java.util.List;
24. import java.util.Locale;
25.
26. @EFragment(R.layout.accueil)
27. @OptionsMenu(R.menu.menu_accueil)
28. public class AccueilFragment extends AbstractFragment {
29.
30.     // les éléments de l'interface visuelle
31.     @ViewById(R.id.spinnerMedecins)
32.     protected Spinner spinnerMedecins;
33.     @ViewById(R.id.edt_JourRv)
34.     protected DatePicker edtJourRv;
35.
36.     // données locales
37.     private List<Medecin> medecins;
38.     private Calendar calendrier;
39.     private String[] spinnerMedecinsDataSource;
40.
41.     // validation de la page
42.     @OptionsItem(R.id.actionValider)
43.     protected void doValider() {
44.         ...
45.     }
46. ...
47.
48.     // implémentation méthodes classe parent -----
49. ...
50. }
```

- ligne 26 : le fragment est associé au menu [menu_accueil] suivant :



```

a) <menu xmlns:android="http://schemas.android.com/apk/res/android"
b)   xmlns:app="http://schemas.android.com/apk/res-auto"
c)   xmlns:tools="http://schemas.android.com/tools"
d)   tools:context=".activity.MainActivity1">
e)   <item
f)     android:id="@+id/menuActions"
g)     app:showAsAction="ifRoom"
h)     android:title="@string/menuActions">
i)     <menu>
j)       <item
k)         android:id="@+id/actionValider"
l)         android:title="@string/actionValider"/>
m)       <item
n)         android:id="@+id/actionAnnuler"
o)         android:title="@string/actionAnnuler"/>
p)     </menu>
q)   </item>
r)   <item
s)     android:id="@+id/menuNavigation"
t)     app:showAsAction="ifRoom"
u)     android:title="@string/menuNavigation">
v)     <menu>
w)       <item
x)         android:id="@+id/navigationToConfig"
y)         android:title="@string/navigationToConfig"/>
z)       </menu>
aa)   </item>
ab) </menu>
```

- lignes 31-34 : les éléments de l'interface visuelle ;
- ligne 37 : la liste des médecins ;
- ligne 38 : un calendrier ;
- ligne 39 : la source de données du spinner des médecins ;

Le clic sur le lien [Valider] est géré par la méthode [doValider] suivante :

```

1. // validation de la page
2. @OptionsItem(R.id.actionValider)
3. protected void doValider() {
4.     // on note l'id du médecin sélectionné
5.     Long idMedecin = medecins.get(spinnerMedecins.getSelectedItemPosition()).getId();
6.     // on mémorise le jour dans la session
7.     String jourRv = String.format(new Locale("Fr-fr"), "%02d-%02d-%04d", edtJourRv.getDayOfMonth(), edtJourRv.getMonth()
+ 1, edtJourRv.getYear());
8.     session.setJourRv(jourRv);
9.     // on passe au format de date yyyy-MM-dd
10.    String dayRv = String.format(new Locale("Fr-fr"), "%04d-%02d-%02d", edtJourRv.getYear(), edtJourRv.getMonth() + 1,
edtJourRv.getDayOfMonth());
11.    session.setDayRv(dayRv);
12.    // début de l'attente
13.    beginWaiting(1);
14.    // on demande l'agenda du médecin
15.    executeInBackground(mainActivity.getAgendaMedecinJour(idMedecin, dayRv), new Action1<Response<AgendaMedecinJour>>() {
16.
17.        @Override
18.        public void call(Response<AgendaMedecinJour> responseAgendaMedecinJour) {
19.            // on consomme la réponse
20.            consumeAgenda(responseAgendaMedecinJour);
21.        }
22.    });
23. }
24.
25. private void consumeAgenda(Response<AgendaMedecinJour> responseAgendaMedecinJour) {
26.     // erreur ?
27.     if (responseAgendaMedecinJour.getStatus() != 0) {
28.         // message
29.         showAlert(responseAgendaMedecinJour.getMessages());
30.         // annulation
31.         doAnnuler();
32.         // retour à l'UI
33.         return;
34.     }
35.     // on met l'agenda dans la session
36.     session.setAgenda(responseAgendaMedecinJour.getBody());
37. }
```

- ligne 5 : on récupère l'identifiant du médecin sélectionné ;
- lignes 7-8 : on met en session, au format français, la date choisie ;
- lignes 10-11 : on met en session, au format anglais, la date choisie ;
- ligne 13 : on indique à la classe parent qu'on va lancer une tâche asynchrone et on prépare l'attente ;
- lignes 15-22 : l'agenda du médecin est demandé ;
 - ligne 15 : la méthode [executeInBackground] attend deux paramètres :
 - ligne 15 : le processus à exécuter et observer est fourni par la méthode [mainActivity.getAgendaMedecinJour(idMedecin, dayRv)] ;
 - lignes 15-22 : le second paramètre est une instance de type [Action1<T>] où T est le type rendu par le processus observé, ici [Response<AgendaMedecinJour>]
 - ligne 20 : lorsqu'on reçoit la réponse, on la passe à la méthode [consumeAgenda] de la ligne 25 ;
- lignes 25-37 : on a reçu l'agenda du médecin. On l'exploite ;
- lignes 27-34 : on regarde d'abord si le serveur a signalé une erreur dans le champ [status] de la réponse ;
- ligne 29 : si erreur il y a, on affiche les messages que le serveur a mis dans le champ [messages] de la réponse ;
- ligne 31 : on annule toutes les tâches ;
- ligne 33 : on revient à l'Ui ;
- ligne 36 : s'il n'y a pas eu d'erreurs, l'agenda est mis en session ;

La méthode [beginWaiting] (ligne 13) est la suivante :

```

1. // début de l'attente
2. protected void beginWaiting(int numberOfRunningTasks) {
3.     // on prépare le lancement des tâches
4.     beginRunningTasks(numberOfRunningTasks);
5.     // état des boutons et menus
6.     setAllMenuOptionsStates(false);
7.     setMenuOptionsStates(new MenuItemState[]{new MenuItemState(R.id.menuActions, true),new
MenuItemState(R.id.actionAnnuler, true)});
```

```
8.  
9. }
```

- ligne 4 : on indique à la tâche parent qu'on va lancer [numberOfRunningTasks] tâches ;
- ligne 6 : on rend invisibles toutes les options du menu ;
- ligne 7 : pour ensuite rendre visible l'option [Actions/Annuler] ;

Le clic sur l'option de menu [Annuler] est géré par la méthode [doAnnuler] :

```
9.     @OptionsItem(R.id.actionAnnuler)  
10.    protected void doAnnuler() {  
11.        if (isDebugEnabled) {  
12.            Log.d(className, "Annulation demandée");  
13.        }  
14.        // on annule les tâches asynchrones  
15.        cancelRunningTasks();  
16.    }
```

- ligne 8 : on demande à la classe parent d'annuler les tâches asynchrones ;

Le clic sur l'option de menu [Retour à la configuration] est géré de la façon suivante :

```
1.     @OptionsItem(R.id.navigationToConfig)  
2.     protected void navigationToConfig() {  
3.         // on navigue vers la vue de configuration  
4.         mainActivity.navigateToView(IMainActivity.VUE_CONFIG, ISession.Action.NAVIGATION);  
5.     }
```

- ligne 4 : on navigue vers la vue de configuration avec l'action [NAVIGATION]. Cela signifie qu'on veut retrouver la vue de configuration dans l'état où on l'a laissée ;

3.6.7.3 Gestion du cycle de vie du fragment

Le fragment a l'état [AccueilFragmentState] suivant :

```
1. package client.android.fragments.state;  
2.  
3. import android.widget.ArrayAdapter;  
4. import client.android.architecture.custom.CoreState;  
5. import client.android.dao.entities.CreneauMedecinJour;  
6.  
7. public class AccueilFragmentState extends CoreState {  
8.  
9.     // état fragment [Accueil]  
10.    // position du médecin sélectionné  
11.    private int selectedMedecinPosition;  
12.    // date sélectionnée  
13.    private int year;  
14.    private int month;  
15.    private int dayOfMonth;  
16.    // source de données du spinner des médecins  
17.    private String[] spinnerMedecinsDataSource;  
18.  
19.    // constructeurs  
20.    public AccueilFragmentState() {  
21.  
22.    }  
23.  
24.    // getters et setters  
25.    ...  
26. }
```

- ligne 11 : permet de restituer l'élément sélectionné dans la liste des médecins ;
- lignes 13-15 : permet de restituer la date choisie dans le calendrier ;
- lignes 17 : permet de restituer la source de données de la liste des médecins ;

Le cycle de vie du fragment est implémenté de la façon suivante :

```
1. // implémentation méthodes classe parent -----  
2. @Override  
3. public CoreState saveFragment() {  
4.     // on sauvegarde la vue  
5.     AccueilFragmentState state = new AccueilFragmentState();  
6.     state.setSelectedMedecinPosition(spinnerMedecins.getSelectedItemPosition());  
7.     state.setDayOfMonth(edtJourRv.getDayOfMonth());  
8.     state.setMonth(edtJourRv.getMonth());  
9.     state.setYear(edtJourRv.getYear());
```

```

10.    state.setSpinnerMedecinsDataSource(spinnerMedecinsDataSource);
11.    return state;
12. }
13.
14. @Override
15. protected int getNumView() {
16.     return IMainActivity.VUE_ACCUEIL;
17. }
18.
19. @Override
20. protected void initFragment(CoreState previousState) {
21.     // on récupère les médecins en session
22.     medecins = session.getMedecins();
23.     // 1ère visite ?
24.     if (previousState == null) {
25.         // on construit le tableau affiché par le spinner
26.         spinnerMedecinsDataSource = new String[medecins.size()];
27.         int i = 0;
28.         for (Medecin medecin : medecins) {
29.             spinnerMedecinsDataSource[i] = String.format("%s %s %s", medecin.getTitre(), medecin.getPrenom(),
30. medecin.getNom());
31.             i++;
32.         }
33.     } else {
34.         // pas 1ère visite
35.         AccueilFragmentState state = (AccueilFragmentState) previousState;
36.         spinnerMedecinsDataSource = state.getSpinnerMedecinsDataSource();
37.     }
38.     // le calendrier
39.     calendrier = Calendar.getInstance();
40. }
41.
42. @Override
43. protected void initView(CoreState previousState) {
44.     // on associe le spinner des médecins à sa source de données
45.     ArrayAdapter<String> dataAdapterMedecins = new ArrayAdapter<>(activity, android.R.layout.simple_spinner_item,
46. spinnerMedecinsDataSource);
47.     dataAdapterMedecins.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
48.     spinnerMedecins.setAdapter(dataAdapterMedecins);
49.     // date minimale à aujourd'hui
50.     edtJourRv.setMinDate(calendrier.getTimeInMillis());
51.     // 1ère visite ?
52.     if (previousState == null) {
53.         // menu
54.         initMenu();
55.     }
56.
57.     @Override
58.     protected void updateOnSubmit(CoreState previousState) {
59.         // menu
60.         initMenu();
61.     }
62.
63.     @Override
64.     protected void updateOnRestore(CoreState previousState) {
65.         // on restaure l'état en session
66.         AccueilFragmentState state = (AccueilFragmentState) previousState;
67.         // spinner des médecins
68.         spinnerMedecins.setSelection(state.getSelectedMedecinPosition());
69.         // calendrier
70.         edtJourRv.updateDate(state.getYear(), state.getMonth(), state.getDayOfMonth());
71.     }
72.
73.     @Override
74.     protected void notifyEndOfUpdates() {
75.     }
76.
77.     @Override
78.     protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
79.         // appelée après quand toutes les tâches sont terminées ou annulées
80.         // état menu
81.         initMenu();
82.         // vue suivante ?
83.         if (!runningTasksHaveBeenCalled) {
84.             mainActivity.navigateToView(IMainActivity.VUE_AGENDA, ISession.Action.SUBMIT);
85.         }
86.     }
87.     // méthodes privées -----
88.     private void initMenu() {
89.         // état menu
90.         setAllMenuOptionsStates(true);
91.         setMenuOptionsStates(new MenuItemState[]{new MenuItemState(R.id.actionAnnuler, false)});
92.     }

```

- lignes 2-9 : lorsque sa classe parent le lui demande, le fragment sauve l'état des éléments suivants :
 - ligne 6 : la position sélectionnée dans la liste des médecins ;
 - lignes 7-9 : le jour du mois, le mois et l'année de la date sélectionnée dans le calendrier ;
 - ligne 10 : la source de données du spinner des médecins ;
- lignes 14-17 : le n° du fragment est [IMainActivity.VUE_ACCUEIL] ;
- lignes 19-39 : exécutées lorsque le fragment est généré la 1ère fois (previousState==null) ou régénéré les fois suivantes (previousState !=null) ;
 - lignes 25-31 : dans le cas d'une 1ère visite, la source de données du spinner des médecins est construite ;
 - lignes 33-35 : pour les autres visites, la source de données du spinner est récupérée dans l'état précédent du fragment ;
- lignes 41-54 : exécutées lorsque la vue associée au fragment est construite la 1ère fois (previousState==null) ou reconstruite les fois suivantes (previousState !=null) ;
 - lignes 50-53 : pour la 1ère visite, on affiche le menu sans l'action [Annuler] (lignes 88-92) ;
 - lignes 43-48 : pour toutes les visites, 1ère ou pas, on associe le spinner des médecins à sa source (lignes 44-46) et on fixe la date minimale du calendrier à la date d'aujourd'hui (ligne 48) ;
- lignes 56-60 : exécutées lorsqu'on arrive au fragment par une opération [SUBMIT]. On vient alors de la vue [CONFIG]. On met le menu dans son état initial ;
- lignes 62-70 : exécutées lorsqu'on arrive au fragment par une opération [NAVIGATION] ou [RESTORE] ;
 - ligne 67 : on repositionne le spinner des médecins sur le dernier médecin sélectionné ;
 - ligne 69 : on positionne le calendrier sur la dernière date choisie ;
- lignes 72-74 : exécutées lorsque toutes les mises à jour précédentes ont été faites. Il n'y a rien de plus à faire ;
- lignes 76-85 : exécutées lorsque toutes les tâches asynchrones sont terminées ;
 - ligne 80 : on remet le menu dans son état par défaut ;
 - lignes 82-84 : si les tâches se sont terminées normalement, alors on passe à la vue suivante, sinon on reste sur la même vue ;

3.6.8 Gestion de la vue Agenda

3.6.8.1 La vue

La vue d'accueil est la suivante :

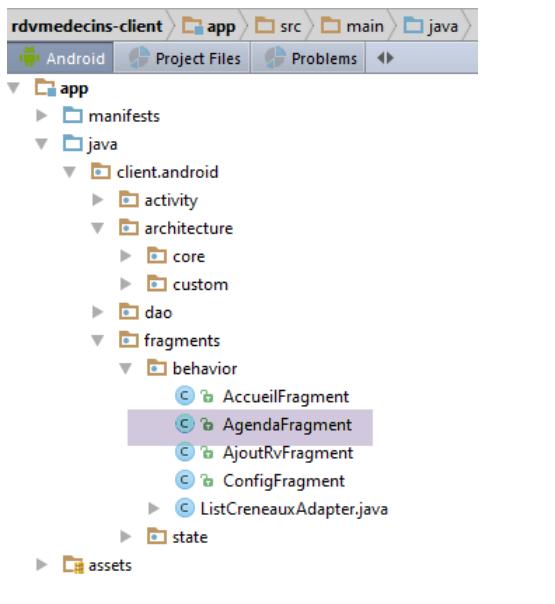


Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	TextView	txtTitre2
2	ListView	lstCreneaux

3.6.8.2 Le fragment

La vue Agenda est gérée par le fragment [AgendaFragment] suivant :



```

1. package client.android.fragments.behavior;
2.
3. import android.util.Log;
4. import android.view.View;
5. import android.widget.ArrayAdapter;
6. import android.widget.ListView;
7. import android.widget.TextView;
8. import android.widget.Toast;
9. import client.android.R;
10. import client.android.architecture.core.AbstractFragment;
11. import client.android.architecture.core.ISession;
12. import client.android.architecture.core.MenuInstanceState;
13. import client.android.architecture.custom.CoreState;
14. import client.android.architecture.custom.IMainActivity;
15. import client.android.dao.entities.AgendaMedecinJour;
16. import client.android.dao.entities.CreneauMedecinJour;
17. import client.android.dao.entities.Medecin;
18. import client.android.dao.entities.Rv;
19. import client.android.dao.service.Response;
20. import client.android.fragments.state.AgendaFragmentState;
21. import org.androidannotations.annotations.EFragment;
22. import org.androidannotations.annotations.OptionsItem;
23. import org.androidannotations.annotations.OptionsMenu;
24. import org.androidannotations.annotations.ViewById;
25. import rx.functions.Action1;
26.
27. @EFragment(R.layout.agenda)
28. @OptionsMenu(R.menu.menu_agenda)
29. public class AgendaFragment extends AbstractFragment {
30.
31.     // les éléments de l'interface visuelle
32.     @ViewById(R.id.txt_titre2_agenda)
33.     protected TextView txtTitre2;
34.     @ViewById(R.id.listViewAgenda)
35.     protected ListView lstCreneaux;
36.
37.     // agenda affiché par le fragment
38.     private AgendaMedecinJour agenda;
39.     // infos ListView des créneaux
40.     private int firstPosition;
41.     private int top;
42.     // rdv supprimé ou non
43.     private boolean rdvSupprimé;

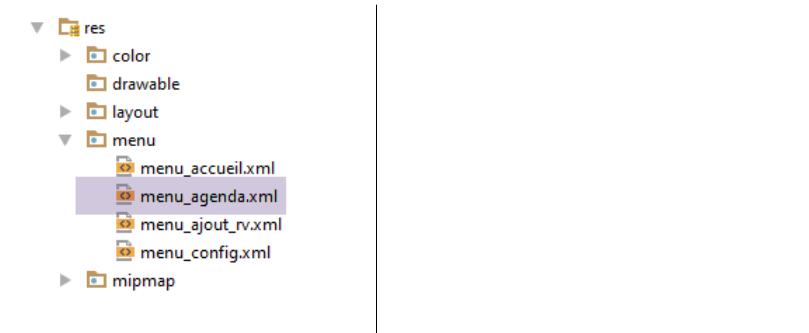
```

```

44.    // n° du créneau ajouté ou supprimé
45.    private int numCreneau;
46.
47.    // mise à jour de l'agenda après un ajout / suppression
48.    private void updateAgenda() {
49.        ...
50.    }
51.
52.    ...
53.
54.    // implémentation méthodes classe parent -----
55.    ...
56. }

```

- ligne 27 : le fragment est associé au menu [menu_accueil] suivant :



```

a) <menu xmlns:android="http://schemas.android.com/apk/res/android"
b)     xmlns:app="http://schemas.android.com/apk/res-auto"
c)     xmlns:tools="http://schemas.android.com/tools"
d)     tools:context=".activity.MainActivityI">
e)     <item
f)         android:id="@+id/menuActions"
g)         app:showAsAction="ifRoom"
h)         android:title="@string/menuActions">
i)         <menu>
j)             <item
k)                 android:id="@+id/actionAnnuler"
l)                 android:title="@string/actionAnnuler"/>
m)             <item
n)                 android:id="@+id/actionAgenda"
o)                 android:title="@string/actionAgenda"/>
p)         </menu>
q)     </item>
r)     <item
s)         android:id="@+id/menuNavigation"
t)         app:showAsAction="ifRoom"
u)         android:title="@string/menuNavigation">
v)         <menu>
w)             <item
x)                 android:id="@+id/navigationToConfig"
y)                 android:title="@string/navigationToConfig"/>
z)             <item
aa)                 android:id="@+id/navigationToAccueil"
ab)                 android:title="@string/navigationToAccueil"/>
ac)         </menu>
ad)     </item>
ae) </menu>

```

- lignes 32-35 : les éléments de l'interface visuelle ;
- lignes 37-45 : données globales aux méthodes ;

3.6.8.2.1 Méthode [updateAgenda]

La (ré)génération de la liste des créneaux de l'agenda est nécessaire à plusieurs endroits du code. Elle a été factorisée dans la méthode privée [updateAgenda] suivante :

```

1.    // mise à jour de l'agenda après un ajout / suppression
2.    private void updateAgenda() {
3.        // (ré)génération des créneaux de l'agenda
4.        // l'agenda est pris dans la session et mémorisé dans un champ du fragment
5.        agenda = session.getAgenda();
6.        // régénération du ListView des créneaux
7.        ArrayAdapter<CreneauMedecinJour> adapter = new ListCreneauxAdapter(activity, R.layout.creneau_medecin,
8.            agenda.getCreneauxMedecinJour(), this);
9.        lstCreneaux.setAdapter(adapter);

```

```

10.    // on se repositionne au bon endroit du ListView
11.    1stCreneaux.setSelectionFromTop(firstPosition, top);
12. }

```

- ligne 5 : l'agenda est pris dans la session et mémorisé dans le champ [agenda] du fragment ;
- lignes 7-9 : on définit l'adaptateur du composant [ListView]. Cet adaptateur définit à la fois la source de données du [ListView] et le modèle d'affichage de chaque élément de celle-ci. Nous allons présenter cet adaptateur prochainement ;
- ligne 11 : on revient sur la position précédente de l'agenda. En effet, on ne voit qu'une partie des créneaux de la journée. Si on ajoute / supprime un rendez-vous dans le dernier créneau, le code ci-dessus va rafraîchir la page pour présenter le nouvel agenda. Ce rafraîchissement fait qu'on est alors positionné de nouveau sur le 1^{er} créneau, ce qui n'est pas souhaitable. La ligne 5 remédie à ce problème. On trouvera la description de cette solution à l'URL [\[http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-returning-to-a-listview\]](http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-returning-to-a-listview) ;

La classe [ListCreneauxAdapter] sert à définir une ligne du [ListView] :



On voit ci-dessus, que selon que le créneau a un rendez-vous ou non, l'affichage n'est pas le même. Le code de la classe [ListCreneauxAdapter] est le suivant :

```

1. ...
2.
3. public class ListCreneauxAdapter extends ArrayAdapter<CreneauMedecinJour> {
4.
5.     // le tableau des créneaux horaires
6.     private CreneauMedecinJour[] creneauxMedecinJour;
7.     // le contexte d'exécution
8.     private Context context;
9.     // l'id du layout d'affichage d'une ligne de la liste des créneaux
10.    private int layoutResourceId;
11.    // listener des clics
12.    private AgendaFragment vue;
13.
14.    // constructeur
15.    public ListCreneauxAdapter(Context context, int layoutResourceId, CreneauMedecinJour[] creneauxMedecinJour,
16.                               AgendaFragment vue) {
17.        super(context, layoutResourceId, creneauxMedecinJour);
18.        // on mémorise les infos
19.        this.creneauxMedecinJour = creneauxMedecinJour;
20.        this.context = context;
21.        this.layoutResourceId = layoutResourceId;
22.        this.vue = vue;
23.        // on trie le tableau des créneaux dans l'ordre des horaires
24.        Arrays.sort(creneauxMedecinJour, new MyComparator());
25.    }
26.
27.    @Override
28.    public View getView(final int position, View convertView, ViewGroup parent) {
29.        ...
30.    }
31.

```

```

32. // tri du tableau des créneaux
33. class MyComparator implements Comparator<CreneauMedecinJour> {
34. ...
35. }
36. }
```

- ligne 3 : la classe [ListCreneauxAdapter] doit étendre un adaptateur prédefini pour les [ListView], ici la classe [ArrayAdapter] qui comme son nom l'indique alimente le [ListView] avec un tableau d'objets, ici de type [CreneauMedecinJour]. Rappelons le code de cette entité :

```

1. public class CreneauMedecinJour implements Serializable {
2.
3.     private static final long serialVersionUID = 1L;
4.     // champs
5.     private Creneau creneau;
6.     private Rv rv;
7. ...
8. }
```

- la classe [CreneauMedecinJour] contient un créneau horaire (ligne 5) et un éventuel rendez-vous (ligne 6) ou *null* si pas de rendez-vous ;

Retour au code de la classe [ListCreneauxAdapter] :

- ligne 15 : le constructeur reçoit quatre paramètres :
 - l'activité Android en cours,
 - le fichier XML définissant le contenu de chaque élément du [ListView],
 - le tableau des créneaux horaires du médecin,
 - la vue elle-même ;
- ligne 24 : le tableau des créneaux horaires est trié dans l'ordre croissant des horaires ;

La méthode [getView] est chargée de générer la vue correspondant à une ligne du [ListView]. Celle-ci comprend trois éléments :



Nº	Id	Type	Rôle
1	txtCreneau	TextView	créneau horaire
2	txtClient	TextView	le client
3	btnValider	TextView	lien pour ajouter / supprimer un rendez-vous

Le code de la méthode [getView] est le suivant :

```

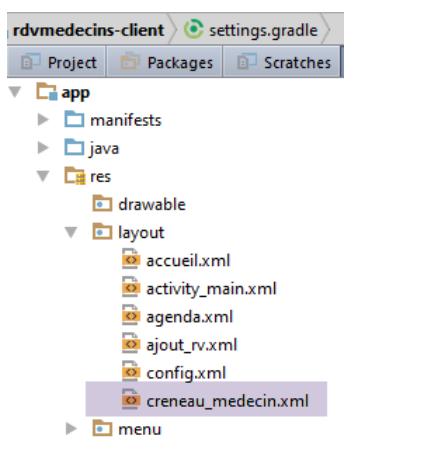
1. @Override
2.     public View getView(final int position, View convertView, ViewGroup parent) {
3.         // on se positionne sur le bon créneau
4.         CreneauMedecinJour creneauMedecin = creneauxMedecinJour[position];
5.         // on crée la ligne
6.         View row = ((Activity) context).getLayoutInflater().inflate(layoutResourceId, parent, false);
7.         // le créneau horaire
8.         TextView txtCreneau = (TextView) row.findViewById(R.id.txt_Creneau);
9.         txtCreneau.setText(String.format("%02d:%02d-%02d:%02d", creneauMedecin.getCreneau().getHdebut(), creneauMedecin
10.             .getCreneau().getMdebut(), creneauMedecin.getCreneau().getHfin(),
11.             creneauMedecin.getCreneau().getMfin())));
12.         // le client
13.         TextView txtClient = (TextView) row.findViewById(R.id.txt_Client);
14.         String text;
15.         if (creneauMedecin.getRv() != null) {
16.             Client client = creneauMedecin.getRv().getClient();
17.             text = String.format("%s %s %s", client.getTitre(), client.getPrenom(), client.getNom());
18.         } else {
19.             text = "";
20.         }
21.         txtClient.setText(text);
22.         // le lien
23.         final TextView btnValider = (TextView) row.findViewById(R.id.btn_Valider);
24.         if (creneauMedecin.getRv() == null) {
25.             // ajouter
26.             btnValider.setText(R.string.btn_ajouter);
27.             btnValider.setTextColor(context.getResources().getColor(R.color.blue));
28.         } else {
29.             // supprimer
30.             btnValider.setText(R.string.btn_supprimer);
31.             btnValider.setTextColor(context.getResources().getColor(R.color.red));
32.         }
33.     }
```

```

29.         btnValider.setText(R.string.btn_supprimer);
30.         btnValider.setTextColor(context.getResources().getColor(R.color.red));
31.     }
32.     // listener du lien
33.     btnValider.setOnClickListener(new OnClickListener() {
34.
35.         @Override
36.         public void onClick(View v) {
37.             // on passe les infos à la vue de l'agenda
38.             vue.doValider(position, btnValider.getText().toString());
39.         }
40.     });
41.     // on rend la ligne
42.     return row;
43. }

```

- ligne 2 : **position** est le n° de ligne qu'on va générer dans le [ListView]. C'est également le n° du créneau dans le tableau [creneauxMedecinJour]. On ignore les deux autres paramètres ;
- ligne 4 : on récupère le créneau horaire à afficher dans la ligne du [ListView] ;
- ligne 6 : la ligne est construite à partir de sa définition XML



Le code de [creneau_medecin.xml] est le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:id="@+id/RelativeLayout1"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:background="@color/wheat" >
7.
8.     <TextView
9.         android:id="@+id/txt_Creneau"
10.        android:layout_width="100dp"
11.        android:layout_height="wrap_content"
12.        android:layout_marginTop="20dp"
13.        android:layout_marginLeft="20dp"
14.        android:text="@string/txt_dummy" />
15.
16.     <TextView
17.         android:id="@+id/txt_Client"
18.         android:layout_width="200dp"
19.         android:layout_height="wrap_content"
20.         android:layout_alignBaseline="@+id/txt_Creneau"
21.         android:layout_marginLeft="20dp"
22.         android:layout_toRightOf="@+id/txt_Creneau"
23.         android:text="@string/txt_dummy" />
24.
25.     <TextView
26.         android:id="@+id/btn_Valider"
27.         android:layout_width="wrap_content"
28.         android:layout_height="wrap_content"
29.         android:layout_alignBaseline="@+id/txt_Client"
30.         android:layout_marginLeft="20dp"
31.         android:layout_toRightOf="@+id/txt_Client"
32.         android:text="@string/btn_valider"
33.         android:textColor="@color/blue" />
34.
35. </RelativeLayout>

```

- lignes 8-10 : le créneau horaire [1] est construit ;
- lignes 12-20 : l'identité du client [2] est construite ;
- ligne 23 : si le créneau n'a pas de rendez-vous ;
- lignes 25-26 : on construit le lien [Ajouter] de couleur bleue ;
- lignes 29-30 : sinon on construit le lien [Supprimer] de couleur rouge ;
- lignes 33-40 : quelque soit la nature lien [Ajouter / Supprimer] c'est la méthode [doValider] de la vue qui gèrera le clic sur le lien. La méthode recevra deux arguments :
 1. le n° du créneau qui a été cliqué,
 2. le libellé du lien qui a été cliqué ;
- ligne 42 : on rend la ligne qu'on vient de construire.

On notera que c'est la méthode [doValider] du fragment [AgendaFragment] qui gère les liens. Celle-ci est la suivante :

```

1.   // clic sur un lien [Ajouter / Supprimer]
2.   public void doValider(int numCreneau, String texte) {
3.       // opération en cours ?
4.       if (numberOfRunningTasks != 0) {
5.           Toast.makeText(activity, "Une opération est en cours. Patientez ou Annulez...", Toast.LENGTH_SHORT).show();
6.           return;
7.       }
8.       // on note la position du scroll pour y revenir
9.       // lire [http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-returning-to-a-
    listview]
10.      // position du 1er élément visible complètement ou non
11.      firstPosition = lstCreneaux.getFirstVisiblePosition();
12.      // offset Y de cet élément par rapport au haut du ListView
13.      // mesure la hauteur de la partie éventuellement cachée
14.      View v = lstCreneaux.getChildAt(0);
15.      top = (v == null) ? 0 : v.getTop();
16.      // on note également le n° du créneau cliqué
17.      this.numCreneau = numCreneau;
18.      // selon le texte du lien, on ne fait pas la même chose
19.      if (texte.equals(getResources().getString(R.string.lnk_ajouter))) {
20.          doAjouter();
21.      } else {
22.          doSupprimer();
23.      }
24.  }
```

- la méthode [doValider] reçoit deux informations :
 - le n° du créneau qui a été cliqué ;
 - le texte (Ajouter / Supprimer) du lien qui a été cliqué ;
- lignes 4-7 : le clic sur les liens [Supprimer / Ajouter] est inhibé s'il y a des tâches asynchrones en cours. C'est un choix qui facilite l'écriture du code. Il peut être discuté ;
- lignes 11-15 : on note les informations (firstPosition, top) du *ListView* des créneaux dans des champs du fragment afin que la méthode privée [updateAgenda] puisse le régénérer avec la même position de scrolling ;
- ligne 17 : on note le n° du créneau cliqué ;
- lignes 19-23 : selon le texte du lien cliqué, on fait un ajout ou une suppression ;

3.6.8.2.2 Méthode [doSupprimer]

La méthode [doSupprimer] assure la suppression du rendez-vous du créneau cliqué :

```

1.   // suppression d'un rdv
2.   private void doSupprimer() {
3.       // attente de la fin de deux tâches
4.       beginWaiting(2);
5.       // on supprime le Rdv en tâche de fond
6.       rdvSupprimé = false;
7.       // identifiant du rv à supprimer
8.       long idRv = agenda.getCreneauxMedecinJour()[numCreneau].getRv().getId();
9.       // suppression par une tâche asynchrone
10.      executeInBackground(mainActivity.supprimerRv(idRv), new Action1<Response<Rv>>() {
11.
12.          @Override
13.          public void call(Response<Rv> responseRv) {
14.              // consommation du résultat
15.          }
16.      });
17.  }
```

```

15.         consumeRv(responseRv);
16.     }
17. });
18. }
19.
20. // consommation d'une réponse
21. private void consumeRv(Response<Rv> responseRv) {
22.     // erreur ?
23.     if (responseRv.getStatus() != 0) {
24.         // message
25.         showAlert(responseRv.getMessages());
26.         // annulation
27.         doAnnuler();
28.         // retour à l'UI
29.         return;
30.     }
31.     // on note que le rdv a été supprimé
32.     rdvSupprimé = true;
33.     // on demande l'agenda le plus récent
34.     executeInBackground(
35.         mainActivity.getAgendaMedecinJour(agenda.getMedecin().getId(), session.getDayRv()),
36.         new Action1<Response<AgendaMedecinJour>>() {
37.
38.             @Override
39.             public void call(Response<AgendaMedecinJour> responseAgendaMedecinJour) {
40.                 // on consomme la réponse
41.                 consumeAgenda(responseAgendaMedecinJour);
42.             }
43.         });
44.     }
45.
46.     // consommation d'un agenda
47.     private void consumeAgenda(Response<AgendaMedecinJour> responseAgendaMedecinJour) {
48.         // erreur ?
49.         if (responseAgendaMedecinJour.getStatus() != 0) {
50.             // message
51.             showAlert(responseAgendaMedecinJour.getMessages());
52.             // annulation
53.             doAnnuler();
54.             // retour à l'UI
55.             return;
56.         }
57.         // on met l'agenda dans la session
58.         session.setAgenda(responseAgendaMedecinJour.getBody());
59.         // on met à jour l'agenda de la vue
60.         updateAgenda();
61.     }

```

- ligne 4 : on signale à la classe parent qu'on va lancer deux tâches asynchrones et on commence l'attente de la fin de ces deux tâches ;
- ligne 8 : on récupère l'identifiant du rendez-vous à supprimer. En effet, le serveur a besoin de cette information ;
- lignes 9-18 : on demande la suppression du rendez-vous via une tâche asynchrone ;
 - ligne 10 : la méthode [executeInBackground] attend deux paramètres :
 - ligne 10 : le processus à exécuter et observer est fourni par la méthode [mainActivity.supprimerRv(idRv)] ;
 - lignes 10-17 : le second paramètre est une instance de type [Action1<T>] où T est le type rendu par le processus observé, ici [Response<Rv>]
 - ligne 15 : lorsqu'on reçoit la réponse, on la passe à la méthode [consumeRv] de la ligne 21 ;
- lignes 21-44 : on a reçu la réponse de la tâche asynchrone. On l'exploite ;
- lignes 23-30 : on regarde d'abord si le serveur a signalé une erreur dans le champ [status] de la réponse ;
 - ligne 25 : si erreur il y a, on affiche les messages que le serveur a mis dans le champ [messages] de la réponse ;
 - ligne 27 : on annule toutes les tâches ;
 - ligne 29 : on revient à l'Ui ;
- ligne 32 : s'il n'y a pas eu d'erreur, on note que le rendez-vous a été supprimé ;
- lignes 34-43 : plutôt que simplement supprimer le rendez-vous dans l'agenda actuellement affiché par le fragment, on demande le nouvel agenda du médecin. En effet, l'application est multi-utilisateurs et d'autres utilisateurs ont pu eux-aussi modifier l'agenda du médecin. Donc autant avoir le plus récent ;
- lignes 34-43, 47-61 : on refait si ce qui avait été fait dans le fragment [AccueilFragment] avec cette fois des informations prises dans la session ;

La méthode [beginWaiting] (ligne 4) est la suivante :

```

1. // début de l'attente
2. protected void beginWaiting(int numberOfWorkingTasks) {
3.     // on prépare le lancement des tâches
4.     beginWorkingTasks(numberOfWorkingTasks);
5.     // état des boutons et menus
6.     setAllMenuOptionsStates(false);

```

```

7.     setMenuOptionsStates(new MenuItemState[]{new MenuItemState(R.id.menuActions, true),new
    MenuItemState(R.id.actionAnnuler, true)});
8.
9. }
```

- ligne 4 : on indique à la tâche parent qu'on va lancer [numberOfRunningTasks] tâches ;
- ligne 6 : on rend invisibles toutes les options du menu ;
- ligne 7 : pour ensuite rendre visible l'option [Actions/Annuler] ;

3.6.8.2.3 Méthode [doAnnuler]

Le clic sur l'option de menu [Annuler] est géré par la méthode [doAnnuler] :

Le clic sur l'option de menu [Annuler] est géré par la méthode [doAnnuler] :

```

1.     @OptionsItem(R.id.actionAnnuler)
2.     protected void doAnnuler() {
3.         if (isDebugEnabled) {
4.             Log.d(className, "Annulation demandée");
5.         }
6.         // on annule les tâches asynchrones
7.         cancelRunningTasks();
8.     }
```

- ligne 8 : on demande à la classe parent d'annuler les tâches asynchrones ;

3.6.8.2.4 Option de menu [Retour à la configuration]

Le clic sur l'option de menu [Retour à la configuration] est géré de la façon suivante :

```

1.     @OptionsItem(R.id.navigationToConfig)
2.     protected void navigationToConfig() {
3.         // on navigue vers la vue de configuration
4.         mainActivity.navigateView(IMainActivity.VUE_CONFIG, ISession.Action.NAVIGATION);
5.     }
```

- ligne 4 : on navigue vers la vue de configuration avec l'action [NAVIGATION]. Cela signifie qu'on veut retrouver la vue de configuration dans l'état où on l'a laissée ;

3.6.8.2.5 Option de menu [Retour à l'accueil]

Le clic sur l'option de menu [Retour à l'accueil] est géré de façon similaire :

```

1.     @OptionsItem(R.id.navigationToAccueil)
2.     protected void navigationToAccueil() {
3.         // on navigue vers la vue d'accueil
4.         mainActivity.navigateView(IMainActivity.VUE_ACCUEIL, ISession.Action.NAVIGATION);
5.     }
```

3.6.8.3 Gestion du cycle de vie du fragment

Le fragment a l'état [AgendaFragmentState] suivant :

```

1. package client.android.fragments.state;
2.
3. import android.widget.ArrayAdapter;
4. import client.android.architecture.custom.CoreState;
5. import client.android.dao.entities.CreneauMedecinJour;
6.
7. public class AgendaFragmentState extends CoreState {
8.
9.     // titre vue
10.    private String titre;
11.    // ListView
12.    private int firstPosition;
13.    private int top;
14.
15.    // constructeurs
16.    public AgendaFragmentState() {
17.
18.    }
19.
20.    public AgendaFragmentState(String titre) {
21.        this.titre = titre;
22.    }
23.
24.    // getters et setters
25.    ...
```

26. }

- ligne 10 : le titre affiché en haut de la vue ;
- lignes 12-13 : permet de restituer le *scrolling* du *ListView* des créneaux du médecin ;

Le cycle de vie du fragment est implémenté de la façon suivante :

```
1. // implémentation méthodes classe parent -----
2. @Override
3. public CoreState saveFragment() {
4.     // sauvegarde état
5.     AgendaFragmentState state = new AgendaFragmentState();
6.     state.setTitre(txtTitre2.getText().toString());
7.     // on note la position du scroll pour y revenir
8.     // lire [http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-returning-to-a-listview]
9.     // position du 1er élément visible complètement ou non
10.    firstPosition = lstCreneaux.getFirstVisiblePosition();
11.    // offset Y de cet élément par rapport au haut du ListView
12.    // mesure la hauteur de la partie éventuellement cachée
13.    View v = lstCreneaux.getChildAt(0);
14.    top = (v == null) ? 0 : v.getTop();
15.    // on mémorise tout ça
16.    state.setTop(top);
17.    state.setFirstPosition(firstPosition);
18.    return state;
19. }
20.
21. @Override
22. protected int getNumView() {
23.     return IMainActivity.VUE_AGENDA;
24. }
25.
26. @Override
27. protected void initFragment(CoreState previousState) {
28.     // 1ère visite ?
29.     if (previousState != null) {
30.         // pas la 1ère visite
31.         AgendaFragmentState state = (AgendaFragmentState) previousState;
32.         // et les informations du ListView
33.         firstPosition = state.getFirstPosition();
34.         top = state.getTop();
35.     }
36. }
37.
38. @Override
39. protected void initView(CoreState previousState) {
40. }
41.
42. @Override
43. protected void updateOnSubmit(CoreState previousState) {
44.     // on récupère l'agenda
45.     agenda = session.getAgenda();
46.     // on génère le titre de la page
47.     Medecin medecin = agenda.getMedecin();
48.     txtTitre2.setText(String.format("Rendez-vous de %s %s %s le %s", medecin.getTitre(), medecin.getPrenom(),
49.         medecin.getNom(), session.getJourRv()));
50.     // état menu
51.     initMenu();
52. }
53.
54. @Override
55. protected void updateOnRestore(CoreState previousState) {
56.     // on régénère le titre de la page
57.     AgendaFragmentState state = (AgendaFragmentState) previousState;
58.     txtTitre2.setText(state.getTitre());
59. }
60.
61. @Override
62. protected void notifyEndOfUpdates() {
63.     // on régénère la liste des créneaux
64.     updateAgenda();
65. }
66.
67. @Override
68. protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
69.     // état menu
70.     initMenu();
71.     // si annulation mais rdv a été supprimé il faut mettre à jour l'agenda local
72.     if (runningTasksHaveBeenCalled && rdvSupprimé) {
73.         // on supprime le rendez-vous dans l'agenda local (on n'a pas pu avoir l'agenda global)
74.         agenda.getCreneauxMedecinJour()[numCréneau].setRv(null);
75.         // on met à jour l'interface visuelle
76.         updateAgenda();
77.     }
78. }
```

```

79.
80.
81.    // méthodes privées -----
82.    private void initMenu() {
83.        // état menu
84.        setAllMenuOptionsStates(true);
85.        setMenuOptionsStates(new MenuItemState[]{new MenuItemState(R.id.actionAnnuler, false)});
86.    }

```

- lignes 2-19 : lorsque sa classe parent le lui demande, le fragment sauve l'état des éléments suivants :
 - ligne 6 : le titre affiché en haut de la vue ;
 - lignes 7-17 : les informations (top, firstPosition) qui vont permettre de restituer le *scrolling* du *ListView* ;
- lignes 21-24 : le n° du fragment est [IMainActivity.VUE_AGENDA] ;
- lignes 26-35 : exécutées lorsque le fragment est généré la 1ère fois (previousState==null) ou réégénéré les fois suivantes (previousState !=null) ;
 - lignes 30-34 : si ce n'est pas la 1ère visite du fragment, on récupère les informations (top, firstPosition) qui vont permettre de restituer le *scrolling* du *ListView* ;
- lignes 38-40 : exécutées lorsque la vue associée au fragment est construite la 1ère fois (previousState==null) ou reconstruite les fois suivantes (previousState !=null). Il n'y a rien à faire ici parce que le *ListView* des créneaux va être généré par la méthode privée [updateAgenda] (lignes 61-65) ;
- lignes 42-52 : exécutées lorsqu'on arrive au fragment par une opération [SUBMIT]. On vient alors de la vue [ACCUEIL] ;
 - ligne 45 : on récupère l'agenda mis en session par [AccueilFragment] ;
 - lignes 47-49 : on génère le titre de la vue ;
 - le *ListView* des créneaux va lui être généré par la méthode privée [updateAgenda] (lignes 61-65) ;
- lignes 54-59 : exécutées lorsqu'on arrive au fragment par une opération [NAVIGATION] ou [RESTORE] ;
 - lignes 57-58 : on régénère le titre de la vue ;
 - le *ListView* des créneaux va lui être généré par la méthode privée [updateAgenda] (lignes 61-65) ;
- lignes 72-74 : exécutées lorsque toutes les mises à jour précédentes ont été faites. On met à jour le *ListView* des créneaux car cette mise à jour est nécessaire quelque soit la façon dont on arrive au fragment ;
- lignes 67-77 : exécutées lorsque toutes les tâches asynchrones sont terminées ;
 - ligne 70 : on remet le menu dans son état par défaut (lignes 82-86) ;
 - ligne 72 : il y avait deux tâches asynchrones. On regarde si la 1ère (la suppression du rendez-vous) a réussi, malgré une annulation ;
 - ligne 74 : si oui, on supprime le rendez-vous de l'agenda local
 - ligne 75 : et on met à jour l'affichage de celui-ci;

3.6.9 Gestion de la vue d'ajout d'un rendez-vous

3.6.9.1 La vue

La vue d'ajout d'un rendez-vous est la suivante :

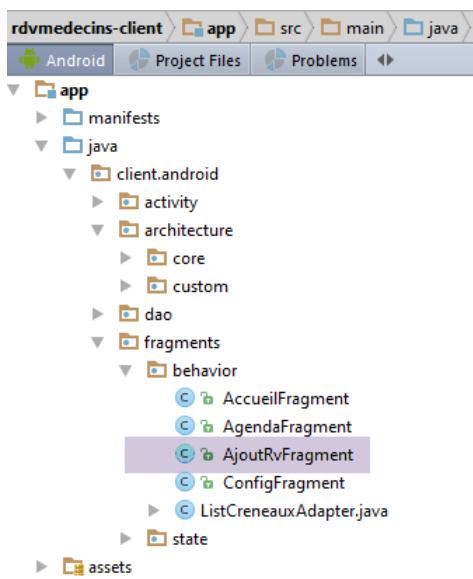


Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	TextView	txtTitre2
2	Spinner	spinnerClients

3.6.9.2 Le fragment

La vue d'ajout d'un rendez-vous est gérée par le fragment [AjoutRvFragment] suivant :

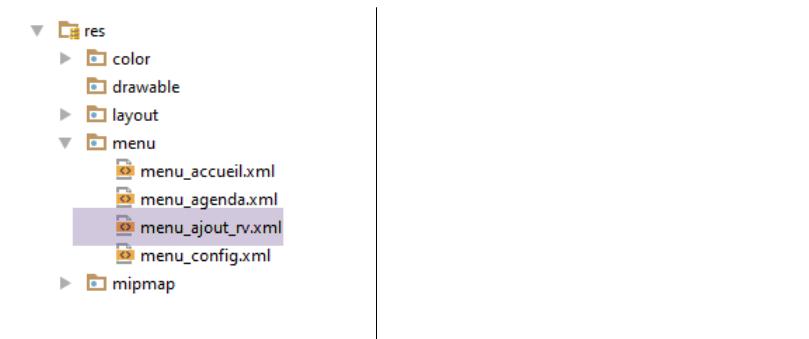


```
1. package client.android.fragments.behavior;
2.
3. import android.util.Log;
4. import android.widget.ArrayAdapter;
5. import android.widget.Spinner;
6. import android.widget.TextView;
7. import client.android.R;
8. import client.android.architecture.core.AbstractFragment;
9. import client.android.architecture.core.ISession;
10. import client.android.architecture.core.MenuItemState;
11. import client.android.architecture.custom.CoreState;
12. import client.android.architecture.custom.IMainActivity;
13. import client.android.dao.entities.*;
14. import client.android.dao.service.Response;
15. import client.android.fragments.state.AjoutRvFragmentState;
16. import org.androidannotations.annotations.EFragment;
17. import org.androidannotations.annotations.OptionsItem;
18. import org.androidannotations.annotations.OptionsMenu;
19. import org.androidannotations.annotations.ViewById;
20. import rx.functions.Action1;
21.
22. import java.util.List;
23. import java.util.Locale;
24.
25. @EFragment(R.layout.ajout_rv)
26. @OptionsMenu(R.menu.menu_ajout_rv)
27. public class AjoutRvFragment extends AbstractFragment {
28.
29.     // les éléments de l'interface visuelle
30.     @ViewById(R.id.spinnerClients)
31.     protected Spinner spinnerClients;
32.     @ViewById(R.id.txt_titre2_ajoutRv)
33.     protected TextView txtTitre2;
34.
35.     // les clients
36.     private List<Client> clients;
37.
38.     // données locales
39.     private Creneau creneau;
40.     private Medecin medecin;
41.     private boolean rdvAjouté;
42.     private Rv rv;
43.     private String[] spinnerClientsDataSource;
44.
45.     // validation de la page
46.     @OptionsItem(R.id.actionValider)
47.     protected void doValider() {
48.         ...
49.     }
50. ...
51.
```

```

52.    // implémentation méthodes classe parent -----
53. ...
54. }
```

- ligne 26 : le fragment est associé au menu [menu_ajout_rv] suivant :



```

a) <menu xmlns:android="http://schemas.android.com/apk/res/android"
b)   xmlns:app="http://schemas.android.com/apk/res-auto"
c)   xmlns:tools="http://schemas.android.com/tools"
d)   tools:context=".activity.MainActivityI"
e)   <item
f)     android:id="@+id/menuActions"
g)     app:showAsAction="ifRoom"
h)     android:title="@string/menuActions"
i)     <menu>
j)       <item
k)         android:id="@+id/actionValider"
l)         android:title="@string/actionValider"/>
m)       <item
n)         android:id="@+id/actionAnnuler"
o)         android:title="@string/actionAnnuler"/>
p)     </menu>
q)   </item>
r)   <item
s)     android:id="@+id/menuNavigation"
t)     app:showAsAction="ifRoom"
u)     android:title="@string/menuNavigation"
v)     <menu>
w)       <item
x)         android:id="@+id/navigationToConfig"
y)         android:title="@string/navigationToConfig"/>
z)       <item
aa)         android:id="@+id/navigationToAccueil"
ab)         android:title="@string/navigationToAccueil"/>
ac)       <item
ad)         android:id="@+id/navigationToAgenda"
ae)         android:title="@string/navigationToAgenda"/>
af)     </menu>
ag)   </item>
ah) </menu>
```

- lignes 30-33 : les éléments de l'interface visuelle ;
- ligne 36 : la liste des clients ;
- ligne 43 : la source de données du spinner des clients ;

Le clic sur le lien [Valider] est géré par la méthode [doValider] suivante :

```

1.  // les clients
2.  private List<Client> clients;
3.
4.  // données locales
5.  private Creneau creneau;
6.  private Medecin medecin;
7.  private boolean rdvAjouté;
8.  private Rv rv;
9.  private String[] spinnerClientsDataSource;
10. ...
11. // validation de la page
12. @OptionsItemSelected(R.id.actionValider)
13. protected void doValider() {
14.   // on récupère le client choisi
15.   Client client = clients.get(spinnerClients.getSelectedItemPosition());
16.   // début de l'attente
17.   beginWaiting(2);
18.   // on ajoute le RV
```

```

19.    rdvAjouté = false;
20.    executeInBackground(
21.        mainActivity.ajouterRv(session.getDayRv(), creneau.getId(), client.getId()),
22.        new Action1<Response<Rv>>() {
23.
24.            @Override
25.            public void call(Response<Rv> responseRv) {
26.                // on consomme la réponse
27.                consumeRv(responseRv);
28.            }
29.        });
30.    }
31.
32.    // consommation d'un objet Response<Rv>
33.    void consumeRv(Response<Rv> responseRv) {
34.        // erreur ?
35.        if (responseRv.getStatus() != 0) {
36.            // message
37.            showAlert(responseRv.getMessages());
38.            // annulation
39.            doAnnuler();
40.            // retour à l'UI
41.            return;
42.        }
43.        // on note que le rdv a été ajouté
44.        rdvAjouté = true;
45.        // on mémorise le rdv
46.        this.rv = responseRv.getBody();
47.        // on demande le nouvel agenda
48.        executeInBackground(mainActivity.getAgendaMedecinJour(session.getAgenda().getMedecin().getId(), session.getDayRv()),
49.            new Action1<Response<AgendaMedecinJour>>() {
50.
51.                @Override
52.                public void call(Response<AgendaMedecinJour> responseAgendaMedecinJour) {
53.                    // on consomme la réponse
54.                    consumeAgenda(responseAgendaMedecinJour);
55.                }
56.            });
57.
58.        // consommation d'un objet Response<AgendaMedecinJour>
59.        private void consumeAgenda(Response<AgendaMedecinJour> responseAgendaMedecinJour) {
60.            // erreur ?
61.            if (responseAgendaMedecinJour.getStatus() != 0) {
62.                // message
63.                showAlert(responseAgendaMedecinJour.getMessages());
64.                // annulation
65.                doAnnuler();
66.                // retour à l'UI
67.                return;
68.            }
69.            // on met l'agenda dans la session
70.            session.setAgenda(responseAgendaMedecinJour.getBody());
71.        }

```

- ligne 13 : lorsque commence la méthode [doValider], les champs 2, 5, 6 et 9 ont été initialisés lors du cycle de vie du fragment. Nous verrons comment ;
- ligne 15 : on récupère l'entité [Client] correspondant à l'élément sélectionné dans le spinner des clients ;
- ligne 17 : on indique à la classe parent qu'on va lancer deux tâches asynchrones et on prépare l'attente ;
- ligne 19 : au départ le rendez-vous n'est pas encore ajouté dans l'agenda du médecin ;
- lignes 20-30 : on demande au serveur l'ajout d'un rendez-vous ;
 - ligne 20 : la méthode [executeInBackground] attend deux paramètres :
 - ligne 20 : le processus à exécuter et observer est fourni par la méthode [mainActivity.ajouterRv(session.getDayRv(), creneau.getId(), client.getId())] ;
 - lignes 22-29 : le second paramètre est une instance de type [Action1<T>] où T est le type rendu par le processus observé, ici [Response<Rv>]
 - ligne 27 : lorsqu'on reçoit la réponse, on la passe à la méthode [consumeRV] de la ligne 33 ;
- lignes 33-56 : on a reçu la réponse du serveur. On l'exploite ;
 - lignes 35-42 : on regarde d'abord si le serveur a signalé une erreur dans le champ [status] de la réponse ;
 - ligne 37 : si erreur il y a, on affiche les messages que le serveur a mis dans le champ [messages] de la réponse ;
 - ligne 39 : on annule toutes les tâches ;
 - ligne 41 : on revient à l'Ui ;
 - ligne 44 : s'il n'y a pas eu d'erreur, on note que le rendez-vous a été ajouté ;
 - ligne 46 : on mémorise le rendez-vous ajouté dans un champ du fragment ;
 - lignes 47-55 : comme il avait été fait lors de la suppression d'un rendez-vous, après l'ajout du rendez-vous on demande au serveur l'agenda le plus récent du médecin ;
- lignes 47-56, 59-71 : on a là un code déjà rencontré plusieurs fois ;

La méthode [beginWaiting] (ligne 17) est la suivante :

```
1. // début de l'attente
2. protected void beginWaiting(int numberofRunningTasks) {
3.     // on prépare le lancement des tâches
4.     beginRunningTasks(numberofRunningTasks);
5.     // état des boutons et menus
6.     setAllMenuOptionsStates(false);
7.     setMenuOptionsStates(new MenuItemState[] [new MenuItemState(R.id.menuActions, true),new
    MenuItemState(R.id.actionAnnuler, true)]);
8.
9. }
```

- ligne 4 : on indique à la tâche parent qu'on va lancer [numberofRunningTasks] tâches ;
- ligne 6 : on rend invisibles toutes les options du menu ;
- ligne 7 : pour ensuite rendre visible l'option [Actions/Annuler] ;

Le clic sur l'option de menu [Annuler] est géré par la méthode [doAnnuler] :

```
9. @OptionsItem(R.id.actionAnnuler)
10. protected void doAnnuler() {
11.     if (isDebugEnabled) {
12.         Log.d(className, "Annulation demandée");
13.     }
14.     // on annule les tâches asynchrones
15.     cancelRunningTasks();
16. }
```

- ligne 8 : on demande à la classe parent d'annuler les tâches asynchrones ;

Les navigations arrières sont assurées par les trois méthodes suivantes :

```
1. @OptionsItem(R.id.navigationToConfig)
2. protected void navigationToConfig() {
3.     // on navigue vers la vue de configuration
4.     mainActivity.navigateView(IMainActivity.VUE_CONFIG, ISession.Action.NAVIGATION);
5. }
6.
7. @OptionsItem(R.id.navigationToAccueil)
8. protected void navigationToAccueil() {
9.     // on navigue vers la vue de configuration
10.    mainActivity.navigateView(IMainActivity.VUE_ACCUEIL, ISession.Action.NAVIGATION);
11. }
12.
13. @OptionsItem(R.id.navigationToAgenda)
14. protected void navigationToAgenda() {
15.     // on navigue vers la vue de l'agenda
16.     mainActivity.navigateView(IMainActivity.VUE_AGENDA, ISession.Action.NAVIGATION);
17. }
```

3.6.9.3 Gestion du cycle de vie du fragment

Le fragment a l'état [AjoutRvFragmentState] suivant :

```
1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. // état du fragment AjoutRvFragment
6. public class AjoutRvFragmentState extends CoreState {
7.
8.     // position client sélectionné
9.     private int selectedClientPosition;
10.    // titre vue
11.    private String titre;
12.    // source de données du spinner des clients
13.    private String[] spinnerClientsDataSource;
14.
15.    // getters et setters
16.    ...
17. }
```

Le cycle de vie du fragment est implémenté de la façon suivante :

```
1. // implémentation méthodes classe parent -----
2. @Override
3. public CoreState saveFragment() {
4.     // sauvegarde vue
```

```

5.     AjoutRvFragmentState state = new AjoutRvFragmentState();
6.     state.setTitre(txtTitre2.getText().toString());
7.     state.setSelectedClientPosition(spinnerClients.getSelectedItemPosition());
8.     state.setSpinnerClientsDataSource(spinnerClientsDataSource);
9.     return state;
10. }
11.
12. @Override
13. protected int getNumView() {
14.     return IMainActivity.VUE_AJOUT_RV;
15. }
16.
17. @Override
18. protected void initFragment(CoreState previousState) {
19.     // on récupère les clients en session
20.     clients = session.getClients();
21.     // 1ère visite ?
22.     if (previousState == null) {
23.         // on construit le tableau affiché par le spinner
24.         spinnerClientsDataSource = new String[clients.size()];
25.         int i = 0;
26.         for (Client client : clients) {
27.             spinnerClientsDataSource[i] = String.format("%s %s %s", client.getTitre(), client.getPrenom(), client.getNom());
28.             i++;
29.         }
30.     } else {
31.         // pas 1ère visite
32.         AjoutRvFragmentState state = (AjoutRvFragmentState) previousState;
33.         spinnerClientsDataSource = state.getSpinnerClientsDataSource();
34.     }
35. }
36.
37. @Override
38. protected void initView(CoreState previousState) {
39.     // association spinner à sa source de données
40.     ArrayAdapter<String> dataAdapterClients = new ArrayAdapter<>(activity, android.R.layout.simple_spinner_item,
41.         spinnerClientsDataSource);
42.     dataAdapterClients.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
43.     spinnerClients.setAdapter(dataAdapterClients);
44.     // 1ère visite ?
45.     if (previousState == null) {
46.         // menu
47.         initMenu();
48.     }
49. }
50.
51. @Override
52. protected void updateOnSubmit(CoreState previousState) {
53.     // on récupère le n° du créneau à réserver en session
54.     int position = session.getPosition();
55.     // on récupère l'agenda du médecin dans la session
56.     AgendaMedecinJour agenda = session.getAgenda();
57.     // on récupère le médecin et le créneau auquel on va mettre un rdv
58.     medecin = agenda.getMedecin();
59.     creneau = agenda.getCreneauxMedecinJour()[position].getCreneau();
60.     // on construit le titre 2 de la page
61.     String jour = session.getJourRv();
62.     txtTitre2.setText(String.format(Locale.FRANCE,
63.         "Prise de rendez-vous de %s %s %s le %s pour le créneau %02d:%02d-%02d:%02d", medecin.getTitre(),
64.         medecin.getPrenom(), medecin.getNom(), jour, creneau.getHdebut(), creneau.getMdebut(), creneau.getHfin(),
65.         creneau.getMfin()));
66.     // sélection client
67.     spinnerClients.setSelection(0);
68.     // menu
69.     initMenu();
70. }
71.
72. @Override
73. protected void updateOnRestore(CoreState previousState) {
74.     // restauration état précédent
75.     AjoutRvFragmentState state = (AjoutRvFragmentState) previousState;
76.     // titre
77.     txtTitre2.setText(state.getTitre());
78.     // spinner
79.     spinnerClients.setSelection(state.getSelectedClientPosition());
80. }
81.
82. @Override
83. protected void notifyEndOfUpdates() {
84. }
85.
86. @Override
87. protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
88.     // état menu
89.     initMenu();
90.     // vue suivante ?
91.     if (!runningTasksHaveBeenCalled) {

```

```

92.     mainActivity.navigateToView(IMainActivity.VUE_AGENDA, ISession.Action.SUBMIT);
93.     return;
94. }
95. // rdv déjà ajouté ?
96. if (rdvAjouté) {
97.     // on modifie l'agenda local (on n'a pas eu l'agenda global)
98.     AgendaMedecinJour agenda = session.getAgenda();
99.     agenda.getCreneauxMedecinJour()[session.getPosition()].setRv(rv);
100.    // on affiche l'agenda
101.    mainActivity.navigateToView(IMainActivity.VUE_AGENDA, ISession.Action.SUBMIT);
102.    return;
103. }
104. }
105.
106. // méthodes privées -----
107. private void initMenu() {
108.     // état menu
109.     setAllMenuOptionsStates(true);
110.     setMenuOptionsStates(new MenuItemState[]{new MenuItemState(R.id.actionAnnuler, false)});
111. }
112.

```

- lignes 2-10 : lorsque sa classe parent le lui demande, le fragment sauve l'état des éléments suivants :
 - ligne 6 : le titre en haut de la vue ;
 - ligne 7 : la position de l'élément sélectionné dans le spinner des clients ;
 - ligne 8 : la source de données du spinner des clients ;
- lignes 12-15 : le n° du fragment est [IMainActivity.VUE_AJOUT_RV] ;
- lignes 17-35 : exécutées lorsque le fragment est généré la 1ère fois (previousState==null) ou régénéré les fois suivantes (previousState !=null) ;
 - ligne 20 : on récupère la liste des clients dans la session pour la mettre dans un champ du fragment ;
 - lignes 22-30 : dans le cas d'une 1ère visite, la source de données du spinner des clients est construite ;
 - lignes 32-33 : pour les autres visites, la source de données du spinner des clients est récupérée dans l'état précédent du fragment ;
- lignes 37-49 : exécutées lorsque la vue associée au fragment est construite la 1ère fois (previousState==null) ou reconstruite les fois suivantes (previousState !=null) ;
 - lignes 40-43 : dans tous les cas, le spinner des clients est associé à sa source de données ;
 - lignes 45-48 : pour la 1ère visite, on affiche le menu sans l'action [Annuler] (lignes 107-111) ;
- lignes 51-70 : exécutées lorsqu'on arrive au fragment par une opération [SUBMIT]. On vient alors de la vue [AGENDA] ;
 - ligne 54 : on récupère le n° du créneau auquel on va mettre un rendez-vous ;
 - lignes 56-59 : on récupère l'entité [Medecin] et l'entité [Creneau] nécessaires pour l'ajout de ce rendez-vous et on les met dans des champs du fragment ;
 - lignes 61-65 : avec ces informations, on est capable de construire le titre de la vue ;
 - ligne 67 : le spinner des clients est positionné sur son 1er élément ;
 - ligne 69 : le menu est mis dans son état initial (sans l'option [Annuler]) ;
- lignes 72-80 : exécutées lorsqu'on arrive au fragment par une opération [NAVIGATION] ou [RESTORE] ;
 - ligne 77 : on régénère le titre de la vue ;
 - ligne 79 : on repositionne le spinner des clients sur le dernier client sélectionné ;
- lignes 82-84 : exécutées lorsque toutes les mises à jour précédentes ont été faites. Ici il n'y a rien de plus à faire ;
- lignes 86-104 : exécutées lorsque toutes les tâches asynchrones sont terminées ;
 - ligne 89 : on remet le menu dans son état par défaut ;
 - lignes 91-94 : si les tâches se sont terminées normalement, alors on revient à la vue [AGENDA] par un [SUBMIT] (ici, cela aurait pu être également une action de type NAVIGATION) ;
 - lignes 96-103 : si les tâches se sont terminées sur une annulation, on regarde quand même si le rendez-vous a été ajouté (cela signifierait que c'est l'obtention du nouvel agenda qui a échoué) ;
 - lignes 98-99 : si le rendez-vous a été ajouté ;
 - lignes 98-99 : le rendez-vous rendu par le serveur est ajouté à l'agenda courant, celui qui est en session ;
 - ligne 101 : on revient à la vue [AGENDA] par un [SUBMIT] (ici, cela aurait pu être également une action de type NAVIGATION) ;

3.7 Exécution

Faites les tests suivants :

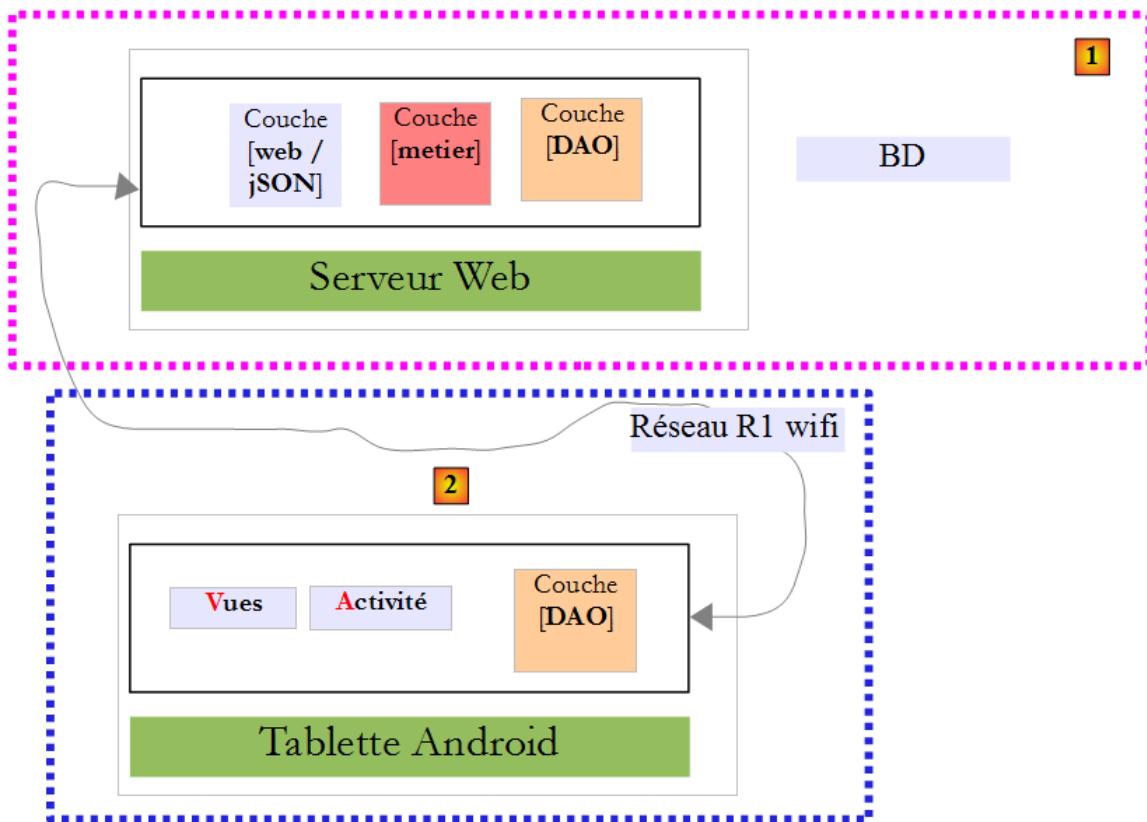
- utiliser l'application en conditions normales et vérifier qu'elle fonctionne ;
- faire tourner le périphérique pour chacune des vues et vérifier que chacune est restaurée correctement ;
- mettre un temps d'attente de quelques secondes dans [IMainActivity] ;
- procéder ensuite à l'annulation des tâches et vérifier que le résultat obtenu est celui attendu ;
- faire tourner le périphérique lors des attentes et vérifier que les tâches sont bien annulées et qu'il n'y a pas de crash ;
- changer l'adjacence des fragments dans [IMainActivity] et vérifier que l'application continue à fonctionner ;

4 TP 1 : Gestion basique d'une fiche de paie

4.1 Introduction

Pour appliquer ce qui a été vu précédemment, nous proposons maintenant un travail consistant à écrire un client Android pour tablette, permettant de simuler des calculs de feuille de salaire des employés d'une association.

L'application aura une architecture client / serveur :



- le serveur [1] est fourni ;
- il faut construire le client Android [2].

4.2 La base de données

4.2.1 Définition

Les données statiques utiles pour construire la fiche de paie seront placées dans une base de données que nous désignerons par la suite **dbpam**. Cette base de données a les tables suivantes :

Table **EMPLOYES** : rassemble des informations sur les différentes assistantes maternelles

Structure :

ID	clé primaire
VERSION	n° de version – augmente à chaque modification de la ligne
SS	numéro de sécurité sociale de l'employé - unique
NOM	nom de l'employé
PRENOM	son prénom

ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDEMNITE_ID	clé étrangère sur le champ [ID] de la table [INDEMNITES]

Son contenu pourrait être le suivant :

ID	Prenom	SS	ADRESSE	CP	VILLE	NOM	VERSION	INDEMNITE_ID
5	Marie	254104940426058	5 rue des oiseaux	49203	St Corentin	Jouveinal	1	8
6	Justine	260124402111742	La Brûlerie	49014	St Marcel	Laverti	1	7

Table **COTISATIONS** : rassemble des pourcentages nécessaires au calcul des cotisations sociales

Structure :

ID	clé primaire
VERSION	n° de version – augmente à chaque modification de la ligne
CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : sécurité sociale, veuvage, vieillesse
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

Son contenu pourrait être le suivant :

ID	SECU	RETRAITE	CSGD	CSGRDS	VERSION
3	9.39	7.88	6.15	3.49	1

Les taux des cotisations sociales sont indépendants du salarié. La table précédente n'a qu'une ligne.

Table **INDEMNITES** : rassemble les éléments permettant le calcul du salaire à payer.

ID	clé primaire
VERSION	n° de version – augmente à chaque modification de la ligne
INDICE	indice de traitement - unique
BASEHEURE	prix net en euro d'une heure de garde
ENTRETIENJOUR	indemnité d'entretien en euro par jour de garde
REPASJOUR	indemnité de repas en euro par jour de garde
INDEMNITESCP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.

Son contenu pourrait être le suivant :

ID	ENTRETIEN_JOUR	REPAS_JOUR	INDICE	INDEMNITES_CP	BASE_HEURE	VERSION
7		2	3	1	12	1.93
8		2.1	3.1	2	15	2.1

On notera que les indemnités peuvent varier d'une assistante maternelle à une autre. Elles sont en effet associées à une assistante maternelle précise via l'indice de traitement de celle-ci. Ainsi Mme Marie Jouveinal qui a un indice de traitement de 2 (table EMPLOYES) a un salaire horaire de 2,1 euro (table INDEMNITES).

4.2.2 Génération

Le script [dbpam_hibernate.sql] de génération de la base de données est fourni :



Créez la base de données [dbpam_hibernate] (c'est le nom de la BD que le serveur web / JSON exploite) et faites en sorte que le login *root* sans mot de passe puisse y accéder. Vous pouvez procéder ainsi :

Lancez MySQL puis [PhpMyAdmin] :

- [1-2] : importez le script [dbpam_hibernate.sql] puis exécutez-le ;

4.2.3 Modélisation Java de la base

Les éléments des tables [EMPLOYES], [INDEMNITES] et [COTISATIONS] sont modélisés par les classes suivantes :

[Employe]

```

1. package pam.entities;
2.
3. import java.io.Serializable;
4.
5. public class Employe implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     private Long id;
9.     private int version;
10.    private String SS;
11.    private String nom;
12.    private String prenom;
13.    private String adresse;
14.    private String ville;
15.    private String codePostal;
16.    private int idIndemnite;
17.    private Indemnite indemnite;
18.
19.    public Employe() {
20.    }
21.
22.    public Employe(String SS, String nom, String prenom, String adresse, String ville, String codePostal, Indemnite
indemnite) {
23.        ...
24.    }
25.    // getters et setters
26.    ....
27. }
```

- lignes 8-15 : ces champs correspondent aux colonnes de la table [EMPLOYES] ;

- ligne 16 : le champ [indemniteId] correspond à la colonne [INDEMNITE_ID] qui est la clé étrangère de la table [EMPLOYES] ;
- ligne 17 : l'indemnité de l'employé. Ce champ n'est pas toujours renseigné :
 - il ne l'est pas lorsqu'on demande l'URL [/employes],
 - il l'est lorsqu'on demande l'URL [/salaire] ;

[Indemnite]

```

1. package pam.entities;
2.
3. import java.io.Serializable;
4.
5. public class Indemnite implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     private Long id;
9.     private int version;
10.    private int indice;
11.    private double baseHeure;
12.    private double entretienJour;
13.    private double repasJour;
14.    private double indemnitesCp;
15.
16.    public Indemnite() {
17.    }
18.
19.    public Indemnite(int indice, double baseHeure, double entretienJour, double repasJour, double indemnitesCP) {
20.        ...
21.    }
22.
23.    // getters et setters
24.    ....
25. }
```

- lignes 8-14 : les champs correspondent aux colonnes de la table [INDEMNITES] ;

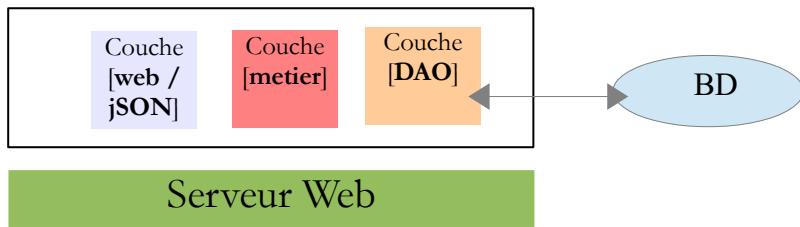
[Cotisation]

```

1. package pam.entities;
2.
3. import java.io.Serializable;
4.
5. public class Cotisation implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     private Long id;
9.     private int version;
10.    private double csgrds;
11.    private double csgd;
12.    private double secu;
13.    private double retraite;
14.
15.    public Cotisation() {
16.    }
17.
18.    public Cotisation(double csgrds, double csgd, double secu, double retraite) {
19.        ...
20.    }
21.    // getters et setters
22.    ...
23. }
```

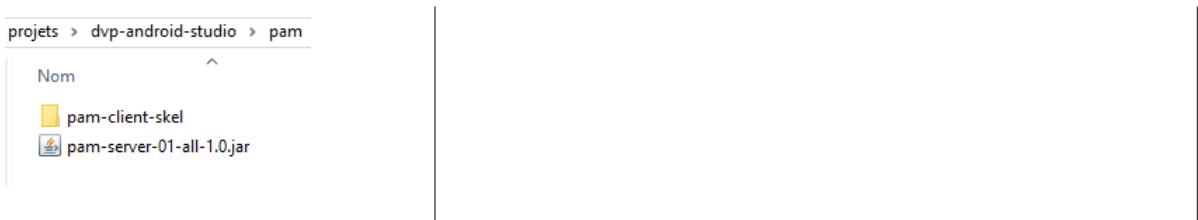
- lignes 8-13 : les champs correspondent aux colonnes de la table [COTISATIONS] ;

4.3 Installation du serveur web / JSON



4.3.1 Installation

Le binaire Java du serveur web / jSON est fourni :



Pour lancer le serveur web / JSON, procédez de la façon suivante :

- lancez le SBD MySQL ;
 - assurez-vous que la BD [dbpam_hibernate] existe ;
 - ouvrez une fenêtre DOS ;
 - placez vous dans le dossier du jar ;
 - tapez la commande :

```
java -jar pam-server-01-all-1.0.jar
```

Cela suppose que le binaire [java.exe] est dans le PATH de votre machine. Si ce n'est pas le cas, tapez le chemin complet de [java.exe], par exemple :

D:\Programs\devjava\java\jdk1.8\bin\java -jar pam-server-01-all-1.0.jar

Des logs s'affichent :

```
1. . \\\\_ | - - - - - ( ) - - - - - \\\_ \
2. ( ( ) \_ | - | - | - \_ | \_ | ) ) )
3. ( \_ | - | - | - | - | - | - | ) ) )
4. \_ | - | - | - | - | - | - | ) ) )
5. ' | - | - | - | - | - | - | ) ) )
6. ====== | - ====== | - / = / _ / /
7. :: Spring Boot ::      (v1.1.1.RELEASE)
8.
9. 2014-10-22 16:45:23.347 INFO 1868 --- [           main] pam.boot.BootWeb                  : Starting BootWeb on
   Gportpers3 with PID 1868 (D:\Temp\14-10-22\pam\server-pam.jar started by ST in D:\Temp\14-10-22\pam)
10. 2014-10-22 16:45:23.414 INFO 1868 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing
    org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@689ab9e: startup date [Wed Oct
22 16:45:23 CEST 2014]; root of context hierarchy
11. ...
12. ...
13. 2014-10-22 16:45:31.147 INFO 1868 --- [           main] org.hibernate.dialect.Dialect          : HHH000400: Using
dialect: org.hibernate.dialect.MySQLDialect
14. 2014-10-22 16:45:31.484 INFO 1868 --- [           main] o.h.h.i.ast.ASTQueryTranslatorFactory   : HHH000397: Using
ASTQueryTranslatorFactory
15. 2014-10-22 16:45:33.564 INFO 1868 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path
[/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
16. 2014-10-22 16:45:33.804 INFO 1868 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[~/salaire/
{SS}/{ht}/{jt},methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public
pam.restapi.FeuilleSalaireResponse pam.restapi.PamController.getFeuilleSalaire(java.lang.String,double,int)
17. 2014-10-22 16:45:33.805 INFO 1868 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"[~/employes],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public
pam.restapi.EmployeesResponse pam.restapi.PamController.getEmployees()
18. 2014-10-22 16:45:33.807 INFO 1868 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"[~/error],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public
```

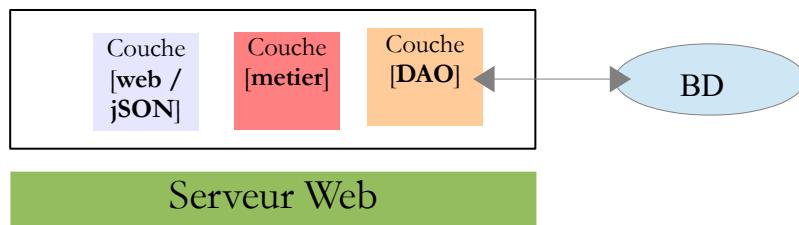
```

org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServlet)
19. 2014-10-22 16:45:33.807 INFO 1868 --- [           main] s.w.s.m.a.RequestMappingHandlerMapping : Mapped
"/{error},methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]" onto public
org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServlet)
20. 2014-10-22 16:45:33.839 INFO 1868 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path
[/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
21. 2014-10-22 16:45:33.839 INFO 1868 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path
[/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
22. 2014-10-22 16:45:34.384 INFO 1868 --- [           main] o.s.j.e.a.AnnotationMBeanExporter      : Registering beans
for JMX exposure on startup
23. 2014-10-22 16:45:34.535 INFO 1868 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on
port(s): 8080/http
24. 2014-10-22 16:45:34.538 INFO 1868 --- [           main] pam.boot.BootWeb                  : Started BootWeb in
11.916 seconds (JVM running for 12.725)
25. 2014-10-22 16:45:39.329 INFO 1868 --- [           Thread-2] a.tationConfigEmbeddedWebApplicationContext : Closing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@689ab9e2: startup date [Wed Oct
22 16:45:23 CEST 2014]; root of context hierarchy
26. 2014-10-22 16:45:39.331 INFO 1868 --- [           Thread-2] o.s.j.e.a.AnnotationMBeanExporter      : Unregistering JMX-
exposed beans on shutdown
27. 2014-10-22 16:45:39.333 INFO 1868 --- [           Thread-2] j.LocalContainerEntityManagerFactoryBean : Closing JPA
EntityManagerFactory for persistence unit 'default'

```

- ligne 16 : l'URL [/salaire/{SS}/{ht}/{jt}] est découverte ;
- ligne 17 : l'URL [/employes] est découverte ;

4.3.2 Les URL du service web/jSON



Le service web / jSON est implémenté par Spring MVC et expose deux URL :

```

1. @RequestMapping(value = "/employes", method = RequestMethod.GET, produces = "application/json; charset=UTF-8")
2. public EmployesResponse getEmployes() {
3. ...
4. @RequestMapping(value = "/salaire/{SS}/{ht}/{jt}", method = RequestMethod.GET, produces = "application/json; charset=UTF-
8")
5. public FeuilleSalaireResponse getFeuilleSalaire(@PathVariable("SS") String SS, @PathVariable("ht") double ht,
@PathVariable("jt") int jt) {

```

Le service web accepte les deux URL suivantes :

- ligne 1 : /employes : pour avoir la liste des employés ;
- ligne 4 : /salaire/SS/ht/jt : pour avoir la feuille de salaire de l'employé de n° [SS] ayant travaillé [ht] heures pendant [jt] jours ;

Voici des copies d'écran montrant cela.

On demande les employés :



On coupe la base, on relance le serveur et on demande les employés :

```
{"status": -1, "messages": ["Could not open JPA EntityManager for transaction; nested exception is org.hibernate.exception.JDBCConnectionException: Could not open connection", "Could not open connection", "Communications link failure\nThe last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.", "Connection refused: connect"], "body": null}
```

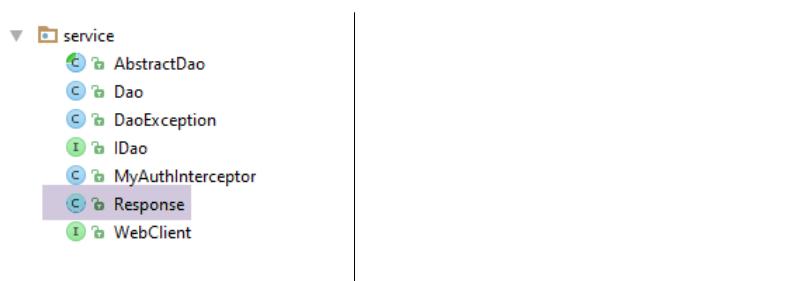
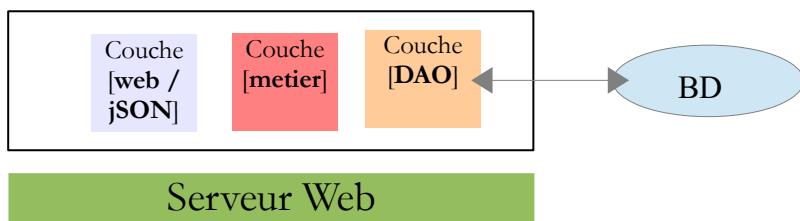
On demande un salaire :

```
{"status": 0, "messages": null, "body": {"employe": {"id": 494, "version": 0, "nom": "Jouveinal", "prenom": "Marie", "adresse": "5 rue des oiseaux", "ville": "St Corentin", "codePostal": "49203", "idIndemnite": 641, "indemnite": {"id": 641, "version": 0, "indice": 2, "baseHeure": 2.1, "entretienJour": 2.1, "repasJour": 3.1, "indemnitesCP": 15.0}, "ss": "254104940426058"}, "cotisation": {"id": 73, "version": 0, "csgrds": 3.49, "csgd": 6.15, "secu": 9.39, "retraite": 7.88}, "elementsSalaire": {"salaireBase": 362.25, "cotisationsSociales": 97.48, "indemnitesEntretien": 42.0, "indemnitesRepas": 62.0, "salaireNet": 368.77}}}}
```

On demande le salaire d'une personne inexistante :

```
{"status": 1, "messages": ["L'employé de n°[xx] est introuvable"], "body": null}
```

4.3.3 Les réponses JSON du service web/jSON



Les URL du service web / jSON envoient des réponses de type [Response<T>] :

```

1. package client.android.dao.service;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
```

```

7.      // ----- propriétés
8.      // statut de l'opération
9.      private int status;
10.     // les éventuels messages d'status
11.     private List<String> messages;
12.     // le corps de la réponse
13.     private T body;
14.
15.     // constructeurs
16.     public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.      this.status = status;
22.      this.messages = messages;
23.      this.body = body;
24.    }
25.
26.    // getters et setters
27. ...
28. }
```

- l'URL [/employes] renvoie un type **Response<List<Employe>>** ;
- l'URL [/salaire] renvoie un type **Response<FeuilleSalaire>** ;

La classe [FeuilleSalaire] est la suivante :

```

1. package pam.entities;
2.
3. import java.io.Serializable;
4.
5. public class FeuilleSalaire implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     // champs privés
9.     private Employe employe;
10.    private Cotisation cotisation;
11.    private ElementsSalaire elementsSalaire;
12.
13.    // constructeurs
14.    public FeuilleSalaire() {
15.    }
16.
17.    public FeuilleSalaire(Employe employe, Cotisation cotisation, ElementsSalaire elementsSalaire) {
18.      ...
19.    }
20.
21.    // getters et setters
22.    ...
23. }
```

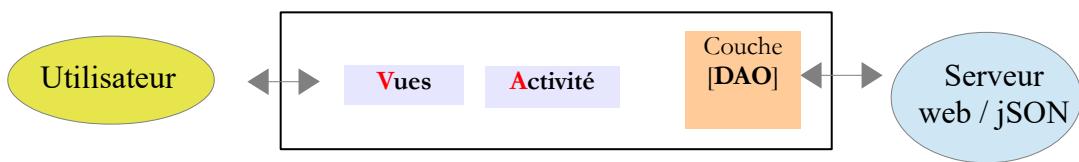
- ligne 9 : la classe [Employe] a été présentée page 403;
- ligne 10 : la classe [Cotisation] a été présentée page 404 ;

La classe [ElementsSalaire] (ligne 11) est la suivante :

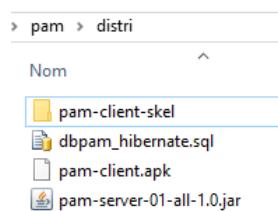
```

1. package pam.entities;
2.
3. import java.io.Serializable;
4.
5. public class ElementsSalaire implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     // champs privés
9.     private double salaireBase;
10.    private double cotisationsSociales;
11.    private double indemnitesEntretien;
12.    private double indemnitesRepas;
13.    private double salaireNet;
14.
15.    // constructeurs
16.    public ElementsSalaire() {
17.
18.    }
19.
20.    public ElementsSalaire(double salaireBase, double cotisationsSociales, double indemnitesEntretien, double
   indemnitesRepas, double salaireNet) {
21.      ...
22.    }
23.
24.    // getters et setters
```

4.4 Tests du client Android



Le binaire exécutable du client Android terminé vous est donné :



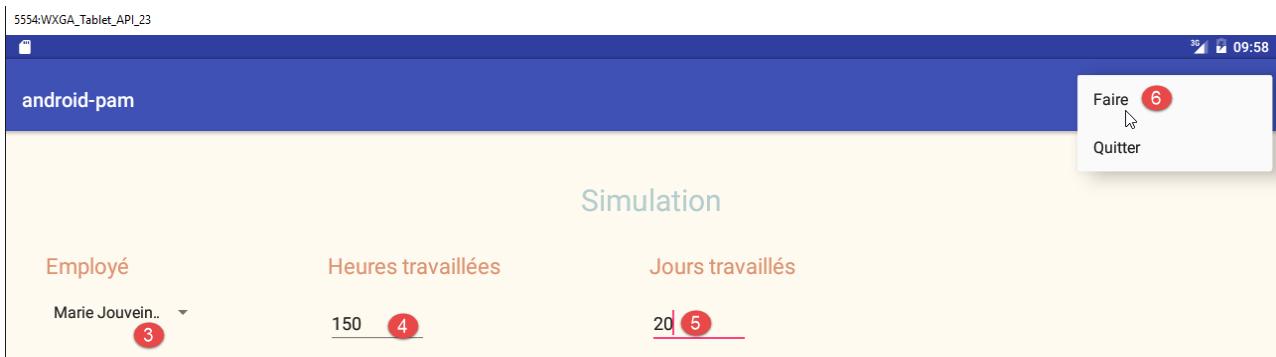
Avec la souris déposez le binaire [pam-client.apk] ci-dessus sur un émulateur de tablette [GenyMotion]. Il va alors être enregistré puis exécuté. Lancez également le serveur web / JSON si ce n'est déjà fait. Le client Android a pour objet de récupérer les informations renvoyées par le serveur web / JSON et de les mettre en forme. Les différentes vues du client Android sont les suivantes :

Il faut tout d'abord se connecter au service web / JSON :



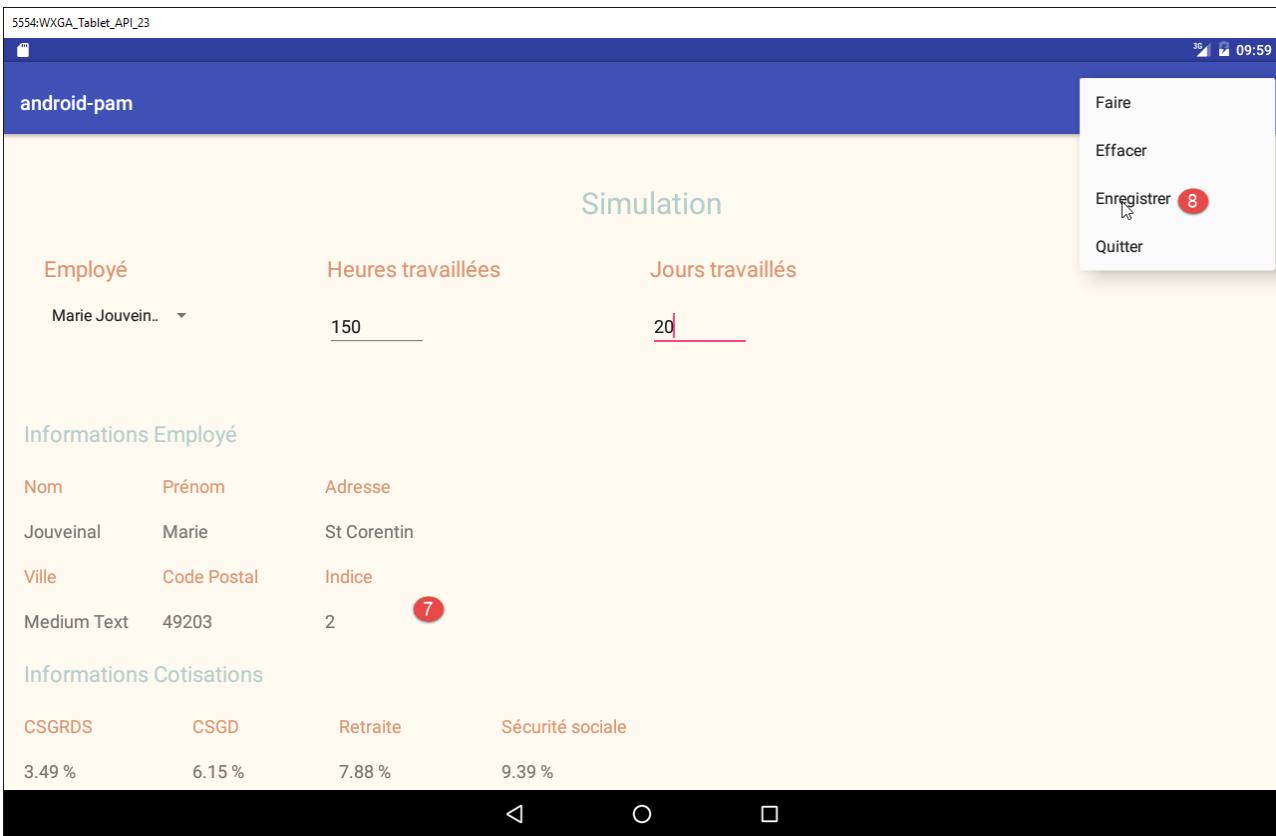
- en [1], on donne l'URL du service web / JSON. Avec l'émulateur, mettez l'une des adresses IP du PC (mais pas 127.0.0.1). Avec une tablette, mettez l'adresse wifi de la machine du serveur web / JSON et inhibez le pare-feu du poste serveur s'il en a un car il risque de bloquer les appels entrants ;
- en [2], on se connecte ;

On arrive alors à la page de simulation :



- en [3], on choisit un employé ;
- en [4], on indique un nombre d'heures ;
- en [5], on indique un nombre de jours ;
- en [6], on demande la simulation ;

La page de simulation obtenue est la suivante :



- en [7], la simulation obtenue ;
- en [8], on l'enregistre ;

android-pam

ACTIONS NAVIGATION

Liste de vos simulations

Vous avez une simulation 9

N°	Nom	Prénom	Heures	Jours	Base	Indemnités	Cotisations	Salaire net	
1	Jouveinal	Marie	150.0	20	362.25	104.0	97.48	368.77	Retirer 10

- en [9], la liste des simulations ;
- en [10], on retire une simulation ;

android-pam

vers Configuration

vers Formulaire 12

Liste de vos simulations

Vous n'avez aucune simulation 11

- en [11], il n'y a plus de simulations ;
- en [12], on retourne au formulaire de simulation ;

5554:WXGA_Tablet_API_23

android-pam

vers Configuration 14

Simulation

Employé	Heures travaillées	Jours travaillés
Marie Jouvein.. ▾	<u>150</u>	<u>20</u>

Informations Employé

Nom	Prénom	Adresse
Jouveinal	Marie	St Corentin

Ville	Code Postal	Indice
Medium Text	49203	<u>2</u> 13

Informations Cotisations

CSGRDS	CSGD	Retraite	Sécurité sociale
3.49 %	6.15 %	7.88 %	9.39 %

◀ ○ □

- en [13], on retrouve le formulaire ;
- en [14], on revient vers la page de configuration;

5554:WXGA_Tablet_API_23

android-pam

ACTIONS

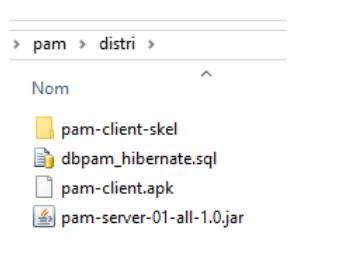
Se connecter au serveur 15

Url du service web / JSON 192.168.82.2:8080

- en [15], on retrouve le formulaire de connexion initial.

4.5 Travail à faire

Le squelette du client Android présenté précédemment vous est donné. Il a été construit à partir du projet [client-android-skel] décrit au paragraphe 2, page 251.



Le projet est exécutable et a déjà les vues nécessaires. Il y a simplement du code à rajouter pour que l'application fasse ce qu'elle a à faire. La procédure à suivre est la suivante :

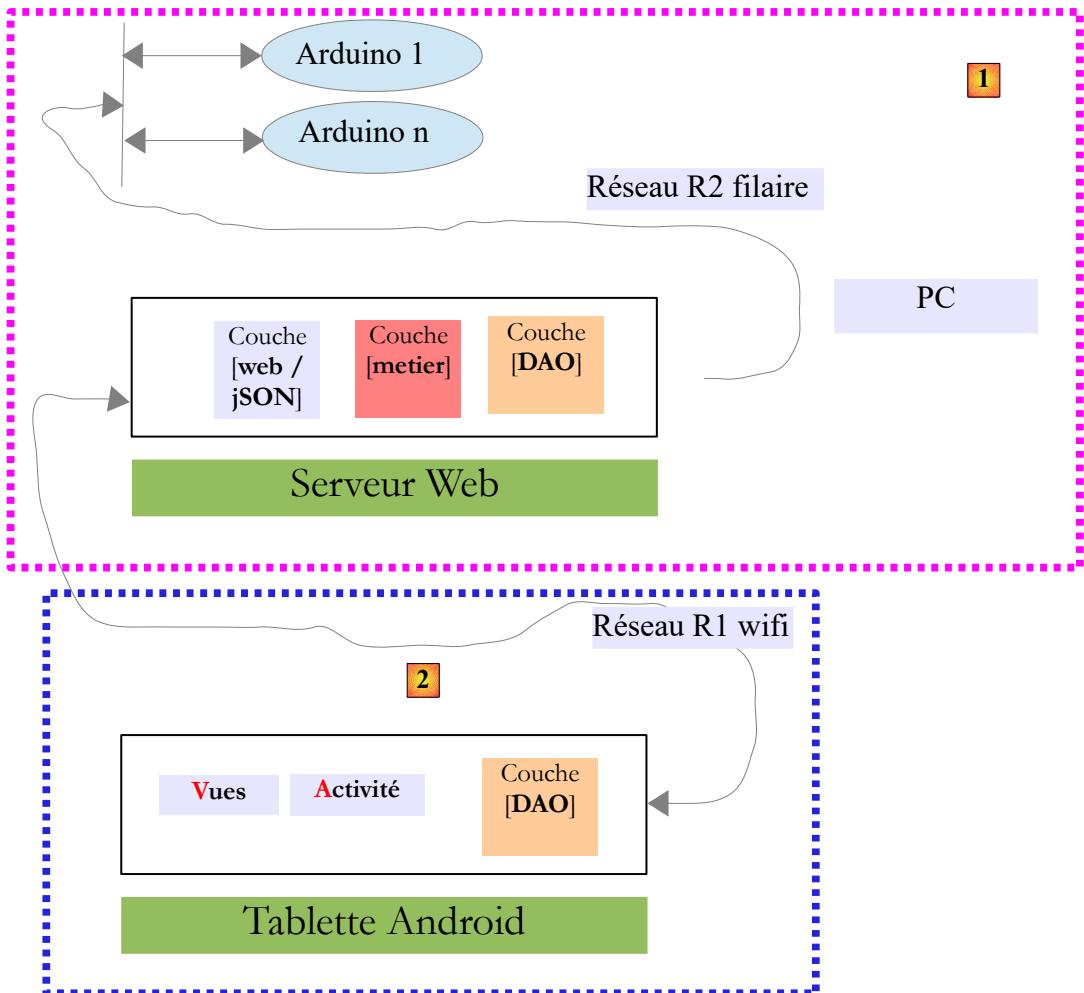
- exécutez la version complète pour appréhender le travail à faire ;
- exécutez la version allégée et étudiez le code de celle-ci. Il respecte les méthodes de conception utilisées dans les pages précédentes ;
- ajoutez le code manquant ;

5 TP 2 - Piloter des Arduinos avec une tablette Android

Nous allons maintenant apprendre à piloter une carte Arduino avec une tablette. L'exemple à suivre est celui du projet [client-android-skel] du cours (cf paragraphe 2, page 251).

5.1 Architecture du projet

L'ensemble du projet aura l'architecture suivante :



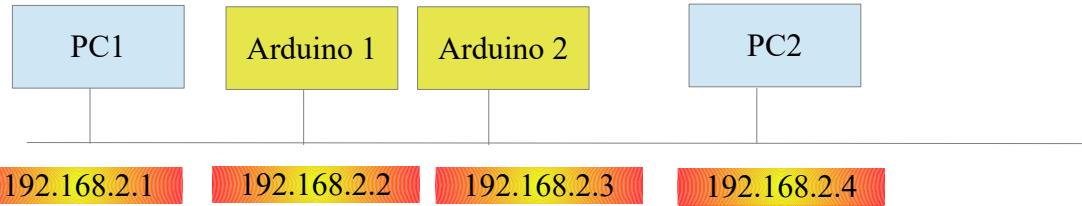
- le bloc [1], serveur web / JSON et Arduinos vous sera donné ;
- vous aurez à construire le bloc [2], la programmation de la tablette Android pour dialoguer avec le serveur web / JSON.

5.2 Le matériel

Vous avez à votre disposition les éléments suivants :

- un Arduino avec une extension Ethernet, une led et un capteur de température ;
- un miniHub à partager avec un autre étudiant ;
- un câble USB pour alimenter l'Arduino ;
- deux câbles réseau pour connecter l'Arduino et le PC sur un même réseau privé ;
- une tablette Android ;

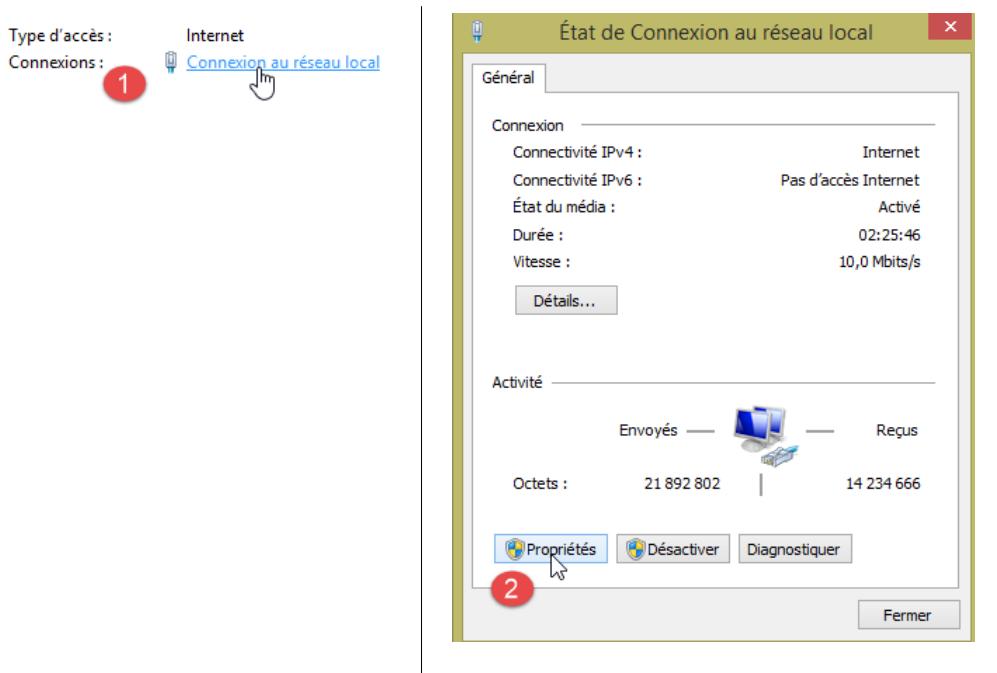
5.2.1 L'Arduino



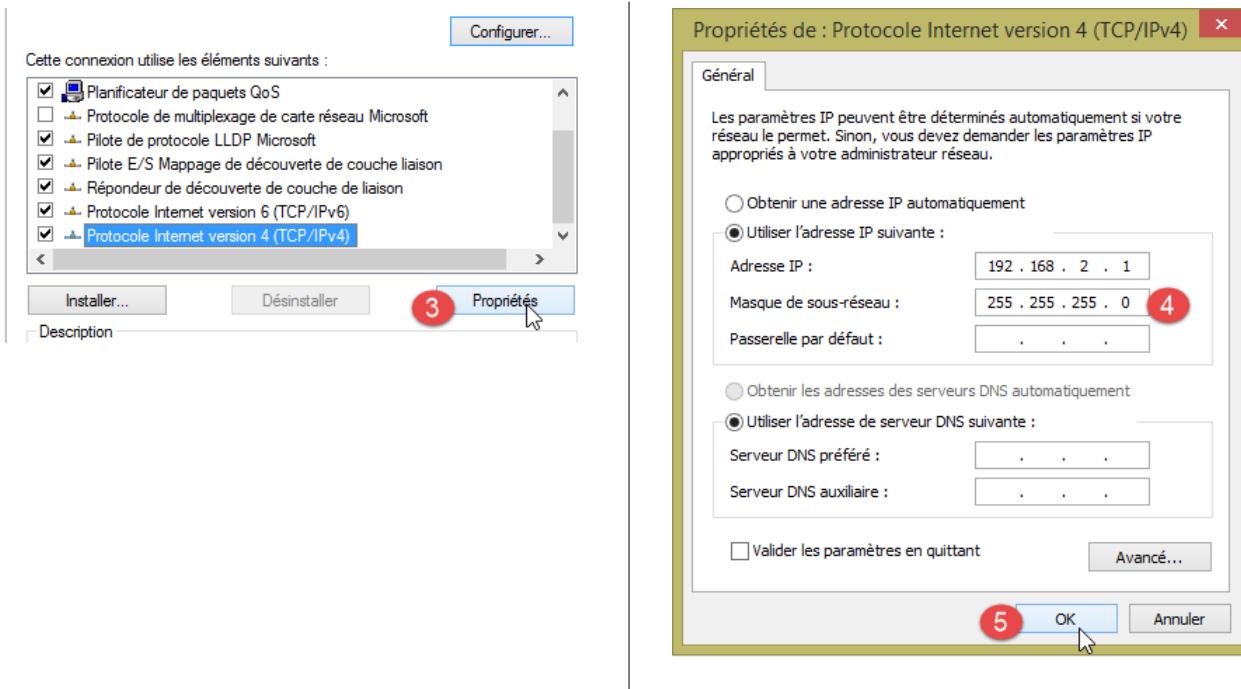
Voici comment procéder pour connecter les différents éléments ensemble :

- retirez le câble réseau de votre PC ;
- joignez votre PC et l'Arduino par un câble réseau ;
- l'Arduino dont vous disposerez aura déjà été programmé. Son adresse IP sera [192.168.2.2]. Pour que votre PC voit l'Arduino, il faut lui donner une adresse IP sur le réseau [192.168.2]. Les Arduinos ont été programmés pour dialoguer avec un PC ayant l'adresse IP [192.168.2.1]. Voici comment procéder :

Allez sur [Panneau de configuration\Réseau et Internet\Centre Réseau et partage] :



- en [1], cliquez sur le lien [réseau local] ;
- en [2], cliquez le bouton [Propriétés] du réseau local ;



- en [3], cliquez sur les propriétés [IPv4] de la carte [réseau local] ;
 - en [4], donnez à cette carte l'adresse IP [192.168.2.1] et le masque de sous-réseau [255.255.255.0] ;
 - en [5], cliquez sur [OK] autant de fois que nécessaire pour sortir de l'assistant.

5.2.2 La tablette

- à l'aide de votre clé wifi, connectez-votre poste au réseau wifi qu'on vous indiquera. Faites de même avec votre tablette ;
 - vérifiez l'adresse IP wifi de votre PC en faisant [ipconfig] dans une fenêtre DOS. Vous allez trouver une adresse du genre [192.168.x.y] ;

- vérifiez l'adresse IP wifi de votre tablette. Demandez à votre encadrant comment faire si vous ne savez pas. Vous allez trouver une adresse du genre [192.168.x.z] ;
 - inhibez le pare-feu de votre PC s'il est actif [Panneau de configuration\Système et sécurité\Pare-feu Windows] ;
 - dans une fenêtre Dos, vérifiez que le PC et la tablette peuvent communiquer en tapant la commande [ping 192.168.x.z] où [192.168.x.z] est l'adresse IP de votre tablette. La tablette doit alors répondre :

```
dos>ping 192.168.1.26

Envoi d'une requête 'Ping' 192.168.1.26 avec 32 octets de données :
Réponse de 192.168.1.26 : octets=32 temps=102 ms TTL=64
Réponse de 192.168.1.26 : octets=32 temps=134 ms TTL=64
Réponse de 192.168.1.26 : octets=32 temps=168 ms TTL=64
Réponse de 192.168.1.26 : octets=32 temps=208 ms TTL=64

Statistiques Ping pour 192.168.1.26:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 102ms Maximum = 208ms Moyenne = 153ms
```

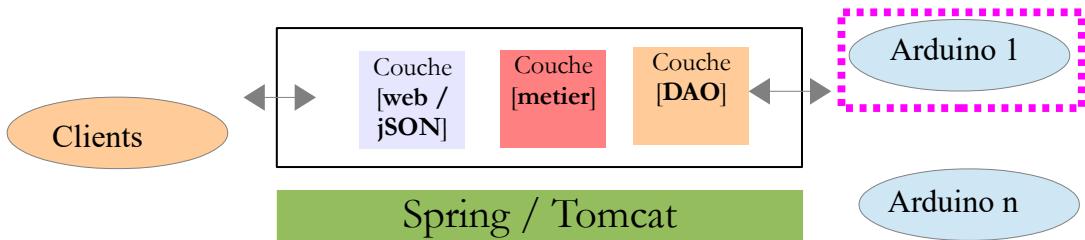
La configuration réseau de votre système est désormais prête.

5.2.3 L'émulateur [Genymotion]

L'émulateur [Genymotion] (cf paragraphe 6.9, page 480) remplace avantageusement la tablette. Il est quasiment aussi rapide et ne nécessite pas de réseau wifi. C'est cette méthode qu'il est conseillé d'utiliser. Vous pourrez utiliser la tablette pour la vérification finale de votre application.

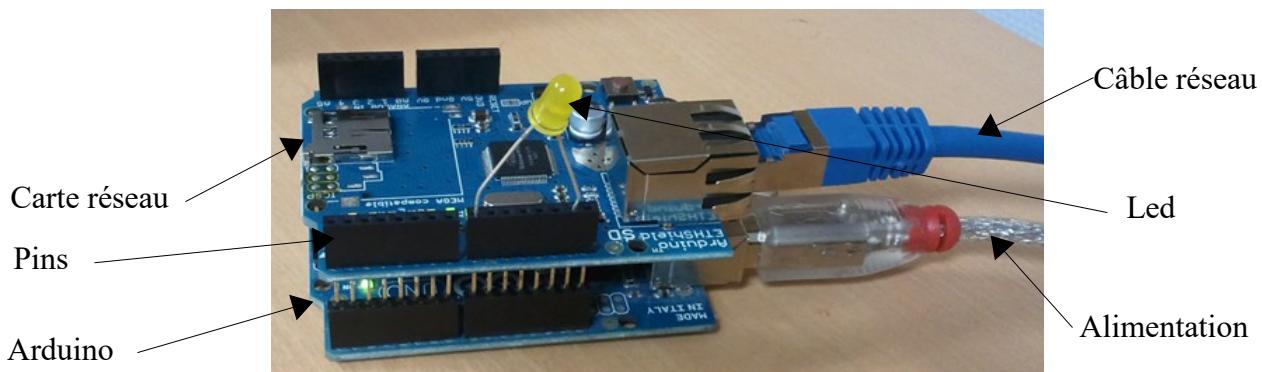
5.3 Programmation des Arduinos

Nous nous intéressons ici à l'écriture du code C des Arduinos :



A lire

- installation de l'IDE de développement Arduino (cf paragraphe 6.1, page 470) ;
- utilisation de bibliothèques JSON (Annexes, paragraphe 6.6) ;
- dans l'IDE de l'Arduino, tester l'exemple d'un serveur TCP (le serveur web par exemple) et celui d'un client TCP (le client Telnet par exemple) ;
- les annexes sur l'environnement de programmation des Arduinos au paragraphe 6.1, page 470.



Un Arduino est un ensemble de pins reliées à du matériel. Ces pins sont des entrées ou des sorties. Leur valeur est binaire ou analogique. Pour commander l'Arduino, il y aura deux opérations de base :

- **écrire une valeur** binaire / analogique sur une pin désignée par son numéro ;
- **lire une valeur** binaire / analogique sur une pin désignée par son numéro ;

A ces deux opérations de base, nous en ajouterons une troisième :

- **faire clignoter une led** pendant une certaine durée et avec une certaine fréquence. Cette opération peut être réalisée en appelant de façon répétée les deux opérations de base précédentes. Mais nous verrons aux tests que les échanges de la couche [DAO] avec un Arduino sont de l'ordre de la seconde. Il n'est alors pas possible de faire clignoter une led toutes les 100 millisecondes par exemple. Aussi implanterons-nous sur l'Arduino lui-même cette fonction de clignotement.

Le fonctionnement de l'Arduino sera le suivant :

- les communications entre la couche [DAO] et un Arduino se font via un réseau TCP-IP par échanges de lignes de texte au format JSON (JavaScript Object Notation) ;
- au démarrage, l'Arduino vient se connecter au port 100 d'un serveur d'enregistrement présent dans la couche [DAO]. Il envoie au serveur une unique ligne de texte :

```
{"id":"cuisine","desc":"duemilanove","mac":"90:A2:DA:00:1D:A7","port":102}
```

C'est une chaîne JSON caractérisant l'Arduino qui se connecte :

- **id** : un identifiant de l'Arduino ;
- **desc** : une description de ce que sait faire l'Arduino. Ici on a simplement mis le type de l'Arduino ;
- **mac** : adresse Mac de l'Arduino ;
- **port** : le numéro du port sur lequel l'Arduino va attendre les commandes de la couche [DAO].

Toutes ces informations sont de type chaînes de caractères sauf le port qui est un nombre entier.

- une fois que l'Arduino s'est inscrit auprès du serveur d'enregistrement, il se met à l'écoute sur le port qu'il a indiqué au serveur. Il attend des commandes JSON de la forme suivante :

```
{"id":"identifiant","ac":"une_action","pa":{"param1":"valeur1","param2":"valeur2",...}}
```

C'est une chaîne JSON avec les éléments suivants :

- **id** : un identifiant de la commande. Peut être quelconque ;
- **ac** : une action. Il y en a trois :
 - **pw (pin write)** pour écrire une valeur sur une pin,
 - **pr (pin read)** pour lire la valeur d'une pin,
 - **cl (clignoter)** pour faire clignoter une led ;
- **pa** : les paramètres de l'action. Ils dépendent de l'action.

- l'Arduino renvoie **systématiquement** une réponse à son client. Celle-ci est une chaîne JSON de la forme suivante :

```
{"id":1,"er":0,"et":{"pinx":valx}}
```

où

- **id** : l'identifiant de la commande à laquelle on répond ;
- **er (erreur)** : un code d'erreur s'il y a eu une erreur, 0 sinon ;
- **et (état)** : un dictionnaire toujours vide sauf pour la commande de lecture **pr**. Le dictionnaire contient alors la valeur de la pin n° x demandée.

Voici des exemples destinés à clarifier les spécifications précédentes :

Faire clignoter la led n° 8 10 fois avec une période de 100 millisecondes :

Commande

```
{"id":1,"ac":"cl","pa":{"pin":8,"dur":100,"nb":10}}
```

Réponse

```
{"id":1,"er":0,"et":{}}
```

Les paramètres **pa** de la commande **cl** sont : la durée **dur** en millisecondes d'un clignotement, le nombre **nb** de clignotements, le n° **pin** de la pin de la led.

Ecrire la valeur binaire 1 sur la pin n° 7 :

Commande

```
{"id":2,"ac":"pw","pa":{"pin":7,"mod":b,"val":1}}
```

Réponse

```
{"id":2,"er":0,"et":{}}
```

Les paramètres **pa** de la commande **pw** sont : le mode **mod b** (binaire) ou **a** (analogique) de l'écriture, la valeur **val** à écrire, le n° **pin** de la pin. Pour une écriture binaire, **val** est 0 ou 1. Pour une écriture analogique, **val** est dans l'intervalle [0,255].

Ecrire la valeur analogique 120 sur la pin n° 2 :

Commande

```
{"id":3,"ac":"pw","pa":{"pin":2,"mod":a,"val":120}}
```

Réponse

```
{"id":3,"er":0,"et":{}}
```

Lire la valeur analogique de la pin 0 :

Commande

```
{"id":4,"ac":"pr","pa":{"pin":0,"mod":a}}
```

Réponse

```
{"id":4,"er":0,"et":{"pin0":1023}}
```

Les paramètres **pa** de la commande **pr** sont : le mode **mod b** (binaire) ou **a** (analogique) de la lecture, le n° **pin** de la pin. S'il n'y a pas d'erreur, l'Arduino met dans le dictionnaire "**et**" de sa réponse, la valeur de la pin demandée. Ici **pin0** indique que c'est la valeur de la pin n° 0 qui a été demandée et **1023** est cette valeur. En lecture, une valeur analogique sera dans l'intervalle [0, 1024].

Nous avons présenté les trois commandes **cl**, **pw** et **pr**. On peut se demander pourquoi on n'a pas utilisé des champs plus explicites dans les chaînes JSON, **action** au lieu de **ac**, **pinwrite** au lieu de **pw**, **paramètres** au lieu de **pa**, ... Un Arduino a une mémoire très réduite. Or les chaînes JSON échangées avec l'Arduino participent à l'occupation mémoire. On a donc choisi de raccourcir celles-ci au maximum.

Voyons maintenant quelques cas d'erreur :

Commande	xx
Réponse	{"id":"","er":"100","et":{}}

On a envoyé une commande qui n'est pas au format JSON. L'Arduino a renvoyé le code d'erreur 100.

Commande	{"id": "4", "ac": "pr", "pa": {"mod": "a"}}
Réponse	{"id": "4", "er": "302", "et": {}}

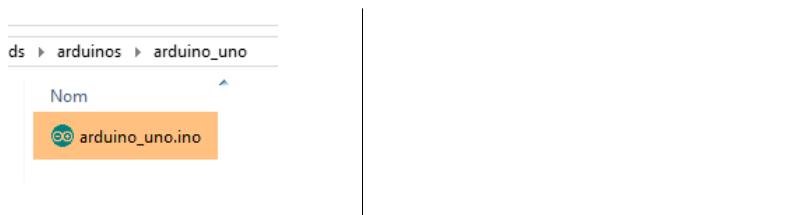
On a envoyé une commande **pr** en oubliant le paramètre **pin**. L'Arduino a renvoyé le code d'erreur 302.

Commande	{"id": "4", "ac": "pinread", "pa": {"pin": "0", "mod": "a"}}
Réponse	{"id": "4", "er": "104", "et": {}}

On a envoyé une commande **pinread** inconnue (c'est pr). L'Arduino a renvoyé le code d'erreur 104.

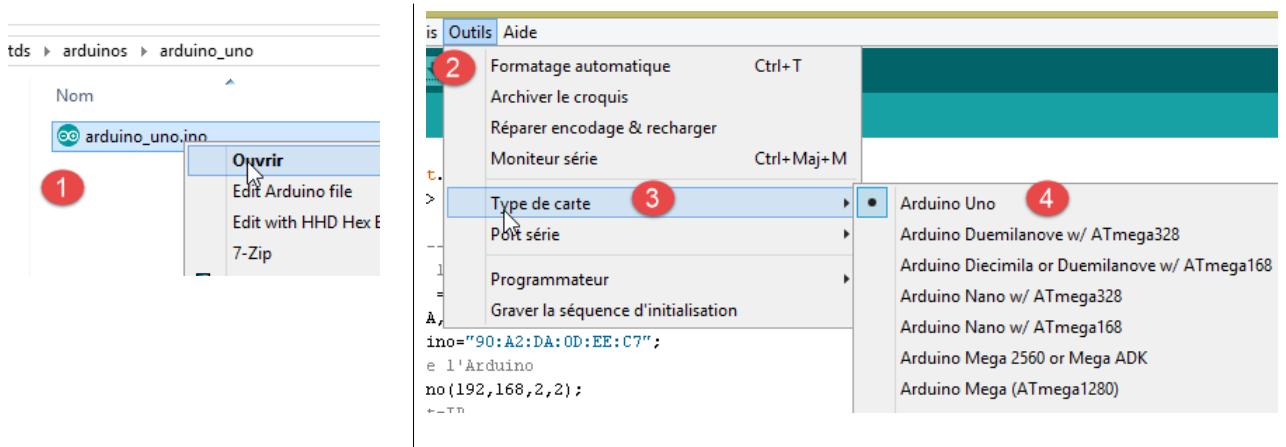
On ne continuera pas les exemples. La règle est simple. L'Arduino **ne doit pas planter**, quelque soit la commande qu'on lui envoie. Avant d'exécuter une commande JSON, il s'assure que celle-ci est correcte. Dès qu'une erreur apparaît, l'Arduino arrête l'exécution de la commande et renvoie à son client la chaîne JSON d'erreur. Là encore, parce qu'on est contraint en espace mémoire, on renvoie un code d'erreur plutôt qu'un message complet.

Le code du programme exécuté sur l'Arduino vous est fourni dans les exemples de ce document :



Pour le transférer sur l'Arduino :

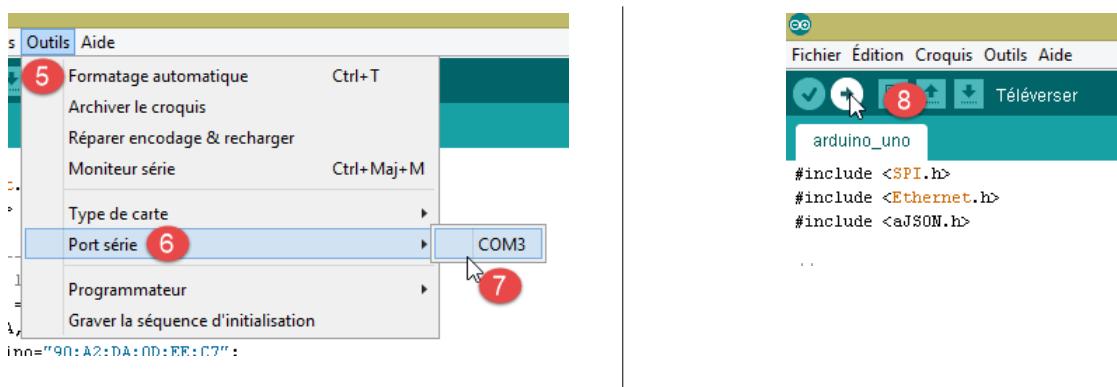
- connectez celui-ci à votre PC ;



- en [1], ouvrez le fichier [arduino_arduino.ino]. L'IDE Arduino va se lancer et charger le fichier ;

Note : le code a été créé et testé originellement avec un IDE ARDUINO 1.5.x. Depuis d'autres versions de l'IDE sont sorties. **Le code n'a pas fonctionné avec un IDE ARDUINO 1.6.x.** Il semble qu'il y ait un problème de compatibilité arrière entre les versions 1.6 et 1.5.

- en [2-4], indiquez le type d'Arduino utilisé ;



- en [5-7], indiquez sur quel port série du PC il se trouve ;
- en [8], téléversez (=chargez) le programme [arduino_arduino] sur l'Arduino ;

Le code du programme est très commenté. Le lecteur intéressé pourra s'y référer. Nous signalons simplement les lignes du code qui permettent de configurer la communication bidirectionnelle client / serveur entre l'Arduino et le PC :

```

1. #include <SPI.h>
2. #include <Ethernet.h>
3. #include <aJSON.h>
4.
5. // ----- CONFIGURATION DE L'ARDUINO UNO
6. // adresse MAC de l'Arduino UNO
7. byte macArduino[] = {
8.   0x90, 0xA2, 0xDA, 0x0D, 0xEE, 0xC7 };
9. char * strMacArduino="90:A2:DA:0D:EE:C7";
10. // l'adresse IP de l'Arduino
11. IPAddress ipArduino(192,168,2,2);
12. // son identifiant
13. char * idArduino="cuisine";
14. // port du serveur Arduino
15. int portArduino=102;
16. // description de l'Arduino
17. char * descriptionArduino="contrôle domotique";
18. // le serveur Arduino travaillera sur le port 102
19. EthernetServer server(portArduino);
20. // IP du serveur d'enregistrement
21. IPAddress ipServeurEnregistrement(192,168,2,1);
22. // port du serveur d'enregistrement
23. int portServeurEnregistrement=100;
24. // le client Arduino du serveur d'enregistrement

```

```

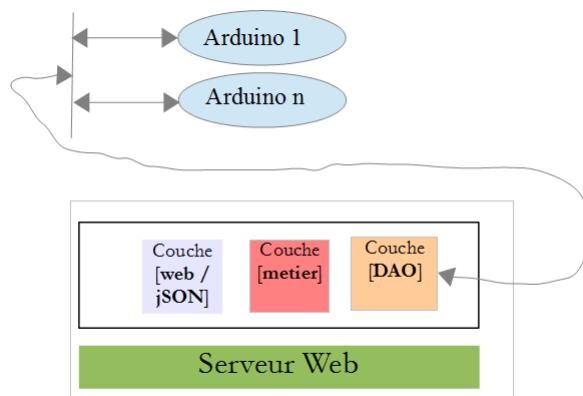
25. EthernetClient clientArduino;
26. // la commande du client
27. char commande[100];
28. // la réponse de l'Arduino
29. char message[100];
30.
31. // initialisation
32. void setup() {
33.   // Le moniteur série permettra de suivre les échanges
34.   Serial.begin(9600);
35.   // démarrage de la connection Ethernet
36.   Ethernet.begin(macArduino,ipArduino);
37.   // mémoire disponible
38.   Serial.print(F("Mémoire disponible : "));
39.   Serial.println(freeRam());
40. }
41.
42. // boucle infinie
43. void loop()
44. {
45.   ...
46. }

```

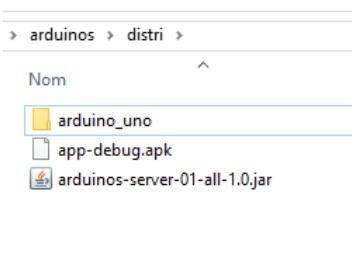
- ligne 8 : l'adresse Mac de l'arduino. Elle n'a pas beaucoup d'importance ici car l'Arduino va être sur un réseau privé où il y a un PC et un ou plusieurs Arduinos. Il faut simplement que l'adresse Mac soit unique sur ce réseau privé. Normalement, la carte réseau de l'Arduino a un sticker où est indiquée l'adresse Mac de la carte. Si ce sticker est absent et si vous ne connaissez pas l'adresse Mac de la carte, vous pouvez mettre ce que vous voulez en ligne 8 tant que la règle d'unicité de l'adresse Mac sur le réseau privé est respectée ;
- ligne 11 : l'adresse IP de la carte. De nouveau, on met ce qu'on veut du type [192.168.2.x] et on fait varier x pour les différents Arduinos du réseau privé ;
- ligne 13 : identifiant de l'Arduino. Doit être unique parmi les identifiants des Arduinos d'un même réseau privé ;
- ligne 15 : le port de service de l'Arduino. On peut mettre ce qu'on veut ;
- ligne 17 : la description de la fonction de l'Arduino. On peut mettre ce qu'on veut. Attention aux longues chaînes à cause de la mémoire restreinte de l'Arduino ;
- ligne 21 : adresse IP du serveur d'enregistrement de l'Arduino sur le PC. **Ne doit pas être modifié** ;
- ligne 23 : port de ce service d'enregistrement. **Ne doit pas être modifié** ;

5.4 Le serveur web / JSON

5.4.1 Installation



Le binaire Java du serveur web / JSON vous est donné :



Placez les deux fichiers [server-arduinoss.*] sur le bureau et exécutez le fichier [server-arduinoss.bat] en double-cliquant dessus. Une autre façon de faire est d'ouvrir une fenêtre de commandes et de taper la commande qui est dans le fichier [server-arduinoss.bat] :

```
dos>java -jar arduinos-server-01-all-1.0.jar
```

Si `[java.exe]` n'est pas dans le PATH de la fenêtre de commandes, il sera nécessaire de taper le chemin complet de `[java.exe]` (en général `c:\Program Files\java\...`).

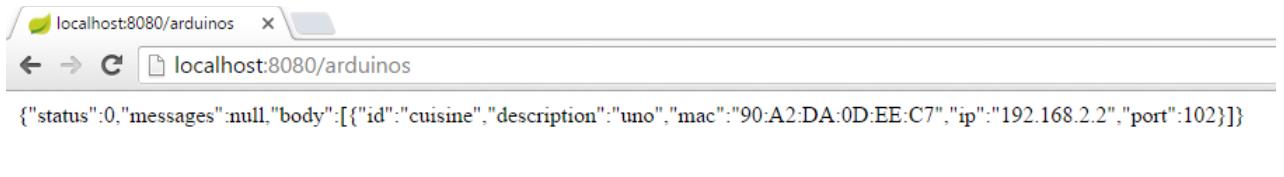
Une fenêtre DOS va s'ouvrir et afficher des logs :

```
1. . ' \ \ / _ _ _ _ _ ( ) _ _ _ \ \ \ \ \
2. ( ( ) \ _ [ ] _ [ ] _ [ ] _ [ ] _ [ ] _ [ ] )
3. \ \ / [ ] [ ] [ ] [ ] [ ] [ ] [ ] )
4. ' [ ] [ ] [ ] [ ] [ ] [ ] [ ] )
5. ======|_=====|/_=/_/_/
6. :: Spring Boot ::          (v0.5.0.M6)
7.
8.
9. 2014-01-06 11:11:35.550 INFO 8408 --- [           main] arduino.rest.metier.Application      : Starting Application
on Gportpers3 with PID 8408 (C:\Users\SergeTah\u\Desktop\part2\server.jar started by ST)
10. 2014-01-06 11:11:35.587 INFO 8408 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@6a4ba620: startup date [Mon Jan
06 11:11:35 CET 2014]; root of context hierarchy
11. 2014-01-06 11:11:36.765 INFO 8408 --- [           main] o.apache.catalina.core.StandardService   : Starting service
Tomcat
12. 2014-01-06 11:11:36.766 INFO 8408 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet
Engine: Apache Tomcat/7.0.42
13. 2014-01-06 11:11:36.876 INFO 8408 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[]/        : Initializing Spring
embedded WebApplicationContext
14. 2014-01-06 11:11:36.877 INFO 8408 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root
WebApplicationContext: initialization completed in 1293 ms
15. 2014-01-06 11:11:37.084 INFO 8408 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[]/        : Initializing Spring
FrameworkServlet 'dispatcherServlet'
16. 2014-01-06 11:11:37.084 INFO 8408 --- [ost-startStop-1] o.s.web.servlet.DispatcherServlet       : FrameworkServlet
'dispatcherServlet': initialization started
17. 2014-01-06 11:11:37.184 INFO 8408 --- [ost-startStop-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path
[/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
18. 2014-01-06 11:11:37.386 INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"[[/arduinoblink/{idCommande}/{idArduino}/{pin}/{duree}]/
{nombre}],methods=[GET],params=[],headers[],consumes[],produces[],custom[]]" onto public java.lang.String
arduino.rest.metier.RestMetier.faireClignoterLed(java.lang.String,java.lang.String,java.lang.String,java.lang.String,java.lang.String,javax.servlet.http.HttpServletResponse)
19. 2014-01-06 11:11:37.388 INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"[[/arduinocommandes/{idArduino}],methods=[POST],params[],headers[],consumes[],produces[],custom[]]" onto public
java.lang.String
arduino.rest.metier.RestMetier.sendCommandesJson(java.lang.String,java.lang.String,javax.servlet.http.HttpServletResponse)
20. 2014-01-06 11:11:37.388 INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"[[/arduinobus],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public java.lang.String
arduino.rest.metier.RestMetier.getArduinos(javax.servlet.http.HttpServletResponse)
21. 2014-01-06 11:11:37.389 INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"[[/arduinobus/pinRead/{idCommande}/{idArduino}/{pin}]/
{mode}],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public java.lang.String
arduino.rest.metier.RestMetier.pinRead(java.lang.String,java.lang.String,java.lang.String,java.lang.String,java.lang.String,javax.servlet.http.HttpServletResponse)
22. 2014-01-06 11:11:37.390 INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"[[/arduinobus/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}]/
{valeur}],methods=[GET],params[],headers[],consumes[],produces[],custom[]]" onto public java.lang.String
arduino.rest.metier.RestMetier.pinWrite(java.lang.String,java.lang.String,java.lang.String,java.lang.String,java.lang.String,javax.servlet.http.HttpServletResponse)
23. 2014-01-06 11:11:37.463 INFO 8408 --- [ost-startStop-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path
[/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
24. 2014-01-06 11:11:37.464 INFO 8408 --- [ost-startStop-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path
[/{webapps/**}] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
25. 2014-01-06 11:11:37.881 INFO 8408 --- [ost-startStop-1] o.s.web.servlet.DispatcherServlet       : FrameworkServlet
'dispatcherServlet': initialization completed in 796 ms
26. Serveur d'enregistrement lanc\u sur 192.168.2.1:100
27. 2014-01-06 11:11:38.101 INFO 8408 --- [ Thread-4] arduino.dao.Recorder
[11:11:38:101] : [Serveur d'enregistrement : attente d'un client] : Recorder :
```

```
28. 2014-01-06 11:11:38.142 INFO 8408 --- [           main] arduino.rest.metier.Application : Started Application in 3.257 seconds
```

- ligne 11 : un serveur Tomcat embarqué est lancé ;
- ligne 15 : la servlet [dispatcherServlet] de Spring MVC est chargée et exécutée ;
- ligne 18 : l'URL Rest [/arduinoss/blink/{idCommande}/{idArduino}/{pin}/{duree}/{nombre}] est détectée ;
- ligne 19 : l'URL Rest [/arduinoss/commands/{idArduino}] est détectée ;
- ligne 20 : l'URL Rest [/arduinoss/] est détectée ;
- ligne 21 : l'URL Rest [/arduinoss/pinRead/{idCommande}/{idArduino}/{pin}/{mode}] est détectée ;
- ligne 22 : l'URL Rest [/arduinoss/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}/{valeur}] est détectée ;
- ligne 26 : le serveur d'enregistrement des Arduinos est lancé ;

Connectez votre Arduino au PC si ce n'est déjà fait. **Le pare-feu du PC doit être désactivé.** Puis avec un navigateur demandez l'URL [<http://localhost:8080/arduinoss>] :



Vous devez voir apparaître l'identifiant de l'Arduino connecté. Si vous n'avez rien, pensez à resetter l'Arduino. Il a un bouton poussoir pour cela.

Le serveur web / JSON est désormais installé.

5.4.2 Les URL exposées par le service web / JSON

A lire : projet [Exemple-15] (cf paragraphe 1.16.1, page 157) ;

Le service web / JSON a été implémenté avec Spring MVC et expose les URL suivantes :

```
1. @Controller
2. public class WebController {
3.
4.     // couche métier
5.     @Autowired
6.     private IMetier métier;
7.
8.     // liste des arduinos
9.     @RequestMapping(value = "/arduinoss", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
10.    @ResponseBody
11.    public String getArduinos() throws JsonProcessingException {
12.        ...
13.    }
14.
15.    // clignotement
16.    @RequestMapping(value = "/arduinoss/blink/{idCommande}/{idArduino}/{pin}/{duree}/{nombre}", method = RequestMethod.GET,
produces = MediaType.APPLICATION_JSON_VALUE)
17.    @ResponseBody
18.    public String faireClignoterLed(@PathVariable("idCommande") String idCommande, @PathVariable("idArduino") String
idArduino, @PathVariable("pin") int pin, @PathVariable("duree") int duree, @PathVariable("nombre") int nombre) throws
JsonProcessingException {
19.        ...
20.    }
21.
22.    // envoi de commandes JSON
23.    @RequestMapping(value = "/arduinoss/commands/{idArduino}", method = RequestMethod.POST, produces =
MediaType.APPLICATION_JSON_VALUE, consumes = MediaType.APPLICATION_JSON_VALUE)
24.    @ResponseBody
25.    public String sendCommandesJson(@PathVariable("idArduino") String idArduino, HttpServletRequest request) throws
IOException {
26.        ...
27.    }
28.
29.    // lecture pin
30.    @RequestMapping(value = "/arduinoss/pinRead/{idCommande}/{idArduino}/{pin}/{mode}", method = RequestMethod.GET, produces =
MediaType.APPLICATION_JSON_VALUE)
31.    @ResponseBody
32.    public String pinRead(@PathVariable("idCommande") String idCommande, @PathVariable("idArduino") String idArduino,
@PathVariable("pin") int pin, @PathVariable("mode") String mode) throws JsonProcessingException {
33.        ...
34.    }
```

```

35.
36.    // écriture pin
37.    @RequestMapping(value = "/arduinoss/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}/{valeur}", method = RequestMethod.GET,
38.        produces = MediaType.APPLICATION_JSON_VALUE)
39.    @ResponseBody
40.    public String pinWrite(@PathVariable("idCommande") String idCommande, @PathVariable("idArduino") String idArduino,
41.        @PathVariable("pin") int pin, @PathVariable("mode") String mode, @PathVariable("valeur") int valeur) throws
42.        JsonProcessingException {
43.        ...
44.    }
45. }
```

Les réponses envoyées par le serveur sont des représentations JSON de la classe [Response<T>] suivante :

```

1. package client.android.dao.service;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.     // ----- propriétés
8.     // statut de l'opération
9.     private int status;
10.    // les éventuels messages d'status
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.    ...
28. }
```

L'URL [/arduinoss] envoie une réponse de type [Response<List<Arduino>>] où [Arduino] est la classe suivante :

```

1. package android.arduinoss.entities;
2.
3. import java.io.Serializable;
4.
5. public class Arduino implements Serializable {
6.     // données
7.     private String id;
8.     private String description;
9.     private String mac;
10.    private String ip;
11.    private int port;
12.
13.    // getters et setters
14.    ...
15. }
```

- ligne 7 : [id] est l'identifiant de l'arduino ;
- ligne 8 : sa description ;
- ligne 9 : son adresse MAC ;
- ligne 10 : son adresse IP ;
- ligne 11 : le port sur lequel il attend des commandes ;

Les URL :

- [/arduinoss/blink/{idCommande}/{idArduino}/{pin}/{duree}/{nombre}] ;
- [/arduinoss/pinRead/{idCommande}/{idArduino}/{pin}/{mode}] ;
- [/arduinoss/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}/{valeur}] ;
- [/arduinoss/commands/{idArduino}] ;

envoient une réponse de type [Response<ArduinoResponse>] où la classe [ArduinoResponse] représente la réponse standard d'un Arduino :

```

1. public class ArduinoResponse implements Serializable {
2.
```

```

3.   private String json;
4.   private String id;
5.   private String erreur;
6.   private Map<String, Object> etat;
7.
8.   // getters et setters
9. ...
10. }

```

- [json] : la chaîne JSON envoyée par un Arduino et qui n'a pu être décodée (cas d'erreur), *null* sinon ;
- [id] : l'identifiant de la commande à laquelle l'Arduino répond ;
- [erreur] : un code d'erreur, 0 si OK, autre chose sinon ;
- [etat] : un dictionnaire contenant la réponse spécifique à la commande. Il est le plus souvent vide sauf si la commande demandait la lecture d'une valeur de l'Arduino auquel cas celle-ci sera placée dans ce dictionnaire ;

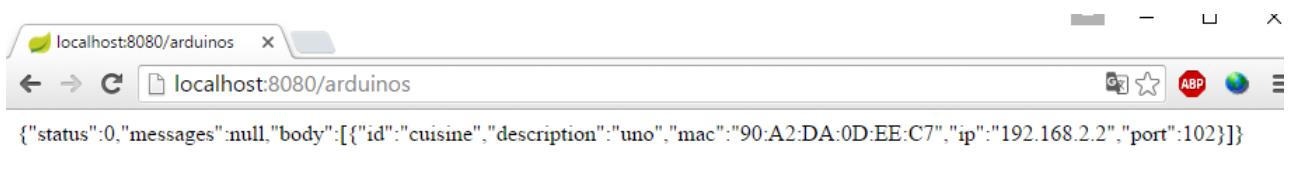
5.4.3 Les tests du service web / JSON

Habitez-vous au serveur web / JSON en testant les URL suivantes :

URL	rôle
http://localhost:8080/arduininos/	rend la liste des Arduinos connectés
http://localhost:8080/arduininos/blink/1/cuisine/8/100/20/	fait clignoter la led de la pin n° 8 de l'Arduino identifié par cuisine, 20 fois toutes les 100 ms.
http://localhost:8080/arduininos/pinRead/1/cuisine/0/a/	lecture analogique de la pin n° 0 de l'Arduino identifié par cuisine
http://localhost:8080/arduininos/pinRead/1/cuisine/5/b/	lecture binaire de la pin n° 5 de l'Arduino identifié par cuisine
http://localhost:8080/arduininos/pinWrite/1/cuisine/8/b/1/	écriture binaire de la valeur 1 sur la pin n° 8 de l'Arduino identifié par cuisine
http://localhost:8080/arduininos/pinWrite/1/cuisine/4/a/100/	écriture analogique de la valeur 100 sur la pin n° 4 de l'Arduino identifié par cuisine

Voici quelques copies d'écran de ce que vous devez obtenir :

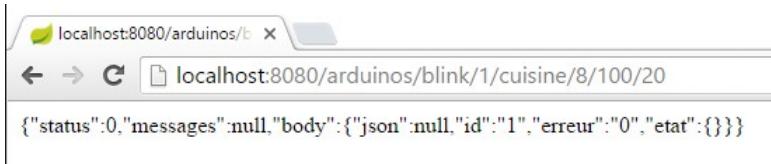
Obtenir la liste des Arduinos connectés :



La chaîne JSON reçue du serveur web / JSON est un objet avec les champs suivants :

- [status] : à 0 indique qu'il n'y a pas eu d'erreur - sinon il y a eu erreur ;
- [messages] : une liste de messages expliquant l'erreur s'il y a eu erreur ;
- [body] : la liste des Arduinos s'il n'y a pas eu d'erreur. Chaque Arduino est alors décrit par un objet avec les champs suivants :
 - [id] : identifiant de l'Arduino. Deux Arduinos ne peuvent avoir le même identifiant ;
 - [description] : courte description de la fonctionnalité de l'Arduino ;
 - [mac] : adresse Mac de l'Arduino ;
 - [ip] : adresse IP de l'Arduino ;
 - [port] : port sur lequel il attend des commandes ;

Faire clignoter la led de la pin n° 8 de l'Arduino identifié par [cuisine], 20 fois toutes les 100 ms :



```
{"status":0,"messages":null,"body":{"json":null,"id":"1","erreur":"0","etat":{}}}
```

La chaîne JSON reçue du serveur web / JSON est un objet avec les champs suivants :

- [status] : à 0 indique qu'il n'y a pas eu d'erreur - sinon il y a eu erreur ;
- [messages] : une liste de messages expliquant l'erreur s'il y a eu erreur :
- [body] : la réponse de l'Arduino s'il n'y a pas eu erreur :
 - [id] : identifiant de la commande. Cet identifiant est le 1 dans [/blink/1]. L'Arduino reprend cet identifiant de commande dans sa réponse ;
 - [erreur] : un n° d'erreur. Une valeur différente de 0 signale une erreur ;
 - [etat] : n'est utilisé que pour la lecture d'une pin. A alors pour valeur, la valeur de la pin ;
 - [json] : n'est utilisé qu'en cas d'erreur JSON entre le client et le serveur. A alors pour valeur, la chaîne JSON erronée envoyée par l'Arduino ;

Lecture analogique de la pin n° 0 de l'Arduino identifié par [cuisine] :



```
{"status":0,"messages":null,"body":{"json":null,"id":"1","erreur":"0","etat":{"pin0":481}}}
```

La chaîne JSON reçue du serveur web / JSON est analogue à la précédente près du champ [etat] qui représente la valeur de la pin n° 0.

Lecture binaire de la pin n° 5 de l'Arduino identifié par [cuisine] :



```
{"status":0,"messages":null,"body":{"json":null,"id":"1","erreur":"0","etat":{"pin5":0}}}
```

La chaîne JSON reçue du serveur web / JSON est analogue à la précédente.

Ecriture binaire de la valeur 1 sur la pin n° 8 de l'Arduino identifié par [chambre-13] :



```
{"status":0,"messages":null,"body":{"json":null,"id":"1","erreur":"0","etat":{}}}
```

La chaîne JSON reçue du serveur web / JSON est analogue à la précédente.

Le test de l'URL [<http://localhost:8080/arduinob/commands/cuisine>] est plus délicat. La méthode du serveur web / JSON qui traite cette URL attend une requête POST qu'on ne peut pas simuler simplement avec un navigateur. Pour tester cette URL, on pourra utiliser un navigateur Chrome avec l'extension [Advanced REST Client] (cf paragraphe 6.13, page 501) :

The screenshot shows a POST request configuration:

- URL:** http://localhost:8080/arduino/commands/cuisine (1)
- Method:** POST (2)
- Content-Type:** application/json (4)
- JSON Payload:** [{"id": "1", "pa": {"nb": "10", "dur": "100", "pin": "8"}, "ac": "cl"}] (5)
- Headers:** application/json (3)
- Buttons:** SEND (6)

- en [1], l'URL de la méthode web / JSON à tester ;
- en [2], la méthode POST pour envoyer la requête ;
- en [3-4], la valeur postée est du JSON ;
- en [5], la chaîne JSON postée. On notera bien les crochets qui commencent et fermentent la liste. Ici, dans la liste il n'y a qu'une commande JSON qui fait clignoter la pin n° 8, 10 fois toutes les 100 ms ;
- en [6], on envoie la requête ;

The screenshot shows a JSON response:

```
Raw
[{"status":0,"messages":null,"body":[{"json":null,"id":"1","erreur":"0","etat":{}}]}
```

Number 7 is highlighted over the JSON object.

- en [7], la réponse JSON envoyée par le serveur. L'objet a reçu un objet avec les deux champs habituels [status, messages] et un champ [body] dont la valeur est la liste des réponses de l'Arduino à chacune des commandes JSON envoyées.

Voyons ce qui se passe lorsqu'on envoie une commande JSON syntaxiquement incorrecte pour l'Arduino :

The screenshot shows a JSON response with a syntax error in the payload:

```
Raw payload
[{"id": "1", "pa": {"nb": "10", "dur": "100", "pin": "8"}, "ac": "xx"}]
```

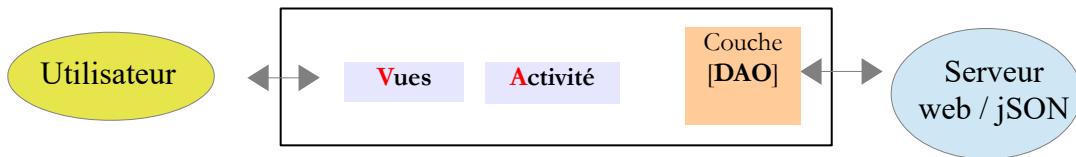
Number 1 is highlighted over the invalid JSON value "xx".

On reçoit alors la réponse suivante :

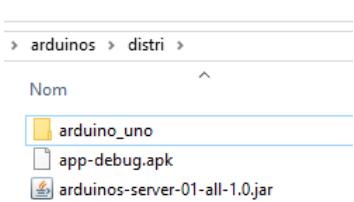
```
Raw
{"status":0,"messages":null,"body":[{"json":null,"id":"1","erreur":"104","etat":{}}]}
```

On voit que dans la réponse de l'Arduino, le n° d'erreur est [104] indiquant par là que la commande [xx] n'a pas été reconnue.

5.5 Tests du client Android



Le binaire exécutable du client Android terminé vous est donné :



Avec la souris déposez le binaire [app-debug.apk] ci-dessus sur un émulateur de tablette [GenyMotion]. Il va alors être enregistré puis exécuté. Lancez également le serveur web / jSON si ce n'est déjà fait. Connectez l'Arduino au PC avec une led dessus. Le client Android permet de gérer les Arduinos à distance. Il présente à l'utilisateur les écrans suivants.

L'onglet [CONFIG] permet de se connecter au serveur et de récupérer la liste des Arduinos connectés :

Liste des Arduinos connectés	
Id:cuisine	Description:uno

- en [1], mettez l'adresse IP [192.168.2.1] donnée à votre PC (cf paragraphe 5.2, page 414).

L'onglet [PINWRITE] permet d'écrire une valeur sur une pin d'un Arduino :

5554:WXGA_Tablet_API_23

android-domotique

[CONFIG] [BLINK] [PINREAD] **[PINWRITE]** [COMMANDS]

Modifier la valeur d'une pin

Liste des Arduinos connectés

<input checked="" type="checkbox"/>	Id: cuisine	Description: uno
-------------------------------------	--------------------	------------------

Commande

Numéro de pin 8

Mode Analogique Binaire

Valeur à écrire

EXÉCUTER

Réponses des Arduinos

```
{"erreur": "0", "etat": {}, "id": "0", "json": null}
```

◀ ○ □

5554:WXGA_Tablet_API_23

android-domotique

[CONFIG] [BLINK] [PINREAD] **[PINWRITE]** [COMMANDS]

Modifier la valeur d'une pin

Liste des Arduinos connectés

<input checked="" type="checkbox"/>	Id: cuisine	Description: uno
-------------------------------------	--------------------	------------------

Commande

Numéro de pin 3

Mode Analogique Binaire

Valeur à écrire

EXÉCUTER

Réponses des Arduinos

```
{"erreur": "0", "etat": {}, "id": "0", "json": null}
```

◀ ○ □

L'onglet [PINREAD] permet de lire la valeur d'une pin d'un Arduino :

5554:WXGA_Tablet_API_23

android-domotique

[CONFIG] [BLINK] [PINREAD] [PINWRITE] [COMMANDS]

Lire la valeur d'une pin

Liste des Arduinos connectés

<input checked="" type="checkbox"/>	Id: cuisine	Description: uno
-------------------------------------	--------------------	-------------------------

Commande

Numéro de pin 0

Mode de lecture Analogique Binaire

EXÉCUTER

Réponses des Arduinos

```
{"erreur": "0", "etat": {"pin0": "242"}, "id": "0", "json": null}
```

◀ O □

L'onglet [BLINK] permet de faire clignoter une led d'un Arduino :

5554:WXGA_Tablet_API_23

android-domotique

[CONFIG] [BLINK] [PINREAD] [PINWRITE] [COMMANDS]

Faire clignoter une Led

Liste des Arduinos connectés

<input checked="" type="checkbox"/>	Id: cuisine	Description: uno
-------------------------------------	--------------------	-------------------------

Commande

Numéro de pin 8

Nombre de clignotements 20

Durée du clignotement en millisecondes 100

EXÉCUTER

Réponses des Arduinos

```
{"erreur": "0", "etat": {}, "id": "0", "json": null}
```

◀ O □

L'onglet [COMMAND] permet d'envoyer une commande JSON à un Arduino :

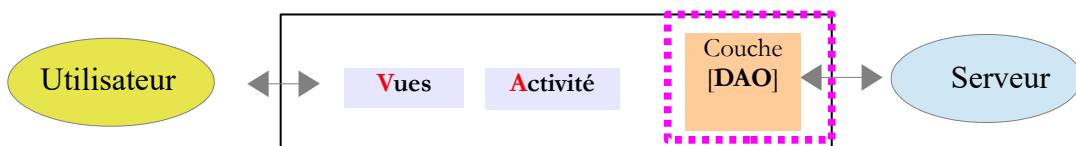


5.6 Le client Android du service web / JSON

Nous abordons maintenant l'écriture du client Android.

5.6.1 L'architecture du client

L'architecture du client Android sera celle du projet [Exemple-15] (cf paragraphe 1.16.2, page 178) ;

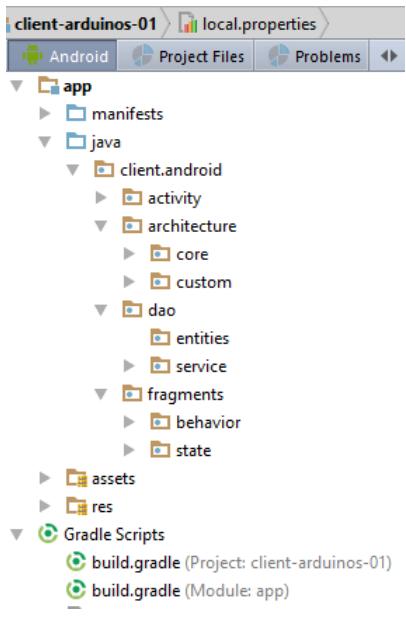


- la couche [DAO] communique avec le serveur web / JSON ;

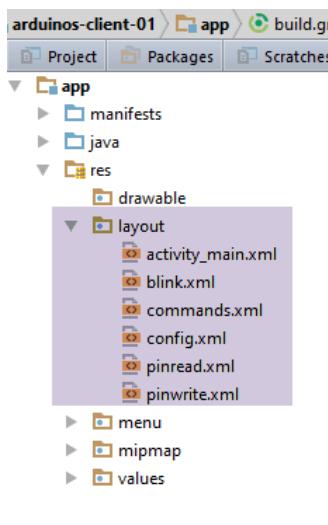
Le client Android doit pouvoir commander plusieurs Arduinos simultanément. Par exemple, on veut pouvoir faire clignoter deux leds placées sur deux Arduinos, en même temps et non pas l'une après l'autre. Aussi notre client Android utilisera-t-il une tâche asynchrone par Arduino et ces tâches s'exécuteront en parallèle.

5.6.2 Le projet Android Studio du client

Dupliquez le projet [client-android-skel] (cf paragraphe 2, page 251) dans le projet [client-arduinoss-01] (si besoin est, revoyez comment dupliquer un projet Gradle au paragraphe 1.15, page 144) :



5.6.3 Les cinq vues XML



Il y aura cinq vues XML :

- [blink] : pour faire clignoter une led d'un Arduino. Elle est associée au fragment [BlinkFragment] ;
- [commands] : pour envoyer une commande JSON à un Arduino. Elle est associée au fragment [CommandsFragment] ;
- [config] : pour configurer l'URL du service web / JSON et obtenir la liste initiale des Arduinos connectés. Elle est associée au fragment [ConfigFragment] ;
- [pinread] : pour lire la valeur binaire ou analogique d'une pin d'un Arduino. Elle est associée au fragment [PinReadFragment] ;
- [pinwrite] : pour écrire une valeur binaire ou analogique sur une pin d'un Arduino. Elle est associée au fragment [PinWriteFragment] ;

Pour le moment, ces cinq vues XML auront toutes le même contenu vide :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:id="@+id/scrollView1"
4.     android:layout_width="wrap_content"
5.     android:layout_height="wrap_content">
6.
7.     <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
8.         android:layout_width="match_parent"
9.         android:layout_height="match_parent">
10.    </RelativeLayout>
```

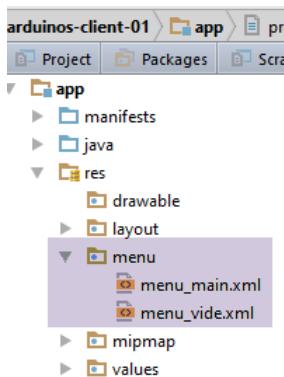
11. </ScrollView>

- la vue est dans un conteneur [RelativeLayout] (lignes 7-10) lui-même inclus dans un conteneur [ScrollView] (lignes 2-11). Cela nous assure de pouvoir 'scroller' la vue si celle-ci dépasse la taille d'un écran de tablette ;

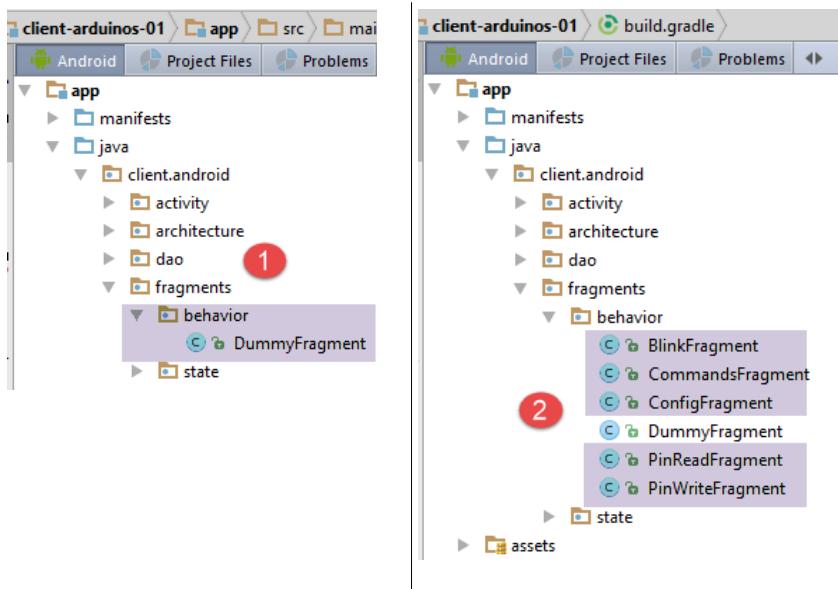
Travail : créez les cinq vues XML.

5.6.4 Le menu des fragments

Nous savons que les fragments d'un projet construit avec [client-android-skel] doivent être associés avec un menu, même vide. Ici, l'application n'aura pas de menu. Le menu vide est déjà dans le projet ;



5.6.5 Les cinq fragments de l'application



Travail : dupliquez le fragment [DummyFragment] dans les cinq fragments de l'application, comme montré en [2].

Le fragment [ConfigFragment] a le squelette suivant :

```
1. package client.android.fragments.behavior;
2.
3. import client.android.R;
4. import client.android.architecture.core.AbstractFragment;
5. import client.android.architecture.custom.CoreState;
6. import client.android.fragments.state.DummyFragmentState;
7. import org.androidannotations.annotations.EFragment;
8. import org.androidannotations.annotations.OptionsMenu;
9.
```

```

10. @EFragment
11. @OptionsMenu(R.menu.menu_vide)
12. public class ConfigFragment extends AbstractFragment {
13.
14. // champs hérités de la classe parent -----
15. ...

```

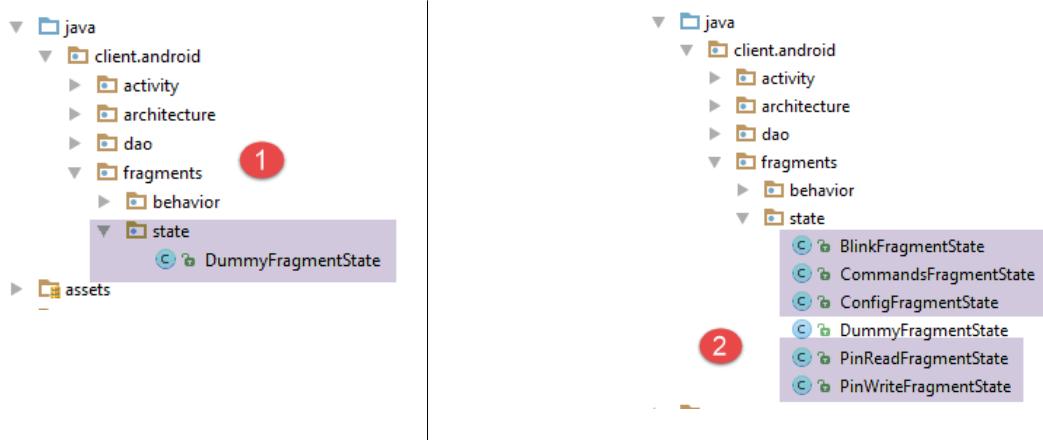
Remplacez la ligne 10 par la ligne suivante :

```
@EFragment(R.layout.config)
```

Travail : faites de même pour les quatre autres fragments en adaptant l'attribut [`@EFragment`] de la classe.

Fragment	Vue
ConfigFragment	R.layout.config
PinReadFragment	R.layout.pinread
PinWriteFragment	R.layout.pinwrite
CommandsFragment	R.layout.commands
BlinkFragment	R.layout.blink

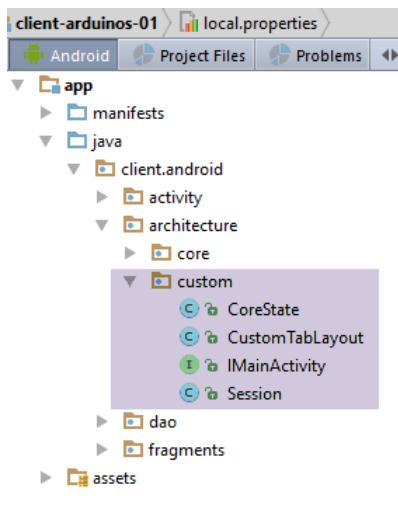
5.6.6 Les états des fragments



Chaque fragment aura un état.

Travail : dupliquez la classe [DummyFragmentState] cinq fois, pour créer les cinq états présentés en [2].

5.6.7 Personnalisation du projet



Le package [architecture / custom] contient les éléments personnalisables de l'architecture de l'application.

5.6.7.1 L'interface [IMainActivity]

L'interface [IMainActivity] définit ce que peuvent demander les fragments à l'activité ainsi que les constantes de l'application. Cette interface sera ici la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.ISession;
4. import client.android.dao.service.IDao;
5.
6. public interface IMainActivity extends IDao {
7.
8.     // accès à la session
9.     ISession getSession();
10.
11.    // changement de vue
12.    void navigateToView(int position, ISession.Action action);
13.
14.    // gestion de l'attente
15.    void beginWaiting();
16.
17.    void cancelWaiting();
18.
19.    // constantes de l'application -----
20.
21.    // mode debug
22.    boolean IS_DEBUG_ENABLED = true;
23.
24.    // délai maximal d'attente de la réponse du serveur
25.    int TIMEOUT = 1000;
26.
27.    // délai d'attente avant exécution de la requête client
28.    int DELAY = 000;
29.
30.    // authentification basique
31.    boolean IS_BASIC_AUTHENTICATION_NEEDED = false;
32.
33.    // adjacence des fragments
34.    int OFF_SCREEN_PAGE_LIMIT = 1;
35.
36.    // barre d'onglets
37.    boolean ARE_TABS_NEEDED = true;
38.
39.    // image d'attente
40.    boolean IS_WAITING_ICON_NEEDED = true;
41.
42.    // nombre de fragments
43.    int FRAGMENTS_COUNT = 5;
44.
45.    // n°s de vue
46.    int VUE_CONFIG = 0;
47.    int VUE_BLINK = 1;
48.    int VUE_PINREAD = 2;
49.    int VUE_PINWRITE = 3;
50.    int VUE_COMMANDS = 4;
```

51. }

- lignes 25, 28, 31, 40 : configuration de la couche [DAO]. Cette application interroge un serveur web / JSON ;
- ligne 37 : cette application a des onglets ;
- ligne 43 : cette application a cinq fragments ;
- lignes 46-50 : les n°s des cinq fragments ;
- ligne 34 : adjacence des fragments. Le développeur peut mettre ici une valeur dans l'intervalle [1,FRAGMENTS_COUNT-1] ;

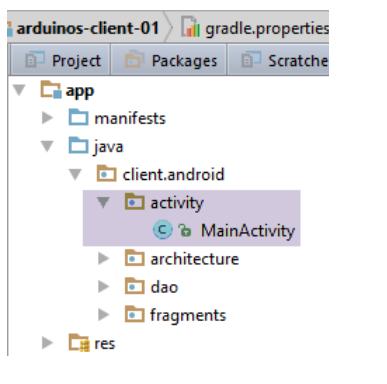
5.6.7.2 La classe [CoreState]

La classe [CoreState] est la classe parent des états des fragments :

```
1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.MenuItemState;
4. import client.android.fragments.state.*;
5. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
6. import com.fasterxml.jackson.annotation.JsonSubTypes;
7. import com.fasterxml.jackson.annotation.JsonProperty;
8.
9. @JsonIgnoreProperties(ignoreUnknown = true)
10. @JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
11. @JsonSubTypes({
12.     @JsonSubTypes.Type(value = ConfigFragmentState.class),
13.     @JsonSubTypes.Type(value = BlinkFragmentState.class),
14.     @JsonSubTypes.Type(value = PinReadFragmentState.class),
15.     @JsonSubTypes.Type(value = PinWriteFragmentState.class),
16.     @JsonSubTypes.Type(value = CommandsFragmentState.class)})
17. )
18. public class CoreState {
19.     // fragment visité ou non
20.     protected boolean hasBeenVisited = false;
21.     // état de l'éventuel menu du fragment
22.     protected MenuItemState[] menuOptionsState;
23.
24.     // getters et setters
25. ...
26. }
```

- lignes 12-16 : il faut déclarer ici les classes des états des cinq fragments ;

5.6.8 La classe [MainActivity]



La classe [MainActivity] sera la suivante :

```
1. package client.android.activity;
2.
3. import android.support.design.widget.TabLayout;
4. import android.util.Log;
5. import client.android.R;
6. import client.android.architecture.core.AbstractActivity;
7. import client.android.architecture.core.AbstractFragment;
8. import client.android.architecture.core.ISession;
9. import client.android.architecture.custom.IMainActivity;
10. import client.android.architecture.custom.Session;
11. import client.android.dao.entities.Arduino;
12. import client.android.dao.entities.ArduinoCommand;
13. import client.android.dao.entities.ArduinoResponse;
```

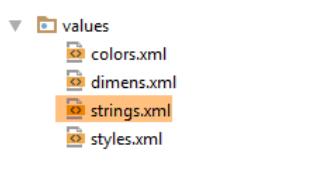
```

14. import client.android.dao.service.Dao;
15. import client.android.dao.service.IDao;
16. import client.android.dao.service.Response;
17. import client.android.fragments.behavior.*;
18. import org.androidannotations.annotations.Bean;
19. import org.androidannotations.annotations.EActivity;
20. import org.androidannotations.annotations.OptionsMenu;
21. import rx.Observable;
22.
23. import java.util.List;
24. import java.util.Locale;
25.
26. @EActivity
27. @OptionsMenu(R.menu.menu_main)
28. public class MainActivity extends AbstractActivity {
29.
30.     // couche [DAO]
31.     @Bean(Dao.class)
32.     protected IDao dao;
33.     // session
34.     private Session session;
35.
36.     // méthodes classe parent -----
37.     @Override
38.     protected void onCreateActivity() {
39.         // log
40.         if (IS_DEBUG_ENABLED) {
41.             Log.d(className, "onCreateActivity");
42.         }
43.         // session
44.         this.session = (Session) super.session;
45.         // création des cinq onglets
46.         for (int i = 0; i < 5; i++) {
47.             TabLayout.Tab newTab = tabLayout.newTab();
48.             newTab.setText(getFragmentTitle(i));
49.             tabLayout.addTab(newTab);
50.         }
51.     }
52.
53.     @Override
54.     protected IDao getDao() {
55.         return dao;
56.     }
57.
58.     @Override
59.     protected AbstractFragment[] getFragments() {
60.         return new AbstractFragment[]{new ConfigFragment_(), new BlinkFragment_(), new PinReadFragment_(), new
PinWriteFragment_(), new CommandsFragment_()};
61.     }
62.
63.     @Override
64.     protected CharSequence getFragmentTitle(int position) {
65.         Locale l = Locale.getDefault();
66.         switch (position) {
67.             case 0:
68.                 return getString(R.string.config_titre).toUpperCase(l);
69.             case 1:
70.                 return getString(R.string.blink_titre).toUpperCase(l);
71.             case 2:
72.                 return getString(R.string.pinread_titre).toUpperCase(l);
73.             case 3:
74.                 return getString(R.string.pinwrite_titre).toUpperCase(l);
75.             case 4:
76.                 return getString(R.string.commands_titre).toUpperCase(l);
77.         }
78.         return null;
79.     }
80.
81.     @Override
82.     protected void navigateOnTabSelected(int position) {
83.         // on affiche le fragment n° position
84.         navigateToView(position, IMainActivity.Action.NAVIGATION);
85.     }
86.
87.     @Override
88.     protected int getFirstView() {
89.         return IMainActivity.VUE_CONFIG;
90.     }
91.
92.     // implémentation IDao -----
93. }

```

- lignes 46-50 : création des cinq onglets de l'application ;
- ligne 48 : les titres des onglets sont fournis par la méthode des lignes 63-79 ;

- les cinq fragments sont instanciés ligne 60. A cause des annotations AA, les classes des fragments sont celles présentées précédemment suffixées avec un underscore ;
- lignes 63-79 : on définit un titre pour chacun des fragments. Ces titres seront cherchés dans le fichier [res / values / strings.xml]



Le contenu de [strings.xml] est le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.
4.     <!-- nom de l'application -->
5.     <string name="app_name">[arduinoss-client-01]</string>
6.     <!-- Fragments et onglets -->
7.     <string name="config_titre">[Config]</string>
8.     <string name="blink_titre">[Blink]</string>
9.     <string name="pinread_titre">[PinRead]</string>
10.    <string name="pinwrite_titre">[PinWrite]</string>
11.    <string name="commands_titre">[Commands]</string>
12.
13. </resources>
```

Travail : créez les éléments précédents et compilez le projet. Il ne doit pas y avoir d'erreurs.

Exécutez le projet. Vous devez obtenir la vue suivante sur la tablette :



Examinez les logs qui ont accompagné l'affichage de la première vue et suivez la trace des différentes étapes exécutées. Passez d'un onglet à l'autre et continuez à suivre les logs.

5.6.9 La vue XML [config]

La vue XML [config] sera la suivante :



La vue ci-dessus est obtenue avec le code XML suivant :

```

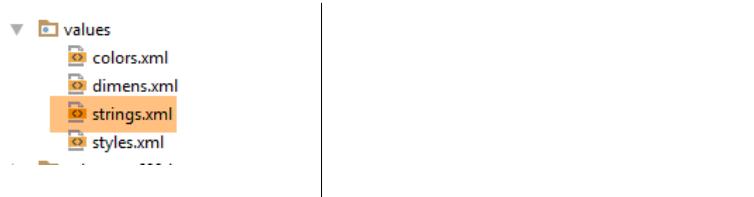
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:id="@+id/scrollView1"
4.      android:layout_width="wrap_content"
5.      android:layout_height="wrap_content">
6.
7.      <RelativeLayout
8.          android:layout_width="match_parent"
9.          android:layout_height="match_parent">
10.
11.         <TextView
12.             android:id="@+id/txt_TitreConfig"
13.             android:layout_width="wrap_content"
14.             android:layout_height="wrap_content"
15.             android:layout_alignParentTop="true"
16.             android:layout_centerHorizontal="true"
17.             android:layout_marginTop="150dp"
18.             android:text="@string/txt_TitreConfig"
19.             android:textSize="@dimen/titre"/>
20.
21.         <TextView
22.             android:id="@+id/txt_URLServiceRest"
23.             android:layout_width="wrap_content"
24.             android:layout_height="wrap_content"
25.             android:layout_alignParentLeft="true"
26.             android:layout_below="@+id/txt_TitreConfig"
27.             android:layout_marginTop="50dp"
28.             android:text="@string/txt_URLServiceRest"
29.             android:textSize="20sp"/>
30.
31.         <EditText
32.             android:id="@+id/edt_URLServiceRest"
33.             android:layout_width="300dp"
34.             android:layout_height="wrap_content"
35.             android:layout_alignBaseline="@+id/txt_URLServiceRest"
36.             android:layout_alignBottom="@+id/txt_URLServiceRest"
37.             android:layout_marginLeft="20dp"
38.             android:layout_toRightOf="@+id/txt_URLServiceRest"
39.             android:ems="10"
40.             android:hint="@string/hint_URLServiceRest"
41.             android:inputType="textUri">
42.
43.             <requestFocus/>
44.         </EditText>
45.
46.         <TextView
47.             android:id="@+id/txt_MsgErreurIpPort"
48.             android:layout_width="wrap_content"
49.             android:layout_height="wrap_content"
50.             android:layout_alignParentLeft="true"
51.             android:layout_below="@+id/txt_URLServiceRest"
52.             android:layout_marginTop="20dp"
53.             android:text="@string/txt_MsgErreurUrlServiceRest"

```

```

54.     android:textColor="@color/red"
55.     android:textSize="20sp"/>
56.
57. <TextView
58.     android:id="@+id/txt_arduinoss"
59.     android:layout_width="wrap_content"
60.     android:layout_height="wrap_content"
61.     android:layout_alignParentLeft="true"
62.     android:layout_below="@+id/txt_MsgErreurIpPort"
63.     android:layout_marginTop="40dp"
64.     android:text="@string/titre_list_arduinoss"
65.     android:textColor="@color/blue"
66.     android:textSize="20sp"/>
67.
68. <Button
69.     android:id="@+id/btn_Rafraichir"
70.     android:layout_width="wrap_content"
71.     android:layout_height="wrap_content"
72.     android:layout_alignBaseline="@+id/txt_arduinoss"
73.     android:layout_alignBottom="@+id/txt_arduinoss"
74.     android:layout_marginLeft="20dp"
75.     android:layout_toRightOf="@+id/txt_arduinoss"
76.     android:text="@string/btn Rafraichir"/>
77.
78. <Button
79.     android:id="@+id/btn_Annuler"
80.     android:layout_width="wrap_content"
81.     android:layout_height="wrap_content"
82.     android:layout_alignBaseline="@+id/txt_arduinoss"
83.     android:layout_alignBottom="@+id/txt_arduinoss"
84.     android:layout_marginLeft="20dp"
85.     android:layout_toRightOf="@+id/txt_arduinoss"
86.     android:text="@string/btn Annuler"
87.     android:visibility="invisible"/>
88.
89. <ListView
90.     android:id="@+id/ListViewArduinos"
91.     android:layout_width="match_parent"
92.     android:layout_height="200dp"
93.     android:layout_alignParentLeft="true"
94.     android:layout_below="@+id/txt_arduinoss"
95.     android:layout_marginTop="30dp"
96.     android:background="@color/wheat">
97. </ListView>
98.
99. </RelativeLayout>
100. </ScrollView>
```

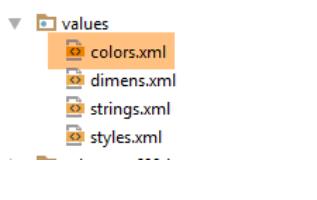
La vue utilise des chaînes de caractères (`android:text` aux lignes 15, 25, 37, 50, 61, 73) qui sont définies dans le fichier [res / values / strings] :



```

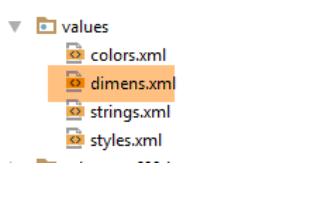
1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.
4.     <string name="app_name">android-domotique</string>
5.
6.     <!-- Fragments et onglets -->
7.     <string name="config_titre">[Config]</string>
8.     <string name="blink_titre">[Blink]</string>
9.     <string name="pinread_titre">[PinRead]</string>
10.    <string name="pinwrite_titre">[PinWrite]</string>
11.    <string name="commands_titre">[Commands]</string>
12.
13.    <!-- Config -->
14.    <string name="txt_TitreConfig">Se connecter au serveur</string>
15.    <string name="txt_UrlServiceRest">Url du service web / JSON</string>
16.    <string name="txt_MsgErreurUrlServiceRest">L'Url du service doit être entrée sous la forme
    Ip1.Ip2.Ip3.IP4:Port/contexte</string>
17.    <string name="hint_UrlServiceRest">ex (192.168.1.120:8080/rest)</string>
18.    <string name="btn_annuler">Annuler</string>
19.    <string name="btn_rafraichir">Rafraîchir</string>
20.    <string name="titre_list_arduinoss">Liste des Arduinos connectés</string>
21.
22. </resources>
```

La vue utilise des couleurs (`android:textColor` aux lignes 51 et 62] définies dans le fichier [res / values / colors] :



```
1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.   <color name="colorPrimary">#3F51B5</color>
4.   <color name="colorPrimaryDark">#303F9F</color>
5.   <color name="colorAccent">#FF4081</color>
6.   <color name="floral_white">#FFFFAF0</color>
7.   <!-- appli -->
8.   <color name="red">#FF0000</color>
9.   <color name="blue">#0000FF</color>
10.  <color name="wheat">#FFFED5</color>
11. </resources>
```

La vue utilise des dimensions (`android:textSize` à la ligne 16) qui sont définies dans le fichier [res / values / dimens] :



```
1. <resources>
2.   <!-- Default screen margins, per the Android Design guidelines. -->
3.   <dimen name="activity_horizontal_margin">16dp</dimen>
4.   <dimen name="activity_vertical_margin">16dp</dimen>
5.   <dimen name="fab_margin">16dp</dimen>
6.   <dimen name="appbar_padding_top">8dp</dimen>
7.   <!-- appli -->
8.   <dimen name="titre">30dp</dimen>
9. </resources>
```

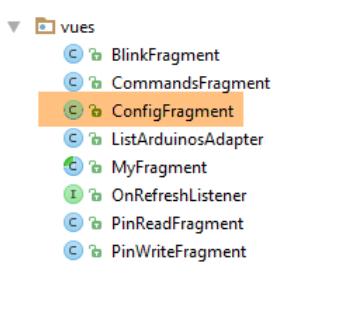
Cette technique n'a pas été utilisée pour toutes les dimensions. C'est cependant celle qui est conseillée. Elle permet de changer des dimensions en un seul endroit.

Travail : créez les éléments précédents.

Exécutez de nouveau votre projet. Vous devez obtenir la vue suivante :



5.6.10 Le fragment [ConfigFragment]



Pour gérer la nouvelle vue [config], le code du fragment [ConfigFragment] évolue de la façon suivante :

```

1. package client.android.fragments.behavior;
2.
3. import android.view.View;
4. import android.widget.Button;
5. import android.widget.EditText;
6. import android.widget.ListView;
7. import android.widget.TextView;
8. import client.android.R;
9. import client.android.architecture.core.AbstractFragment;
10. import client.android.architecture.custom.CoreState;
11. import client.android.architecture.custom.IMainActivity;
12. import client.android.fragments.state.ConfigFragmentState;
13. import org.androidannotations.annotations.Click;
14. import org.androidannotations.annotations.EFragment;
15. import org.androidannotations.annotationsOptionsMenu;
16. import org.androidannotations.annotations.ViewById;
17.
18. @EFragment(R.layout.config)
19. @OptionsMenu(R.menu.menu_vide)
20. public class ConfigFragment extends AbstractFragment {
21.
22.     // les éléments de l'interface visuelle
23.     @ViewById(R.id.btn_Rafraichir)
24.     protected Button btnRafraichir;

```

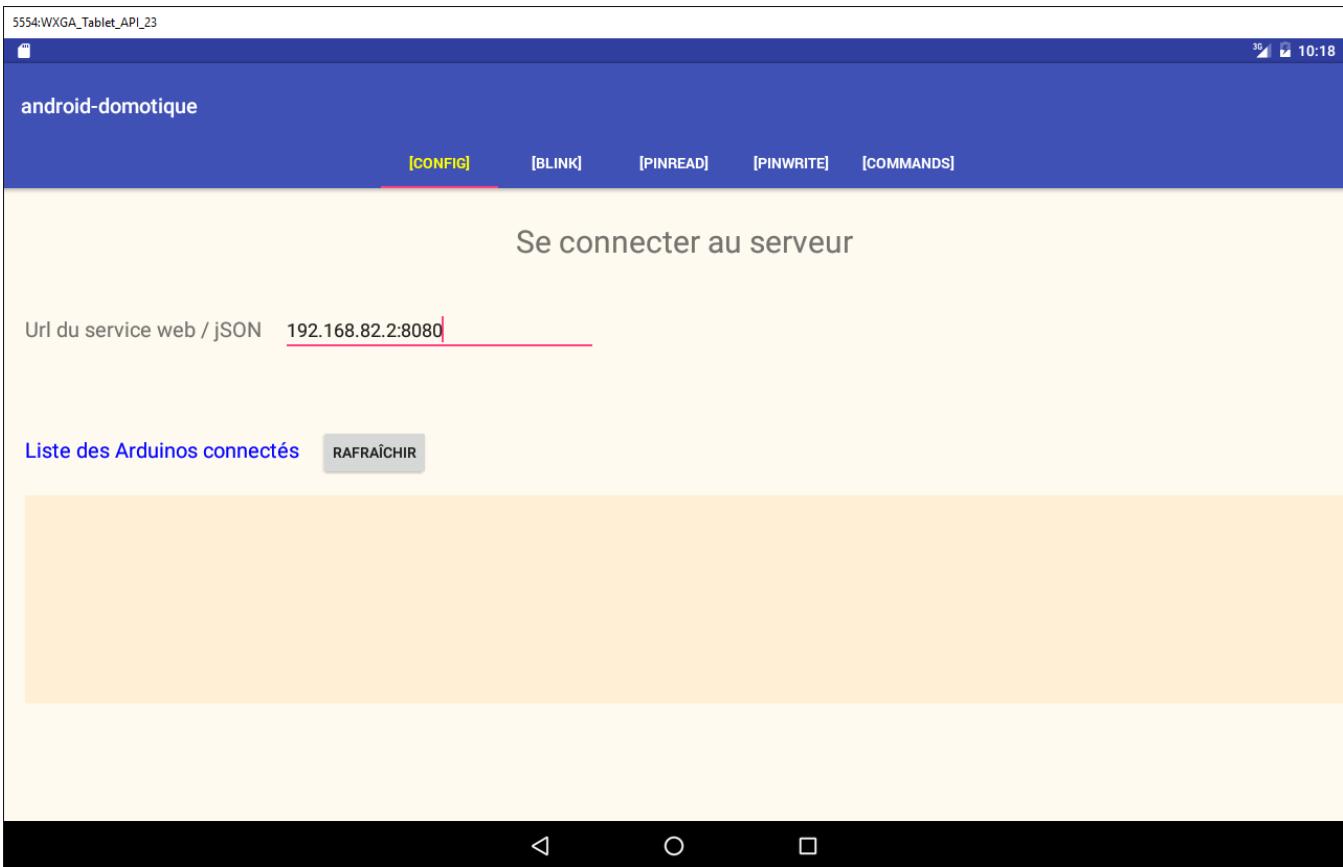
```

25.    @ViewById(R.id.btn_Annuler)
26.    protected Button btnAnnuler;
27.    @ViewById(R.id.edt_UrlServiceRest)
28.    protected EditText edtUrlServiceRest;
29.    @ViewById(R.id.txt_MsgErreurIpPort)
30.    protected TextView txtMsgErreurIpPort;
31.    @ViewById(R.id.ListViewArduinos)
32.    protected ListView listArduinos;
33.
34.    @Click(R.id.btn_Rafraichir)
35.    protected void doRafraichir() {
36.    }
37.
38.    // gestion du cycle de vie du fragment -----
39.
40.    @Override
41.    public CoreState saveFragment() {
42.        return new ConfigFragmentState();
43.    }
44.
45.    @Override
46.    protected int getNumView() {
47.        return IMainActivity.VUE_CONFIG;
48.    }
49.
50.    @Override
51.    protected void initFragment(CoreState previousState) {
52.
53.    }
54.
55.    @Override
56.    protected void initView(CoreState previousState) {
57.        // 1ère visite ?
58.        if(previousState==null){
59.            txtMsgErreurIpPort.setVisibility(View.INVISIBLE);
60.        }
61.    }
62.
63.    @Override
64.    protected void updateOnSubmit(CoreState previousState) {
65.
66.    }
67.
68.    @Override
69.    protected void updateOnRestore(CoreState previousState) {
70.    }
71.
72.    @Override
73.    protected void notifyEndOfUpdates() {
74.        // boutons
75.        initButtons();
76.    }
77.
78.    @Override
79.    protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
80.    }
81.
82.    // méthodes privées -----
83.
84.    private void initButtons() {
85.        // le bouton [Exécuter] remplace le bouton [Annuler]
86.        btnAnnuler.setVisibility(View.INVISIBLE);
87.        btnRafraichir.setVisibility(View.VISIBLE);
88.    }
89. }
```

- lignes 23-32 : les éléments de l'interface visuelle ;
- lignes 58-60 : lors de la 1ère visite faite au fragment, le message d'erreur est caché ;
- lignes 73-76 : à chaque fois que le fragment sera affiché, le bouton [Annuler] sera caché (ligne 82) et le bouton [Rafraîchir] sera affiché (lignes 86-87). En effet dans cette application, un fragment ne peut pas être affiché alors qu'une opération asynchrone est en cours et donc le bouton [Annuler] visible ;

Travail : créez les éléments précédents.

Exécutez cette nouvelle version. La 1ère vue doit être désormais la suivante :



5.6.10.1 Le bouton [Rafraîchir]

Nous allons pour le moment gérer le clic sur le bouton [Rafraîchir] de la façon suivante :

```

1.  @Click(R.id.btn_Rafraichir)
2.  protected void doRafraichir() {
3.      // on va lancer une tâche - on prépare l'attente
4.      beginWaiting(1);
5.  }
6.
7.  @Click(R.id.btn_Annuler)
8.  protected void doAnnuler() {
9.      if (isDebugEnabled) {
10.          Log.d(className, "Annulation demandée");
11.      }
12.      // on annule les tâches asynchrones
13.      cancelRunningTasks();
14.  }
15.
16. protected void beginWaiting(int numberOfRunningTasks) {
17.     // on prépare l'attente des tâches
18.     beginRunningTasks(numberOfRunningTasks);
19.     // le bouton [Annuler] remplace le bouton [Rafraîchir]
20.     btnRafraichir.setVisibility(View.INVISIBLE);
21.     btnAnnuler.setVisibility(View.VISIBLE);
22. }
23. // gestion du cycle de vie du fragment -----
24. ...
25. @Override
26. protected void notifyEndOfTasks(boolean runningTasksHaveBeenCalled) {
27.     // boutons dans leur état initial
28.     initButtons();
29. }
30.
31. // méthodes privées -----
32.
33. private void initButtons() {
34.     // le bouton [Exécuter] remplace le bouton [Annuler]
35.     btnAnnuler.setVisibility(View.INVISIBLE);
36.     btnRafraichir.setVisibility(View.VISIBLE);
37. }
```

- lignes 1-5 : la méthode exécutée lors d'un clic sur le bouton [Rafraîchir] ;
- ligne 4 : on commence l'attente ;
- ligne 18 : on passe à la classe parent le nombre de tâches asynchrones qu'on va lancer. L'image d'attente va apparaître ;
- lignes 20-21 : cette attente va se traduire par une apparition du bouton [Annuler], la disparition du bouton [Rafraîchir], l'apparition de l'image d'attente. Il ne se passe rien d'autre. L'utilisateur peut cependant cliquer sur le bouton [Annuler]. La méthode des lignes 7-14 va alors s'exécuter ;
- ligne 13 : on demande à la classe parent d'annuler toutes les tâches. La classe va le faire et appeler en retour la méthode des lignes 25-29 pour signaler que toutes les tâches sont terminées. Le paramètre [runningTasksHaveBeenCalled] aura la valeur *true* pour indiquer qu'il y a eu annulation des tâches ;
- lignes 35-36 : le bouton [Annuler] va disparaître alors que le bouton [Rafraîchir] va réapparaître.

Travail : Faites ces modifications puis exécutez le projet. Vérifiez que le bouton [Rafraîchir] lance l'attente et que le bouton [Annuler] l'arrête. Observez les logs.

5.6.10.2 Vérification des saisies

Dans la version précédente, nous ne vérifions pas la validité de l'URL saisie. Pour la vérifier, nous ajoutons le code suivant dans [ConfigFragment] :

```

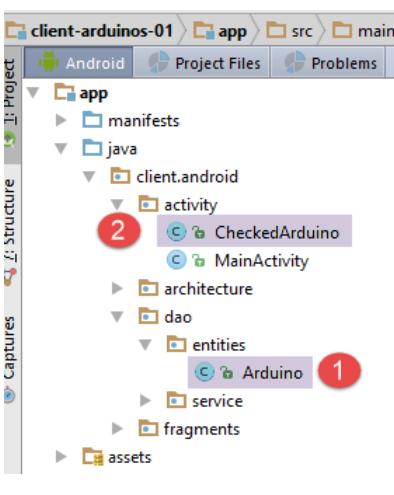
1. // les valeurs saisies
2. private String urlServiceRest;
3.
4. @Click(R.id.btn_Rafraichir)
5. protected void doRafraichir() {
6.     // on vérifie les saisies
7.     if (!pageValid()) {
8.         return;
9.     }
10.    // on va lancer une tâche - on prépare l'attente
11.    beginWaiting(1);
12. }
13.
14. // vérification des saisies
15. private boolean pageValid() {
16.     // au départ pas de msg d'erreur
17.     txtMsgErreurUrlServiceRest.setVisibility(View.INVISIBLE);
18.     // on récupère l'Ip et le port du serveur
19.     urlServiceRest = String.format("http://%", edtUrlServiceRest.getText().toString().trim());
20.     // on vérifie sa validité
21.     try {
22.         URI uri = new URI(urlServiceRest);
23.         String host = uri.getHost();
24.         int port = uri.getPort();
25.         if (host == null || port == -1) {
26.             throw new Exception();
27.         }
28.     } catch (Exception ex) {
29.         // affichage msg d'erreur
30.         txtMsgErreurUrlServiceRest.setVisibility(View.VISIBLE);
31.         // retour à l'UI
32.         return false;
33.     }
34.     // c'est bon
35.     return true;
36. }
```

- ligne 2 : l'URL saisie ;
- lignes 7-9 : avant de faire quoique ce soit, on vérifie la validité des saisies ;
- ligne 19 : on récupère l'URL saisie et on lui ajoute le préfixe [http://] ;
- ligne 22 : on essaie de construire un objet URI (**Uniform Resource Identifier**) avec. Si l'URL saisie est syntaxiquement incorrecte, on aura une exception ;
- lignes 23-27 : on crée une exception si l'URI est correcte mais qu'on a cependant [host==null] et [port== -1]. C'est un cas possible ;
- ligne 30 : on a eu une exception. On affiche le message d'erreur ;
- ligne 32 : on retourne [false] pour indiquer que la page est invalide ;
- ligne 35 : on n'a pas eu d'erreurs. On retourne [true] pour indiquer que la page est valide ;

Travail : créez les éléments précédents.

Testez cette nouvelle version et vérifiez que les URL invalides sont bien signalées.

5.6.10.3 Affichage de la liste des Arduinos



Les différentes vues vont avoir besoin d'afficher la liste des Arduinos connectés. Pour cela, nous allons définir différentes classes et une vue XML :

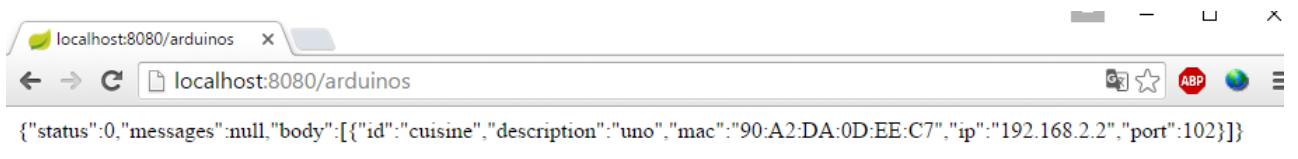
- un Arduino sera représenté par la classe [Arduino] [1] ;
- la classe [CheckedArduino] [1] est héritée de la classe [Arduino] à laquelle on a ajouté un booléen pour savoir si l'Arduino a été sélectionné ou non dans une liste ;

La classe [Arduino] est celle déjà utilisée par le serveur et présentée au paragraphe 5.4.2, page 423 C'est la suivante :

```
16. package android.arduisos.entities;
17.
18. import java.io.Serializable;
19.
20. public class Arduino implements Serializable {
21.     // données
22.     private String id;
23.     private String description;
24.     private String mac;
25.     private String ip;
26.     private int port;
27.
28.     // getters et setters
29.     ...
30. }
```

- ligne 7 : [id] est l'identifiant de l'arduino ;
- ligne 8 : sa description ;
- ligne 9 : son adresse MAC ;
- ligne 10 : son adresse IP ;
- ligne 11 : le port sur lequel il attend des commandes ;

Cette classe correspond à la chaîne JSON reçue du serveur lorsqu'on lui demande la liste des Arduinos connectés :



La classe [CheckedArduino] hérite de la classe [Arduino] :

```
1. package android.arduisos.entities;
```

```

3. public class CheckedArduino extends Arduino {
4.     private static final long serialVersionUID = 1L;
5.     // un Arduino peut être sélectionné
6.     private boolean isChecked;
7.
8.     // constructeur
9.     public CheckedArduino(Arduino arduino, boolean isChecked) {
10.         // parent
11.         super(arduino.getId(), arduino.getDescription(), arduino.getMac(), arduino.getIp(), arduino.getPort());
12.         // local
13.         this.isChecked = isChecked;
14.     }
15.
16.     // getters et setters
17.     public boolean isChecked() {
18.         return isChecked;
19.     }
20.
21.     public void setChecked(boolean isChecked) {
22.         this.isChecked = isChecked;
23.     }
24.
25. }
```

- ligne 3 : la classe [CheckedArduino] hérite de la classe [Arduino] ;
- ligne 6 : on lui ajoute un booléen qui nous servira à savoir si dans la liste des Arduinos affichée, un Arduino a été sélectionné ou non ;

Dans [ConfigFragment], nous allons simuler l'obtention de la liste des Arduinos connectés.



```

1.     @ViewById(R.id.ListViewArduinos)
2.     protected ListView listArduinos;
3.     ..
4.     @Click(R.id.btn_Rafraichir)
5.     protected void doRafraichir() {
6.         // on vérifie les saisies
7.         if (!pageValid()) {
8.             return;
9.         }
10.        // on va lancer une tâche - on prépare l'attente
11.        beginWaiting(1);
12.        // on nettoie la liste des Arduinos
13.        clearArduinos();
14.        // on demande la liste des Arduinos en tâche de fond
15.        getArduinosInBackground();
16.    }
17.
18.    private void getArduinosInBackground() {
19.        ...
20.    }
21.
22.    // raz liste des Arduinos
23.    private void clearArduinos() {
24.        // on crée une liste vide
25.        List<String> strings = new ArrayList<>();
26.        // on l'affiche
27.        listArduinos.setAdapter(new ArrayAdapter<String>(activity, android.R.layout.simple_list_item_1, android.R.id.text1,
28.            strings));
```

- ligne 2 : le *ListView* qui affiche les Arduinos connectés au serveur ;
- ligne 5 : la méthode qui demande la liste des Arduinos connectés ;
- ligne 11 : on indique à la classe parent qu'on va lancer une tâche asynchrone ;
- ligne 12 : on efface la liste des Arduinos actuellement affichée ;
- ligne 15 : on demande en tâche de fond la liste des Arduinos connectés ;
- lignes 23-28 : la méthode qui efface la liste des Arduinos actuellement affichée ;

La méthode [getArduinosInBackground] est la suivante :

```
1.  private void getArduinosInBackground() {
2.      // on crée une liste d'arduinoss fictive
3.      List<Arduino> arduinos = new ArrayList<>();
4.      for (int i = 0; i < 20; i++) {
5.          arduinos.add(new Arduino("id" + i, "desc" + i, "mac" + i, "ip" + i, i));
6.      }
7.      // on simule une réponse du serveur
8.      Response<List<Arduino>> response = new Response<>();
9.      response.setBody(arduios);
10.     // on annule l'attente
11.     cancelWaitingTasks();
12.     // on change les boutons
13.     initButtons();
14.     // on consomme la réponse
15.     consumeArduinosResponse(response);
16. }
```

- lignes 3-6 : on crée une liste de 20 Arduinos ;
- lignes 8-9 : on construit la réponse de type [Response<List<Arduino>>] (paragraphe 5, page 424) qui va encapsuler la liste des Arduinos créée ;
- ligne 11 : on annule l'attente ;
- ligne 13 : on remet les boutons dans leur état initial ;
- ligne 15 : on consomme la réponse ;

La méthode [consumeArduinosResponse] est la suivante :

```
1.  // affichage réponse
2.  private void consumeArduinosResponse(Response<List<Arduino>> response) {
3.      // erreur ?
4.      if (response.getStatus() != 0) {
5.          // affichage
6.          showAlert(response.getMessages());
7.          // retour à l'Ui
8.          return;
9.      }
10.     // on crée une liste de [CheckedArduino]
11.     List<CheckedArduino> checkedArduinos = new ArrayList<>();
12.     for (Arduino arduino : response.getBody()) {
13.         checkedArduinos.add(new CheckedArduino(arduino, false));
14.     }
15.     // on les affiche
16.     showArduinos(checkedArduinos);
17. }
```

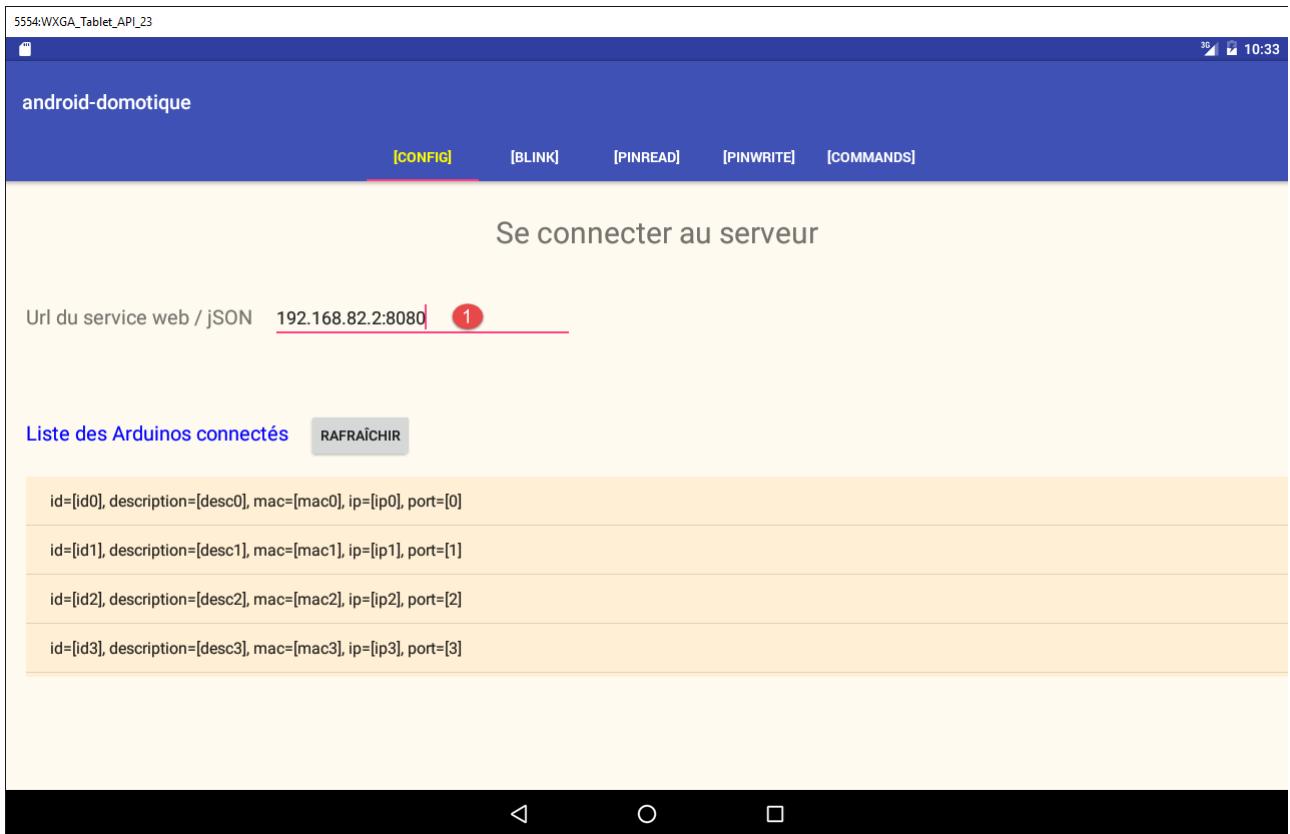
- lignes 4-11 : on regarde le code d'erreur de la réponse envoyée par le serveur :
- ligne 4 : si le code d'erreur est différent de zéro ;
- ligne 6 : on affiche les messages stockés par le serveur dans le champ [messages] de la réponse ;
- ligne 8 : on revient à l'Ui ;
- lignes 11-16 : s'il n'y a pas eu d'erreurs, on affiche la liste des Arduinos reçue, après l'avoir transformée en un type *List<CheckedArduino>* ;

La méthode [showArduinos] est la suivante :

```
1.  private void showArduinos(List<CheckedArduino> checkedArduinos) {
2.      // on crée une liste de String à partir la liste des Arduinos
3.      List<String> strings = new ArrayList<>();
4.      for (CheckedArduino checkedArduino : checkedArduinos) {
5.          strings.add(checkedArduino.toString());
6.      }
7.      // on l'affiche
8.      listArduinos.setAdapter(new ArrayAdapter<>(activity, android.R.layout.simple_list_item_1, android.R.id.text1,
9.          strings));
9. }
```

Travail : faites les modifications précédentes et exécutez votre projet.

Vous devez obtenir la vue suivante lorsque vous cliquez sur le bouton [Rafraîchir] :



La saisie en [1] n'est pas utilisée. Vous pouvez donc mettre n'importe quoi tant que ça respecte le format attendu.

5.6.10.4 Un patron pour afficher un Arduino

Pour l'instant, les Arduinos connectés sont affichés dans la vue [Config] de la façon suivante :

```
id=[id0], description=[desc0], mac=[mac0], ip=[ip0], port=[0]
id=[id1], description=[desc1], mac=[mac1], ip=[ip1], port=[1]
id=[id2], description=[desc2], mac=[mac2], ip=[ip2], port=[2]
id=[id3], description=[desc3], mac=[mac3], ip=[ip3], port=[3]
```

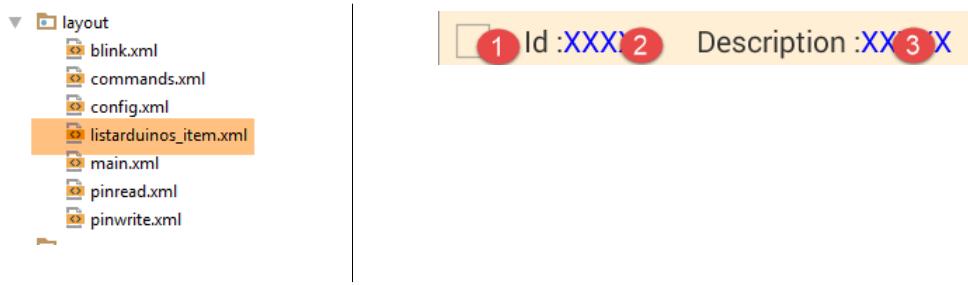
On veut désormais les afficher de la façon suivante :

1 Id : XXX 2 Description : XX 3 X

- en [1], une case à cocher qui permettra de sélectionner un Arduino. Cette case à cocher sera cachée lorsqu'on voudra présenter une liste d'Arduinos non sélectionnables ;
- en [2], l'identifiant de l'Arduino ;
- en [3], sa description ;

Ce qui suit reprend des concepts développés dans les projets [exemple-19] et [exemple-19B] du paragraphe 1.20, page 223. Revoyez-les si besoin est.

Nous créons tout d'abord la vue qui va afficher un élément de la liste des Arduinos :



Le code de la vue [listarduinos_item] ci-dessus est le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:id="@+id/RelativeLayout1"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:background="@color/wheat"
7.     android:orientation="vertical" >
8.
9.     <CheckBox
10.         android:id="@+id/checkBoxArduino"
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:layout_alignParentLeft="true"
14.         android:layout_alignParentTop="true"
15.         android:layout_toRightOf="@+id/txt_arduino_description" />
16.
17.     <TextView
18.         android:id="@+id/TextView1"
19.         android:layout_width="wrap_content"
20.         android:layout_height="wrap_content"
21.         android:layout_alignBaseline="@+id/checkBoxArduino"
22.         android:layout_marginLeft="40dp"
23.         android:text="@string/txt_arduino_id" />
24.
25.     <TextView
26.         android:id="@+id/txt_arduino_id"
27.         android:layout_width="wrap_content"
28.         android:layout_height="wrap_content"
29.         android:layout_alignBaseline="@+id/checkBoxArduino"
30.         android:layout_alignParentTop="true"
31.         android:layout_toRightOf="@+id/TextView1"
32.         android:text="@string/dummy"
33.         android:textColor="@color/blue" />
34.
35.     <TextView
36.         android:id="@+id/TextView2"
37.         android:layout_width="wrap_content"
38.         android:layout_height="wrap_content"
39.         android:layout_alignBaseline="@+id/checkBoxArduino"
40.         android:layout_alignParentTop="true"
41.         android:layout_marginLeft="20dp"
42.         android:layout_toRightOf="@+id/txt_arduino_id"
43.         android:text="@string/txt_arduino_description" />
44.
45.     <TextView
46.         android:id="@+id/txt_arduino_description"
47.         android:layout_width="wrap_content"
48.         android:layout_height="wrap_content"
49.         android:layout_alignBaseline="@+id/checkBoxArduino"
50.         android:layout_alignTop="@+id/TextView2"
51.         android:layout_toRightOf="@+id/TextView2"
52.         android:text="@string/dummy"
53.         android:textColor="@color/blue" />
54.
55. </RelativeLayout>
```

- lignes 9-15 : la case à cocher ;
- lignes 17-23 : le texte [Id :] ;
- lignes 25-33 : l'id de l'Arduino sera inscrit ici ;
- lignes 35-43 : le texte [Description :] ;
- lignes 45-53 : la description de l'Arduino sera inscrite ici ;

Cette vue utilise des textes (lignes 23, 32, 43) définis dans [res / values / strings.xml] :

```
1.     <string name="dummy">XXXXX</string>
```

```

2.
3.     <!-- listarduinos_item -->
4.     <string name="txt_arduino_id">Id : </string>
5.     <string name="txt_arduino_description">Description : </string>

```

La vue utilise également une couleur (lignes 33, 53) définie dans [res / values / colors.xml] :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.
4.     <color name="red">#FF0000</color>
5.     <color name="blue">#0000FF</color>
6.     <color name="wheat">#FFEFD5</color>
7.     <color name="floral_white">#FFFAF0</color>
8.
9. </resources>

```

Le gestionnaire d'affichage d'un élément de la liste des Arduinos



La classe [ListArduinosAdapter] est la classe appelée par le [ListView] pour afficher chacun des éléments de la liste des Arduinos. Son code est le suivant :

```

1. package istia.st.android.vues;
2.
3. import istia.st.android.R;
4. ...
5.
6. public class ListArduinosAdapter extends ArrayAdapter<CheckedArduino> {
7.
8.     // le tableau des arduinos
9.     private List<CheckedArduino> arduinos;
10.    // le contexte d'exécution
11.    private Context context;
12.    // l'id du layout d'affichage d'une ligne de la liste des arduinos
13.    private int layoutResourceId;
14.    // la ligne comporte ou non un checkbox
15.    private Boolean selectable;
16.
17.    // constructeur
18.    public ListArduinosAdapter(Context context, int layoutResourceId, List<CheckedArduino> arduinos, Boolean selectable) {
19.        // parent
20.        super(context, layoutResourceId, arduinos);
21.        // on mémorise les infos
22.        this.arduisos = arduinos;
23.        this.context = context;
24.        this.layoutResourceId = layoutResourceId;
25.        this.selectable = selectable;
26.    }
27.
28.    @Override
29.    public View getView(final int position, View convertView, ViewGroup parent) {
30.        ...
31.    }
32. }

```

- ligne 18 : le constructeur de la classe admet quatre paramètres : l'activité en cours d'exécution, l'identifiant de la vue à afficher pour chaque élément de la source de données, la source de données qui alimente la liste, un booléen qui indique si la case à cocher associée à chaque Arduino doit être affichée ou non ;
- lignes 8-15 : ces quatre informations sont mémorisées localement ;

Ligne 29, la méthode [getView] est chargée de générer la vue n° [position] dans le [ListView] et d'en gérer les événements. Son code est le suivant :

```

1. @Override
2. public View getView(int position, View convertView, ViewGroup parent) {

```

```

3.     // l'arduino courant
4.     final CheckedArduino arduino = arduinos.get(position);
5.     // on crée la ligne courante
6.     View row = ((Activity) context).getLayoutInflater().inflate(layoutResourceId, parent, false);
7.     // on récupère les références sur les [TextView]
8.     TextView txtArduinoId = (TextView) row.findViewById(R.id.txt_arduino_id);
9.     TextView txtArduinoDesc = (TextView) row.findViewById(R.id.txt_arduino_description);
10.    // on remplit la ligne
11.    txtArduinoId.setText(arduino.getId());
12.    txtArduinoDesc.setText(arduino.getDescription());
13.    // la CheckBox n'est pas toujours visible
14.    CheckBox ck = (CheckBox) row.findViewById(R.id.checkBoxArduino);
15.    ck.setVisibility(selectable ? View.VISIBLE : View.INVISIBLE);
16.    if (selectable) {
17.        // on lui affecte sa valeur
18.        ck.setChecked(arduino.isChecked());
19.        // on gère le clic
20.        ck.setOnCheckedChangeListener(new OnCheckedChangeListener() {
21.
22.            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
23.                arduino.setChecked(isChecked);
24.            }
25.        });
26.    }
27.    // on rend la ligne
28.    return row;
29. }

```

- ligne 2 : le 1er paramètre est la position dans le [ListView] de la ligne à créer. C'est également la position dans la liste des Arduinos mémorisée localement ;
- ligne 4 : on récupère une référence sur l'Arduino qui va être associé à la ligne construite ;
- ligne 6 : la ligne courante est construite à partir de la vue [listarduinos_item.xml] ;
- lignes 8-9 : les références sur les deux [TextView] sont récupérées ;
- lignes 11-12 : les deux [TextView] reçoivent leur valeur ;
- ligne 14 : on récupère une référence sur la case à cocher ;
- ligne 15 : on la rend visible ou non, selon la valeur [selectable] passée initialement au constructeur ;
- ligne 16 : si la case à cocher est présente ;
- ligne 18 : on lui affecte la valeur [isChecked] de l'Arduino courant ;
- lignes 20-26 : on gère le clic sur la case à cocher ;
- ligne 23 : la valeur de la case à cocher est mémorisée dans l'Arduino courant ;

Gestion de la liste des Arduinos

L'affichage de la liste des Arduinos est pour le moment géré par deux méthodes de la classe [ConfigFragment] :

- [clearArduinos] : qui affiche une liste vide ;
- [showArduinos] : qui affiche la liste renvoyée par le serveur ;

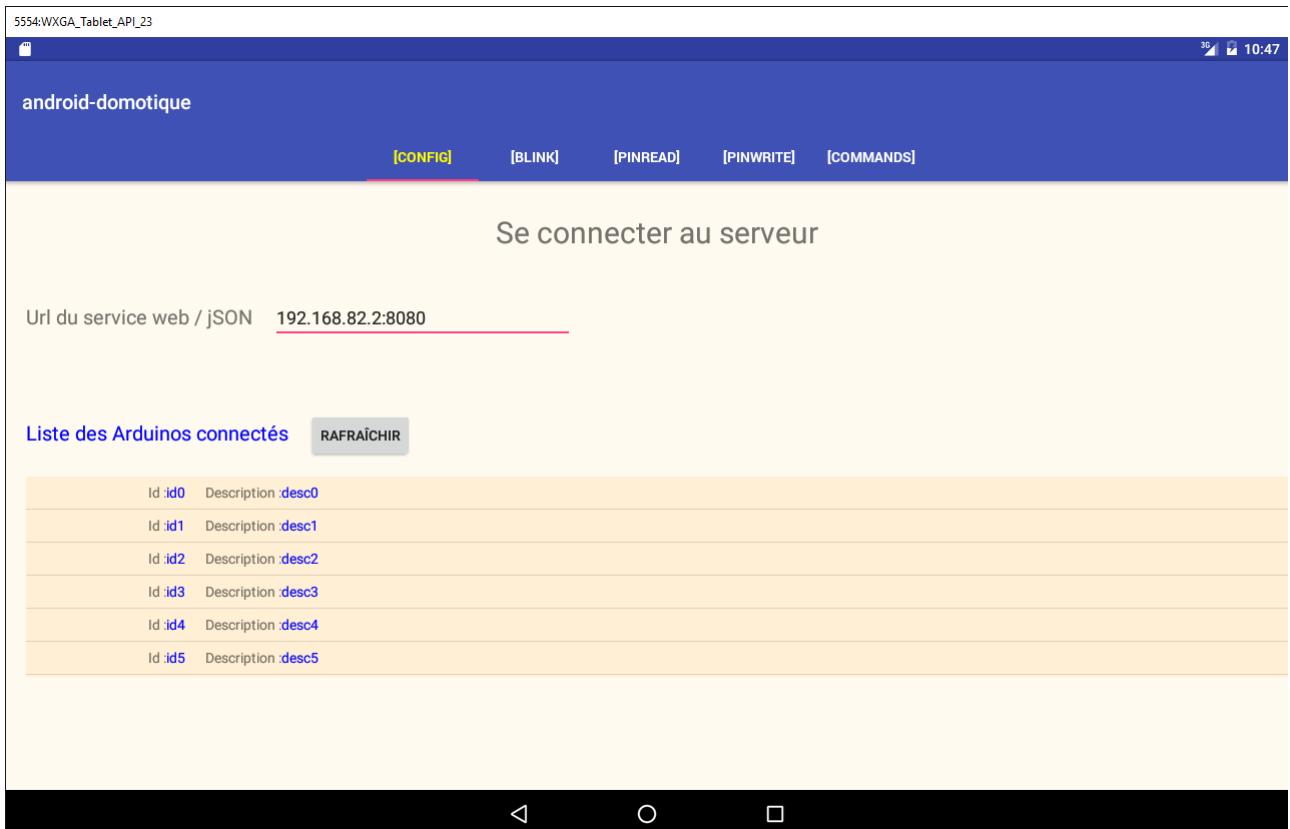
Ces deux méthodes évoluent de la façon suivante :

```

1.    // raz liste des Arduinos
2.    private void clearArduinos() {
3.        // on affiche une liste vide
4.        ListArduinosAdapter adapter = new ListArduinosAdapter(getActivity(), R.layout.listarduinos_item, new
5.        ArrayList<CheckedArduino>(), false);
6.        listArduinos.setAdapter(adapter);
7.    }
8.    // affichage liste d'Arduinos
9.    private void showArduinos(List<CheckedArduino> checkedArduinos) {
10.        // on affiche les Arduinos
11.        ListArduinosAdapter adapter = new ListArduinosAdapter(getActivity(), R.layout.listarduinos_item, checkedArduinos,
12.        false);
13.        listArduinos.setAdapter(adapter);

```

Travail : Faites ces modifications et testez la nouvelle application.



5.6.10.5 La session

La session est l'endroit où nous mettons les informations partagées par les fragments et l'activité. Les fragments ont tous besoin de faire afficher la liste des Arduinos connectée. Aussi une première version de la session sera la suivante :

```

1. package client.android.architecture.custom;
2.
3. import client.android.activity.CheckedArduino;
4. import client.android.architecture.core.AbstractSession;
5.
6. import java.util.ArrayList;
7. import java.util.List;
8.
9. public class Session extends AbstractSession {
10.     // données à partager entre fragments eux-mêmes et entre fragments et activité
11.     // les éléments qui ne peuvent être sérialisés en JSON doivent avoir l'annotation @JsonIgnore
12.     // ne pas oublier les getters et setters nécessaires pour la sérialisation / désérialisation JSON
13.
14.     // la liste des Arduinos
15.     private List<CheckedArduino> checkedArduinos = new ArrayList<>();
16.
17.     // getters et setters
18. ...
19. }
```

Travail : créez la classe [Session] précédente.

La création de cette session nous amène à modifier le code précédent de la façon suivante :

```

1.     // affichage réponse
2.     private void consumeArduinosResponse(Response<List<Arduino>> response) {
3.         // erreur ?
4.         if (response.getStatus() != 0) {
5.             // affichage
6.             showAlert(response.getMessages());
7.             // annulation
8.             doAnnuler();
9.             // retour à l'Ui
10.            return;
11.        }
```

```

12.     // on crée une liste de [CheckedArduino]
13.     List<CheckedArduino> checkedArduinos = new ArrayList<>();
14.     for (Arduino arduino : response.getBody()) {
15.         checkedArduinos.add(new CheckedArduino(arduino, false));
16.     }
17.     // on la met en session
18.     session.setCheckedArduinos(checkedArduinos);
19.     // on les affiche
20.     showArduinos(checkedArduinos);
21.     // on annule l'attente
22.     cancelWaitingTasks();
23. }
```

- ligne 18 : la liste des Arduinos créée par les lignes précédentes est mise dans la session ;

5.6.10.6 Gestion de l'état du fragment

Lors d'une rotation du périphérique, les composants visuels de la vue sont restitués (par défaut) dans l'état où ils étaient lors de la conception de la vue :

- le [ListView] contient les éléments que le concepteur y a mis ;
- le message d'erreur est dans l'état visible ou pas dans lequel le concepteur l'a mis ;

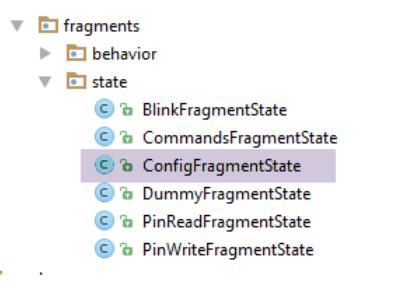
Les états des composants visuels à la conception peuvent convenir ou pas lors de la restauration d'un fragment. Qu'en est-il ici ?

- le [ListView] doit afficher la liste des Arduinos connectés. La valeur du [ListView] à la conception ne peut donc pas être utilisée ;
- le [TextView] du message d'erreur doit être restitué dans l'état visible ou pas qu'il avait lors de la sauvegarde. Sa valeur à la conception ne peut convenir à ces deux cas ;

Il nous faut donc sauvegarder l'état de ces deux composants lors d'une sauvegarde de l'état du fragment :

- la liste des Arduinos connectés ;
- la visibilité (affiché / caché) du message d'erreur sur la saisie de l'URL du service web / JSON ;

La liste des Arduinos étant présente en session, elle sera automatiquement sauvegardée. La visibilité du message d'erreur sera mémorisée dans la classe [ConfigFragmentState] suivante :



```

1. package client.android.fragments.state;
2.
3. import client.android.architecture.custom.CoreState;
4.
5. public class ConfigFragmentState extends CoreState {
6.
7.     // visibilité message d'erreur
8.     private boolean txtMsgErreurUrlServiceRestVisible;
9.
10.    // getters et setters
11.    ...
12. }
```

Travail : créez la classe [ConfigFragmentState] précédente.

Pour restituer correctement les états des fragments il faut que leurs méthodes [getNumView] et [saveFragment] soient modifiées. Par exemple, celle du fragment [BlinkFragment] est actuellement la suivante :

```

1. @Override
2. public CoreState saveFragment() {
3.     // il faut sauvegarder le fragment
```

```

4.     DummyFragmentState state=new DummyFragmentState();
5.     // ...
6.     return state;
7.     // s'il n'y a rien à sauvegarder faire [return new CoreState();] et supprimer la classe [DummyFragmentState]
8. }
9.
10.    @Override
11.    protected int getNumView() {
12.        // il faut retourner le n° du fragment dans le tableau des fragments gérés par l'activité (cf MainActivity)
13.        return 0;
14.    }

```

Si on ne fait rien, l'état rendu ligne 6 va être sauvegardé dans l'élément 0 (ligne 13) du tableau `CoreState[] coreStates` de la classe `[AbstractSession]` (ligne 5 ci-dessous) :

```

1. public class AbstractSession implements ISession {
2.     ...
3.
4.     // état des vues
5.     private CoreState[] coreStates = new CoreState[0];
6.     ...

```

Or il doit être sauvegardé dans l'élément correspondant au n° du fragment `[BlinkFragment]` dans le tableau des fragments définis dans la classe `[MainActivity]` (ligne 9 ci-dessous) :

```

1. @EActivity
2. @OptionsMenu(R.menu.menu_main)
3. public class MainActivity extends AbstractActivity {
4.
5.     ...
6.
7.     @Override
8.     protected AbstractFragment[] getFragments() {
9.         return new AbstractFragment[]{new ConfigFragment_(), new BlinkFragment_(), new PinReadFragment_(), new
PinWriteFragment_(), new CommandsFragment_()};
10.    }
11.
12.

```

Les n°s des fragments ont été définis dans l'interface `[IMainActivity]` :

```

1. public interface IMainActivity extends IDao {
2.
3.     ...
4.
5.     // n°s de vue
6.     int VUE_CONFIG = 0;
7.     int VUE_BLINK = 1;
8.     int VUE_PINREAD = 2;
9.     int VUE_PINWRITE = 3;
10.    int VUE_COMMANDS = 4;
11. }

```

Au final, l'état du fragment `[BlinkFragment]` sera géré correctement si on écrit :

```

1.     @Override
2.     public CoreState saveFragment() {
3.         // il faut sauvegarder le fragment
4.         DummyFragmentState state=new DummyFragmentState();
5.         // ...
6.         return state;
7.         // s'il n'y a rien à sauvegarder faire [return new CoreState();] et supprimer la classe [DummyFragmentState]
8.     }
9.
10.    @Override
11.    protected int getNumView() {
12.        // il faut retourner le n° du fragment dans le tableau des fragments gérés par l'activité (cf MainActivity)
13.        return IMainActivity.VUE_BLINK;
14.    }

```

- ligne 14 : on retourne le n° du fragment `[BlinkFragment]` dans le tableau des fragments gérés par l'activité ;

Par ailleurs, la classe `[CoreState]` parente des états des fragments est pour l'instant la suivante (cf paragraphe 5.6.7.2, page 436) :

```

1. package client.android.architecture.custom;
2.
3. import client.android.architecture.core.MenuItemState;
4. import client.android.fragments.state.*;
5. import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
6. import com.fasterxml.jackson.annotation.JsonSubTypes;

```

```

7. import com.fasterxml.jackson.annotation.JsonProperty;
8.
9. @JsonIgnoreProperties(ignoreUnknown = true)
10. @JsonProperty(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY)
11. @JsonSubTypes({
12.     @JsonSubTypes.Type(value = ConfigFragmentState.class),
13.     @JsonSubTypes.Type(value = BlinkFragmentState.class),
14.     @JsonSubTypes.Type(value = PinReadFragmentState.class),
15.     @JsonSubTypes.Type(value = PinWriteFragmentState.class),
16.     @JsonSubTypes.Type(value = CommandsFragmentState.class)})
17. }
18. public class CoreState {
19.     // fragment visité ou non
20.     protected boolean hasBeenVisited = false;
21.     // état de l'éventuel menu du fragment
22.     protected MenuItemState[] menuOptionsState;
23.
24.     // getters et setters
25.     ...
26. }
```

- lignes 12-16 : la classe [DummyFragmentState] ne figure pas dans la liste des classes filles de la classe [CoreState]. Or la méthode [saveFragment] de la classe [BlinkFragment] rend actuellement un type [DummyFragmentState]. Si on laisse les choses en l'état, la sérialisation / désérialisation de la session va échouer et la session ne sera pas restaurée conduisant à un plantage de l'application ;

La méthode [saveFragment] du fragment [BlinkFragment] doit être réécrite de la façon suivante :

```

1.     @Override
2.     public CoreState saveFragment() {
3.         // il faut sauvegarder le fragment
4.         BlinkFragmentState state=new BlinkFragmentState();
5.         // ...
6.         return state;
7.         // s'il n'y a rien à sauvegarder faire [return new CoreState();] et supprimer la classe [DummyFragmentState]
8.     }
```

Travail : dans chacun des fragments, modifiez la méthode [getNumView] pour qu'elle rende le n° du fragment et la méthode [saveFragment] pour qu'elle rende une instance de la classe d'état du fragment (comme ci-dessus).

5.6.10.7 Gestion du cycle de vie du fragment

Nous nous intéressons ici au cycle de vie du fragment [ConfigFragment], notamment aux quatre méthodes :

- [saveFragment] : doit sauvegarder l'état du fragment afin qu'il puisse être restitué ultérieurement ;
- [initFragment] : qui doit initialiser certains champs du fragment si besoin est. Cette méthode est appelée au démarrage de l'application et à chaque fois qu'il y a une rotation du périphérique. Très exactement, elle est appelée lorsque le fragment devient visible après l'un des deux événements précédents ;
- [initView] : qui doit initialiser certains composants de la vue si besoin est. Cette méthode est appelée à chaque fois que [initFragment] a été appelée et lorsque la vue doit être régénérée parce que le fragment est, à un moment donné, sorti de l'adjacence du fragment affiché. Comme précédemment, elle est appelée lorsque le fragment devient visible après l'un des ces événements ;
- [updateOnRestore] : qui est exécutée après les deux précédentes méthodes lorsqu'il y a eu rotation du périphérique mais également lorsqu'il y a eu navigation. Son rôle est de restituer l'état précédent du fragment ;

Ces méthodes seront les suivantes :

```

1. // adaptateur de la liste des Arduinos
2. private ListArduinosAdapter adapterListArduinos;
3.
4. ...
5. // gestion du cycle de vie du fragment -----
6.
7.     @Override
8.     public CoreState saveFragment() {
9.         ConfigFragmentState state = new ConfigFragmentState();
10.        state.setTxtMsgErreurUrlServiceRestVisible(txtMsgErreurUrlServiceRest.getVisibility() == View.VISIBLE);
11.        return state;
12.    }
13.
14.     @Override
15.     protected void initFragment(CoreState previousState) {
16.         // adaptateur listArduinos
```

```

17.     adapterListArduinos = new ListArduinosAdapter(activity, R.layout.listarduinos_item, session.getCheckedArduinos(),
18.         false);
19. }
20.
21. @Override
22. protected void initView(CoreState previousState) {
23.     // liaison listview / adaptateur
24.     listArduinos.setAdapter(adapterListArduinos);
25.     // 1ère visite ?
26.     if (previousState == null) {
27.         // ListView vide - fait par [initFragment]
28.         // message d'erreur caché
29.         txtMsgErreurUrlServiceRest.setVisibility(View.INVISIBLE);
30.     } else {
31.         // on restitue la visibilité du message d'erreur
32.         ConfigFragmentState state = (ConfigFragmentState) previousState;
33.         txtMsgErreurUrlServiceRest.setVisibility(state.isTxtMsgErreurUrlServiceRestVisible() ? View.VISIBLE :
View.INVISIBLE);
34.     }
35. }
36.
37.
38. @Override
39. protected void updateOnSubmit(CoreState previousState) {
40. }
41.
42.
43. @Override
44. protected void updateOnRestore(CoreState previousState) {
45. }
46.
47.
48. @Override
49. protected void notifyEndOfUpdates() {
50.     // boutons
51.     initButtons();
52. }

```

- ligne 2 : l'adaptateur du *ListView* des Arduinos. Est une variable globale parce qu'utilisée dans différentes méthodes ;
- lignes 7-12 : la méthode `[saveFragment]` sauve dans un type `[ConfigFragmentState]` la visibilité du *TextView* `txtMsgErreurUrlServiceRestVisible` (ligne 10) ;
- lignes 14-19 : la méthode `[initFragment]` initialise l'adaptateur de la ligne 2 avec la liste des Arduinos présente en session (ligne 17). On rappelle que le rôle de `[initFragment]` est d'initialiser des champs du fragment. Ici cette initialisation est à faire dans tous les cas, 1ère visite (`previousState==null`) ou pas ;
- ligne 17 : on voit que l'adaptateur est lié à la source de données `[session.getCheckedArduinos]`. Il ne faut pas que celle-ci ait la valeur `null`. Pour cette raison, le champ `[session.checkedArduinos]` est initialisé avec une liste vide dans la session :

```

1.     // la liste des Arduinos
2.     private List<CheckedArduino> checkedArduinos = new ArrayList<>();

```

- lignes 21-35 : la méthode `[initView]` a pour rôle d'initialiser certains composants de l'interface visuelle, notamment ceux dont la valeur n'est pas conservée lors de la rotation du périphérique ;
- ligne 24 : le *ListView* des Arduinos est associé à l'adaptateur de la ligne 2 ;
- lignes 28-32 : on distingue la 1ère visite des autres visites ;
- ligne 29 : lors de la 1ère visite, on doit afficher un *ListView* vide. C'est le cas puisque lors de la 1ère visite, l'adaptateur du *[ListView]* a été associé à une liste vide (ligne 17) ;
- ligne 31 : le message d'erreur est caché ;
- lignes 32-36 : le cas où ce n'est pas la 1ère visite ;
- le *[ListView]* est déjà dans le bon état depuis la ligne 24. Il n'y a rien de plus à faire ;
- lignes 34-35 : on restaure le message d'erreur dans l'état où il était lors de la dernière sauvegarde du fragment ;
- lignes 31-36 : la méthode `[updateOnRestore]` doit remettre le fragment dans son état initial. On arrive à la méthode `[updateOnRestore]` de deux façons :
 - soit parce qu'il y a eu rotation du périphérique. Dans ce cas, toutes les initialisations à faire l'ont été dans `[initView]` ;
 - soit parce qu'on navigue d'un onglet vers l'onglet `[Config]`. Si le fragment `[Config]` est sorti de l'adjacence des fragments affichés depuis qu'on l'a quitté, la méthode `[initView]` a alors été exécutée et le fragment est déjà dans l'état souhaité. Si le fragment `[Config]` n'est pas sorti de l'adjacence des fragments affichés depuis qu'on l'a quitté, ses composants visuels n'ont alors pas changé d'état et il n'y a rien à faire ;

On voit que la méthode `[updateOnRestore]` n'a rien à faire. C'est parfois le cas, parfois pas. La différence vient de la méthode `[updateOnSubmit]` : si cette méthode fait quelque chose qui rend inutiles certaines initialisations faites dans `[initView]`, alors ces initialisations devraient être faites dans la méthode `[updateOnRestore]`. Prenons l'exemple d'un

bouton radio à trois valeurs V1, V2, V3. Peut-être que dans le cas d'une navigation associée à une action [SUBMIT], le bouton radio coché doit toujours être celui de valeur V1. Dans ce cas, restaurer la valeur du bouton radio dans la méthode [initView] est inutile, car dans le cas d'un [SUBMIT], cette valeur va être remplacée par celle donnée par la méthode [updateOnSubmit]. Il est préférable alors de déplacer cette restauration dans la méthode [updateOnRestore] pour éviter de faire parfois une opération inutile.

- lignes 48-52 : la méthode [notifyEndOfUpdates] est exécutée après toutes les précédentes ;
- ligne 51 : les boutons sont mis dans leur état initial : bouton [Rafraîchir] affiché, bouton [Annuler] caché :

Travail : ajoutez le code précédent dans [ConfigFragment] puis exécutez l'application. Constatez que lorsque vous faites une rotation du périphérique, l'onglet [Config] garde son état (message d'erreur, liste des Arduinos). Vérifiez qu'il en est de même lorsque vous faites une simple navigation onglet [config] --> onglet [Commands] --> onglet [Config]. Dans ce dernier cas, si vous avez gardé dans [IMainActivity] une adjacence de fragments de 1, alors la vue du fragment [ConfigFragment] est détruite lorsqu'on passe à l'onglet [Commands] puis recréée lorsqu'on revient à l'onglet [Config]. Lors des tests, examinez les logs.

5.6.10.8 Amélioration du code

Le code du fragment [ConfigFragment] peut être amélioré. Nous avons par exemple écrit :

```

1. // adaptateur de la liste des Arduinos
2. private ListArduinosAdapter adapterListArduinos;
3.
4. ...
5.
6. // affichage liste d'Arduinos
7. private void showArduinos(List<CheckedArduino> checkedArduinos) {
8.     // on affiche les Arduinos
9.     ListArduinosAdapter adapter = new ListArduinosAdapter(getActivity(), R.layout.listarduinos_item, checkedArduinos,
10.        false);
11.    listArduinos.setAdapter(adapter);
12. }
13. // raz liste des Arduinos
14. private void clearArduinos() {
15.     // on affiche une liste vide
16.     ListArduinosAdapter adapter = new ListArduinosAdapter(getActivity(), R.layout.listarduinos_item, new
17.        ArrayList<CheckedArduino>(), false);
18.    listArduinos.setAdapter(adapter);
19. }
```

- on voit que lignes 9 et 16, on utilise une variable locale déconnectée du champ de la ligne 2 alors que c'est bien la même entité qu'on veut manipuler ;

Nous faisons évoluer le code de la façon suivante :

```

1. // adaptateur de la liste des Arduinos
2. private ListArduinosAdapter adapterListArduinos;
3.
4. @Click(R.id.btn_Rafraichir)
5. protected void doRafraichir() {
6.     ...
7. }
8.
9. private void getArduinosInBackground() {
10. ...
11.     // on la consomme
12.     consumeArduinosResponse(response);
13. }
14.
15. // affichage réponse
16. private void consumeArduinosResponse(Response<List<Arduino>> response) {
17.     // erreur ?
18.     if (response.getStatus() != 0) {
19.         // affichage
20.         showAlert(response.getMessages());
21.         // annulation
22.         doAnnuler();
23.         // retour à l'Ui
24.         return;
25.     }
26.     // on crée une liste de [CheckedArduino]
27.     List<CheckedArduino> checkedArduinos = session.getCheckedArduinos();
28.     checkedArduinos.clear();
29.     for (Arduino arduino : response.getBody()) {
30.         checkedArduinos.add(new CheckedArduino(arduino, false));
31.     }
32. }
```

```

32.    // on les affiche
33.    adapterListArduinos.notifyDataSetChanged();
34.    // on annule l'attente
35.    cancelWaitingTasks();
36. }
37.
38. @Override
39. protected void initFragment(CoreState previousState) {
40.     // adaptateur listArduinos
41.     adapterListArduinos = new ListArduinosAdapter(activity, R.layout.listarduinoss_item, session.getCheckedArduinos(),
42.         false);
42.
43. }
44.
45. @Override
46. protected void initView(CoreState previousState) {
47.     // liaison listview / adaptateur
48.     listArduinos.setAdapter(adapterListArduinos);
49.     ...
50. }

```

- lorsque la méthode de la ligne 5 est exécutée, le cycle de vie du fragment a été exécuté. Donc :
 - l'adaptateur de la ligne 2 a été associé à sa source de données (ligne 41) ;
 - le [ListView] des Arduinos connectés a été relié à cet adaptateur (ligne 48) ;

Lorsque nous voulons changer l'affichage du [ListView], il faut faire deux choses :

- changer le contenu de la source de données [session.checkedArduinos] ;
- signaler ce changement à l'adaptateur par l'instruction [adapterListArduinos.notifyDataSetChanged()] ;

Il s'agit bien de changer le contenu de la source de données et non la source de données elle-même. Si on change la source de données elle-même, l'opération [adapterListArduinos.notifyDataSetChanged()] continuera à faire afficher l'ancienne source de données. Il faudrait alors associer l'adaptateur avec la nouvelle source de données.

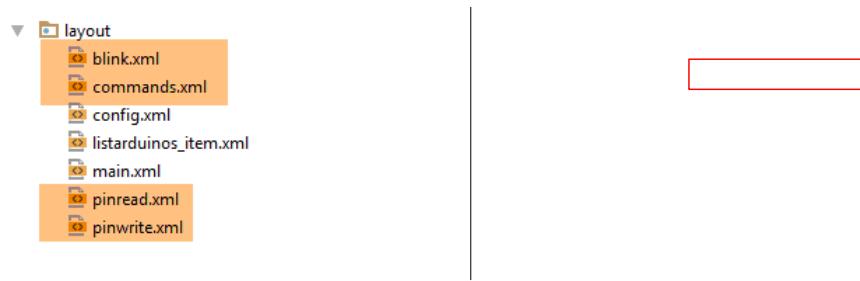
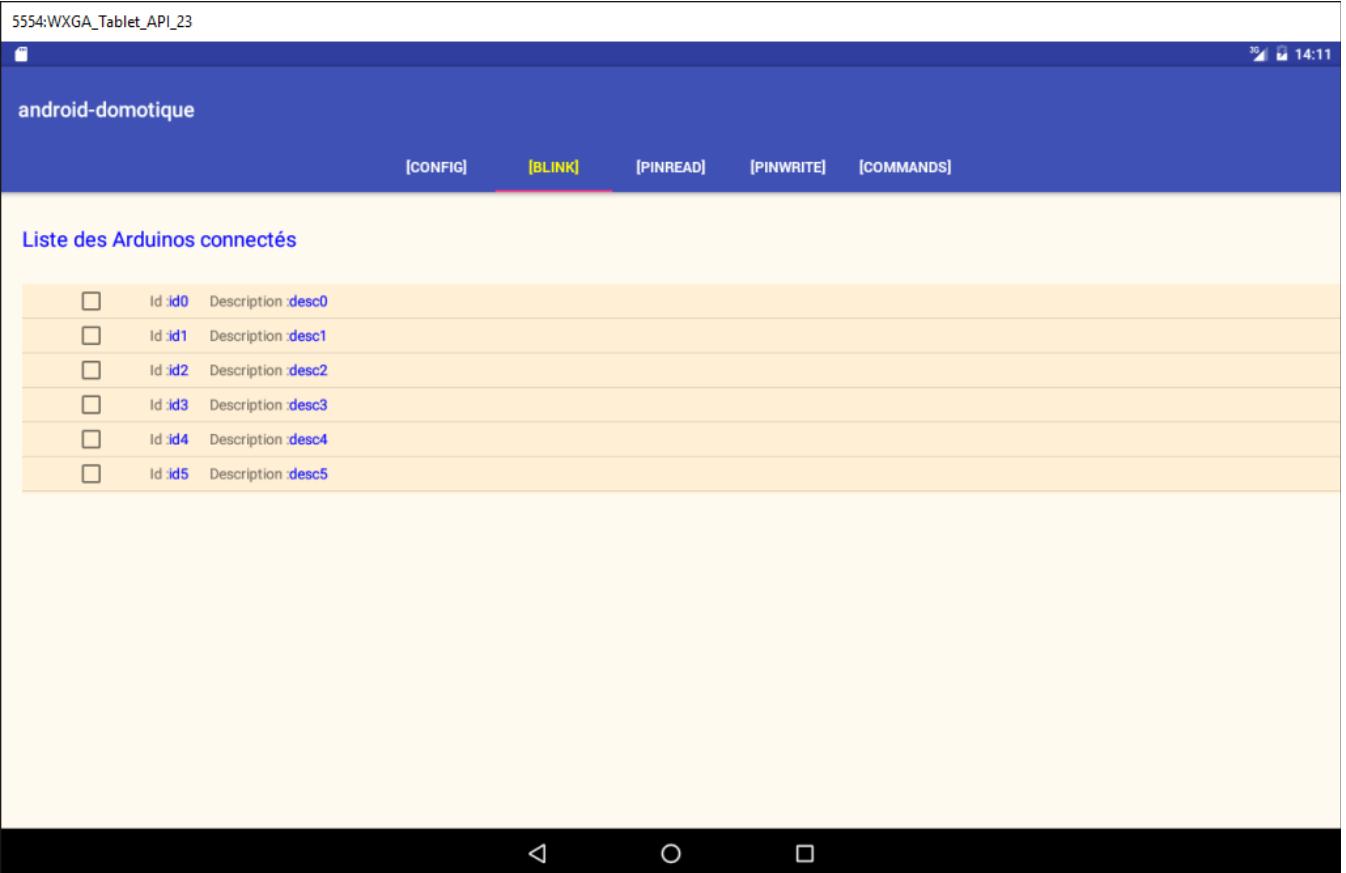
Le code est le suivant :

- ligne 27 : nous récupérons la source de données ;
- ligne 28 : nous la vidons. Pour cette raison, nous avons supprimé la méthode [clearArduinos] ;
- lignes 29-31 : dans cette liste devenue vide, on ajoute de nouveaux éléments ;
- ligne 33 : on dit à l'adaptateur de se rafraîchir. Cela va rafraîchir l'affichage du [ListView] associé ;

Travail : faites ces modifications et vérifiez que votre application fonctionne toujours.

5.6.11 Communication entre vues

Pour vérifier la communication entre vues, on va faire afficher par toutes les autres vues la liste des Arduinos obtenue par la vue [Config]. Commençons par la vue [blink.xml]. Alors qu'elle n'affichait rien, elle va désormais afficher la liste des Arduinos connectés :



Le code XML de la vue [blink.xml] sera le suivant :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3.         android:id="@+id/scrollView1"
4.         android:layout_width="wrap_content"
5.         android:layout_height="wrap_content">
6.
7.     <?xml version="1.0" encoding="utf-8"?>
8.     <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
9.             xmlns:tools="http://schemas.android.com/tools"
10.            android:id="@+id/RelativeLayout1"
11.            android:layout_width="match_parent"
12.            android:layout_height="match_parent">
13.
14.         <TextView
15.             android:id="@+id/txt_arduino"
16.             android:layout_width="wrap_content"
17.             android:layout_height="wrap_content"
18.             android:layout_alignParentLeft="true"
19.             android:layout_marginTop="150dp"
20.             android:text="@string/titre_list_arduino"
21.             android:textColor="@color/blue"
22.             android:textSize="20sp" />
23.
24.         <ListView
```

```

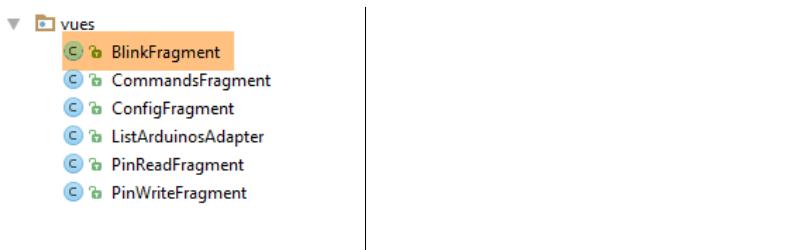
25.     android:id="@+id/ListViewArduinos"
26.     android:layout_width="match_parent"
27.     android:layout_height="200dp"
28.     android:layout_alignParentLeft="true"
29.     android:layout_below="@+id/txt_arduisos"
30.     android:layout_marginTop="30dp"
31.     android:background="@color/wheat">
32.   </ListView>
33.
34. </RelativeLayout>
35. </ScrollView>

```

Ce code a été repris directement de la vue [config.xml]. On a simplement modifié la marge haute ligne 19.

Travail : dupliquez ce code dans les vues [commands.xml, pinread.xml, pinwrite.xml].

Le code du fragment [BlinkFragment] associé à la vue [blink.xml] évolue lui aussi :



```

1.  // composants visuels
2.  @ViewById(R.id.ListViewArduinos)
3.  protected ListView listArduinos;
4.
5.  // adaptateur de la liste des Arduinos
6.  private ListArduinosAdapter adapterListArduinos;
7. ...
8.
9.  // méthodes imposées par la classe parent -----
10.
11. ...
12.  @Override
13.  protected void initFragment(CoreState previousState) {
14.      // adaptateur listArduinos
15.      adapterListArduinos = new ListArduinosAdapter(activity, R.layout.listarduios_item, session.getCheckedArduinos(),
16.          true);
17.  }
18.
19.  @Override
20.  protected void initView(CoreState previousState) {
21.      // liaison listview / adaptateur
22.      listArduinos.setAdapter(adapterListArduinos);
23.  }
24. ...

```

- lignes 2-3 : le composant [ListView] des Arduinos connectés ;
- ligne 6 : l'adaptateur de ce [ListView] ;
- lignes 12-23 : le code des méthodes [initFragment] et [initView] est celui déjà utilisé pour le fragment [ConfigFragment] ;
- ligne 15 : lorsque le fragment doit être réinitialisé, on réinitialise l'adaptateur de la ligne 2 en l'associant à la liste des Arduinos mémorisée en session. Le dernier paramètre [true] du constructeur [ListArduinosAdapter] signifie que l'on veut voir une case à cocher à côté de chaque Arduino ;
- ligne 22 : lorsque la vue du fragment doit être réinitialisée, on associe le [ListView] des Arduinos connectés avec l'adaptateur de la ligne 6 ;

Travail : Dupliquez ce code dans les autres fragments [CommandsFragment, PinReadFragment, PinWriteFragment]. Exécutez l'application et constatez maintenant que chaque onglet affiche la liste des Arduinos connectés. Constatez également que si vous cochez des Arduinos dans un onglet et que vous naviguez vers un autre onglet, vous les retrouvez cochés dans ce dernier.

Note : L'explication du maintien des Arduinos cochés est la suivante. La classe [ListArduinosAdapter] a été présentée au paragraphe 5.6.10.4, page 449. Le code lié à la case à cocher est le suivant :

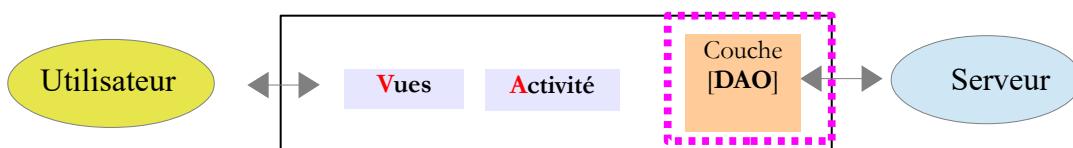
```

1.      // l'arduino courant
2.      final CheckedArduino arduino = arduinos.get(position);
3. ...
4.      // la CheckBox n'est pas toujours visible
5.      CheckBox ck = (CheckBox) row.findViewById(R.id.checkBoxArduino);
6.      ck.setVisibility(selectable ? View.VISIBLE : View.INVISIBLE);
7.      if (selectable) {
8.          // on lui affecte sa valeur
9.          ck.setChecked(arduino.isChecked());
10.         // on gère le clic
11.         ck.setOnCheckedChangeListener(new OnCheckedChangeListener() {
12.
13.             public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
14.                 arduino.setChecked(isChecked);
15.             }
16.         });
17.     }

```

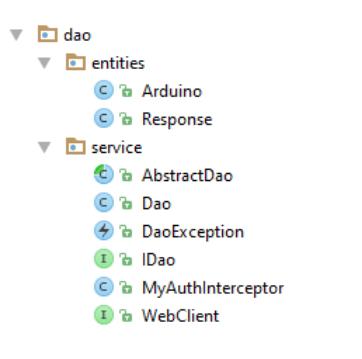
- lignes 11-15 : si dans l'onglet X, on coche une case, l'Arduino de la ligne 2 a sa propriété [checked] mise à *true* (ligne 14) ;
- lorsqu'on passe à l'onglet Y, le [ListView] des Arduinos de cet onglet est affiché. Ligne 9, on voit que si l'Arduino de la ligne 2 a sa propriété [checked] mise à *true*, alors la case [ck] de la ligne 5 va être cochée ;

5.6.12 La couche [DAO]



Note : pour cette partie, revoyez l'implémentation de la couche [DAO] dans le projet [exemple-16B] (cf paragraphe 2.8.3, page 316).

Pour l'instant, nous avons généré manuellement la liste des Arduinos connectés. Nous allons maintenant la demander au serveur web / JSON. Pour cela, nous allons construire la couche [DAO] :



5.6.12.1 L'interface IDao

L'interface [IDao] de la couche [DAO] sera la suivante :

```

1. package client.android.dao.service;
2.
3. import client.android.dao.entities.Arduino;
4. import client.android.dao.entities.Response;
5. import rx.Observable;
6.
7. import java.util.List;
8.
9. public interface IDao {
10.     // Url du service web
11.     void setUrlServiceWebJson(String url);
12.
13.     // utilisateur
14.     void setUser(String user, String mdp);
15.
16.     // timeout du client

```

```

17.     void setTimeout(int timeout);
18.
19.     // authentification basique
20.     void setBasicAuthentification(boolean isBasicAuthenticationNeeded);
21.
22.     // mode debug
23.     void setDebugMode(boolean isDebugEnabled);
24.
25.     // délai d'attente en millisecondes du client avant requête
26.     void setDelay(int delay);
27.
28.     // spécifique -----
29.     // liste des arduinos
30.     Observable<Response<List<Arduino>>> getArduinos();
31. }

```

- lignes 11-26 : ces lignes sont déjà présentes dans l'interface [IDao] du projet modèle [client-android-skel] ;
- ligne 30 : la méthode [getArduinos] permet d'obtenir la liste des Arduinos connectés sous la forme d'un observable de type [Response<List<Arduino>>] ;

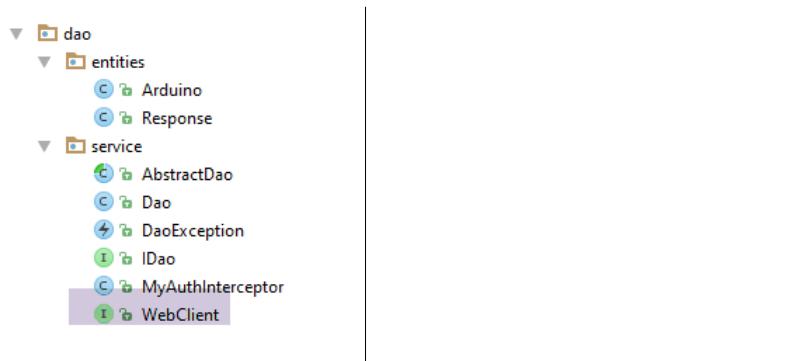
On rappelle que [Response<T>] est le type de toutes les réponses envoyées par le serveur sous la forme d'une chaîne JSON :

```

1. package client.android.dao.entities;
2.
3. import java.util.List;
4.
5. public class Response<T> {
6.
7.     // ----- propriétés
8.     // statut de l'opération
9.     private int status;
10.    // les éventuels messages d'erreur
11.    private List<String> messages;
12.    // le corps de la réponse
13.    private T body;
14.
15.    // constructeurs
16.    public Response() {
17.
18.    }
19.
20.    public Response(int status, List<String> messages, T body) {
21.        this.status = status;
22.        this.messages = messages;
23.        this.body = body;
24.    }
25.
26.    // getters et setters
27.    ...
28. }

```

5.6.12.2 L'interface [WebClient]



L'interface [WebClient] est une interface dont la bibliothèque AA fournit une implémentation. Cette interface sera la suivante :

```

1. package client.android.dao.service;
2.
3. import client.android.dao.entities.Arduino;
4. import client.android.dao.entities.Response;
5. import org.androidannotations.rest.spring.annotations.Get;
6. import org.androidannotations.rest.spring.annotations.Path;
7. import org.androidannotations.rest.spring.annotations.Rest;
8. import org.androidannotations.rest.spring.api.RestClientRootUrl;

```

```

9. import org.androidannotations.rest.spring.api.RestClientSupport;
10. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
11. import org.springframework.web.client.RestTemplate;
12.
13. import java.util.List;
14.
15. @Rest(converters = {MappingJackson2HttpMessageConverter.class})
16. public interface WebClient extends RestClientRootUrl, RestClientSupport {
17.
18.     // RestTemplate
19.     void setRestTemplate(RestTemplate restTemplate);
20.
21.     // spécifique -----
22.     // liste des arduinos
23.     @Get("/arduinos")
24.     Response<List<Arduino>> getArduinos();
25. }

```

- lignes 15-19 : ces lignes sont présentes de base dans l'interface [WebClient] du projet modèle [client-android-skel] ;
- ligne 23 : l'URL du serveur qui permet d'obtenir la liste des Arduinos avec une opération GET. On rappelle que cette URL est mesurée par rapport à l'URL racine [RestClientRootUrl] de la ligne 16 ;
- ligne 24 : le serveur renvoie la chaîne JSON d'un type [Response<List<Arduino>>]. Cette chaîne JSON est automatiquement déserialisée en le type [Response<List<Arduino>>] grâce au convertisseur JSON [MappingJackson2HttpMessageConverter] de la ligne 15 ;

5.6.12.3 La classe [Dao]

La classe [Dao] implémente l'interface [IDao] de la façon suivante :

```

1. package client.android.dao.service;
2.
3. import android.util.Log;
4. import client.android.dao.entities.Arduino;
5. import client.android.dao.entities.Response;
6. import org.androidannotations.annotations.AfterInject;
7. import org.androidannotations.annotations.Bean;
8. import org.androidannotations.annotations.EBean;
9. import org.androidannotations.rest.spring.annotations.RestService;
10. import org.springframework.http.client.ClientHttpRequestInterceptor;
11. import org.springframework.http.client.SimpleClientHttpRequestFactory;
12. import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
13. import org.springframework.web.client.RestTemplate;
14. import rx.Observable;
15.
16. import java.util.ArrayList;
17. import java.util.List;
18.
19. @EBean(scope = EBean.Scope.Singleton)
20. public class Dao extends AbstractDao implements IDao {
21.
22.     // client du service web
23.     @RestService
24.     protected WebClient webClient;
25.     // sécurité
26.     @Bean
27.     protected MyAuthInterceptor authInterceptor;
28.     // le RestTemplate
29.     private RestTemplate restTemplate;
30.     // factory du RestTemplate
31.     private SimpleClientHttpRequestFactory factory;
32.
33.     @AfterInject
34.     public void afterInject() {
35.         // log
36.         Log.d(className, "afterInject");
37.         // on construit le restTemplate
38.         factory = new SimpleClientHttpRequestFactory();
39.         restTemplate = new RestTemplate(factory);
40.         // on fixe le convertisseur JSON
41.         restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
42.         // on fixe le restTemplate du client web
43.         webClient.setRestTemplate(restTemplate);
44.     }
45.
46.     @Override
47.     public void setUrlServiceWebJson(String url) {
48.         // on fixe l'URL du service web
49.         webClient.setRootUrl(url);
50.     }
51.
52.     @Override
53.     public void setUser(String user, String mdp) {

```

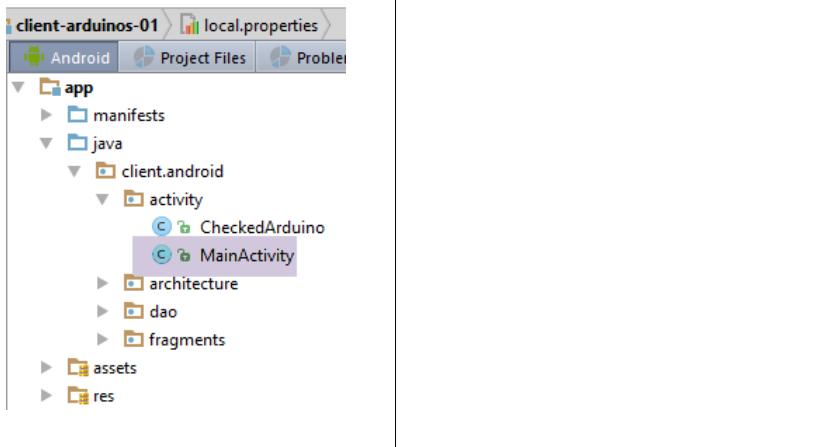
```

54.     // on enregistre l'utilisateur dans l'intercepteur
55.     authInterceptor.setUser(user, mdp);
56. }
57.
58. @Override
59. public void setTimeout(int timeout) {
60.     if (isDebugEnabled) {
61.         Log.d(className, String.format("setTimeout thread=%s, timeout=%s", Thread.currentThread().getName(), timeout));
62.     }
63.     // configuration factory
64.     factory.setReadTimeout(timeout);
65.     factory.setConnectTimeout(timeout);
66. }
67.
68. @Override
69. public void setBasicAuthentification(boolean isBasicAuthentificationNeeded) {
70.     if (isDebugEnabled) {
71.         Log.d(className, String.format("setBasicAuthentification thread=%s, isBasicAuthentificationNeeded=%s",
72.             Thread.currentThread().getName(), isBasicAuthentificationNeeded));
73.     }
74.     // intercepteur d'authentification ?
75.     if (isBasicAuthentificationNeeded) {
76.         // on ajoute l'intercepteur d'authentification
77.         List<ClientHttpRequestInterceptor> interceptors = new ArrayList<ClientHttpRequestInterceptor>();
78.         interceptors.add(authInterceptor);
79.         restTemplate.setInterceptors(interceptors);
80.     }
81.
82.     // méthodes privées -----
83.     private void log(String message) {
84.         if (isDebugEnabled) {
85.             Log.d(className, message);
86.         }
87.     }
88.
89.     // implémentation IDao spécifique -----
90.
91.     @Override
92.     public Observable<Response<List<Arduino>>> getArduinos() {
93.         // exécution client web
94.         return getResponse(new IRequest<Response<List<Arduino>>>() {
95.             @Override
96.             public Response<List<Arduino>> getResponse() {
97.                 return webClient.getArduinos();
98.             }
99.         });
100.    }
101. }

```

- lignes 19-87 : ces lignes sont de base dans la classe [Dao] du projet [client-android-skel] ;
- lignes 91-100 : implémentation de la méthode [getArduinos] ;
- ligne 94 : on appelle la méthode [getResponse] de la classe parent. L'unique paramètre de cette méthode est une instance de l'interface [IRequest<T>] ;
- lignes 95-99 : l'unique méthode de l'interface [IRequest<T>] est la méthode [T getResponse()] ;
- ligne 94 : le type T de [IRequest<T>] doit être le type T du résultat *Observable<T>* de la méthode de la ligne 92, donc ici, un type [Response<List<Arduino>>] ;
- ligne 97 : la méthode [IRequest.getResponse()] délègue le travail à la méthode [webClient.getArduinos()] que nous avons présentée. [webClient], défini ligne 24, est instancié par la bibliothèque AA et est une instance de l'interface [WebClient] que nous avons présentée ;

5.6.13 L'activité [MainActivity]



Nous avons déjà présenté l'activité [MainActivity] au paragraphe 5.6.8, page 436. Elle étend la classe [AbstractActivity] et à ce titre implémente l'interface [IMainActivity] qui elle même étend l'interface [IDao]. A chaque fois qu'on ajoute une méthode à l'interface [IDao], il faut l'implémenter dans la classe [MainActivity]. La méthode [IDao.getArduinos] ajoutée à l'interface [IDao] sera implémentée de la façon suivante dans [MainActivity] :

```

1. ...
2. @EActivity
3. @OptionsMenu(R.menu.menu_main)
4. public class MainActivity extends AbstractActivity {
5.
6.     // couche [DAO]
7.     @Bean(Dao.class)
8.     protected IDao dao;
9.     // session
10.    private Session session;
11.
12. ...
13.
14.     // implémentation IDao -----
15.     @Override
16.     public Observable<Response<List<Arduino>>> getArduinos() {
17.         return dao.getArduinos();
18.     }
19. }
```

- lignes 15-18 : la méthode [getArduinos] est implémentée en déléguant le travail à la classe [Dao] qu'on vient de présenter et dont on a une référence ligne 8 ;

5.6.14 Le fragment [ConfigFragment] revisité

Dans la classe [ConfigFragment], le code exécuté lors d'un clic sur le bouton [Rafraîchir] est pour l'instant le suivant :

```

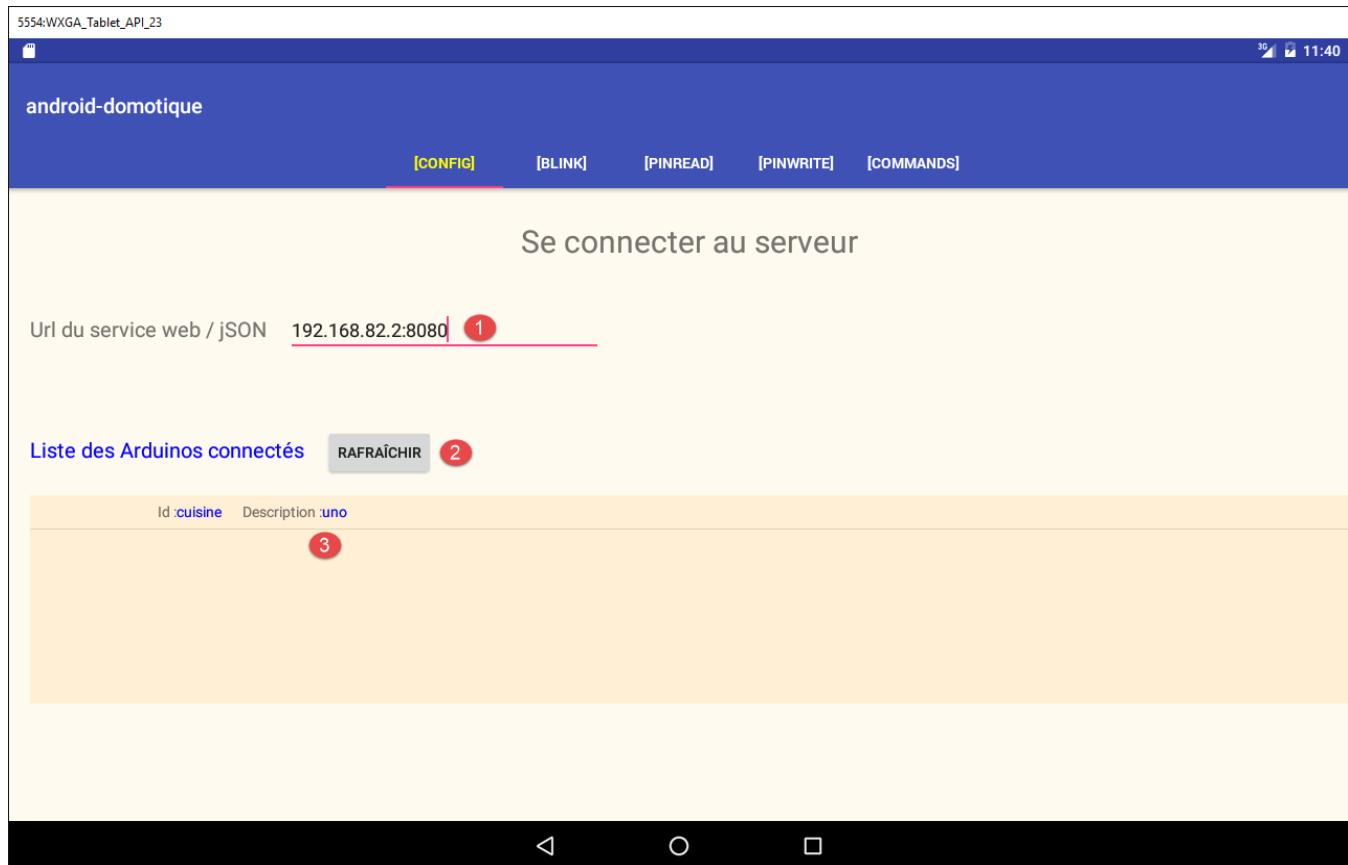
1.     @Click(R.id.btn_Rafraichir)
2.     protected void doRafraichir() {
3.         ...
4.         // on demande la liste des Arduinos en tâche de fond
5.         getArduinosInBackground();
6.     }
7.
8.     private void getArduinosInBackground() {
9.         // on crée une liste d'arduinoss fictive
10.        List<Arduino> arduinos = new ArrayList<>();
11.        for (int i = 0; i < 20; i++) {
12.            arduinos.add(new Arduino("id" + i, "desc" + i, "mac" + i, "ip" + i, i));
13.        }
14.        // on simule une réponse du serveur
15.        Response<List<Arduino>> response = new Response<>();
16.        response.setBody(arduios);
17.        // on la consomme
18.        consumeArduinosResponse(response);
19.    }
20.
21.    // affichage réponse
22.    private void consumeArduinosResponse(Response<List<Arduino>> response) {
23.        ...
24.    }
```

Il nous faut réécrire les lignes 10-16 qui généraient en dur une réponse de type [Response<List<Arduino>>]. Il nous faut désormais demander cette liste à la couche [DAO] via l'activité. Le code devient le suivant :

```
1.  @Click(R.id.btn_Rafraichir)
2.  protected void doRafraichir() {
3.      // on vérifie les saisies
4.      if (!pageValid()) {
5.          return;
6.      }
7.      // on mémorise la saisie
8.      mainActivity.setUrlServiceWebJson(urlServiceRest);
9.      // on prépare l'attente
10.     beginWaiting(1);
11.     // on exécute la tâche asynchrone
12.     executeInBackground(mainActivity.getArduinos(), new Action1<Response<List<Arduino>>>() {
13.
14.         @Override
15.         public void call(Response<List<Arduino>> response) {
16.             // on consomme la réponse
17.             consumeArduinosResponse(response);
18.         }
19.     });
20. }
```

- ligne 8 : l'URL racine du service web / JSON saisie par l'utilisateur est passée à la couche [DAO] via l'activité. Ce sera l'URL racine de l'interface [WebClient] (cf paragraphe 5.6.12.2, page 463) ;
- ligne 10 : on prévient la classe parent qu'on va lancer une tâche asynchrone ;
- lignes 12-19 : lancement de la tâche asynchrone qui va ramener la liste des Arduinos connectés au serveur ;
- ligne 12 : appel de la méthode [executeInBackground] de la classe parent. Cette méthode attend deux paramètres :
 - ligne 12 : le processus à observer. Ce processus est ici fourni par la méthode [mainActivity.getArduinos()] ;
 - lignes 12-19 : une instance de l'interface [Action1<T>], où le type T est le type fourni par le processus, ici un type [Response<List<Arduino>>] ;
- lignes 14-18 : la méthode appelée lorsque la tâche asynchrone rend son résultat de type [Response<List<Arduino>>] ;
- ligne 17 : on passe la réponse reçue à la méthode [consumeArduinosResponse] déjà écrite ;

Travail : Lancez le serveur comme il a été indiqué au paragraphe 5.4, page 421. Connectez un ou plusieurs Arduinos au PC sur lequel le serveur a été lancé. Puis lancez le client Android et vérifiez que vous pouvez bien obtenir la liste des Arduinos connectés. Observez les logs.



- tapez l'URL indiquée en [1]. C'est l'une des adresses IP de votre serveur ;
- cliquez sur le bouton [2] ;
- vous devrez obtenir la liste des Arduinos connectés en [3] ;

Vérifiez que cette liste apparaît également dans les autres onglets.

5.7 Travail à faire

En procédant comme il a été fait pour la vue [Config], réalisez puis testez les quatre autres vues de l'application : [Blink], [PinRead], [PinWrite] et [Commands].

Les vues à réaliser ont été présentées au paragraphe 5.5, page 428.

Pour chaque vue, il faut :

- dessiner la vue XML (cf paragraphe 5.6.9, page 438) ;
- construire le fragment associé (cf paragraphe 5.6.10, page 442) ;
- ajouter une méthode à l'interface [WebClient] (cf paragraphe 5.6.12.2, page 463) ;
- ajouter une méthode à l'interface [IDao] (cf paragraphe 5.6.12.2, page 463) ;
- ajouter une méthode à la classe [Dao] (cf paragraphe 5.6.12.3, page 464) ;
- ajouter une méthode à l'activité [MainActivity] (cf paragraphe 5.6.13, page 465) ;
- écrire les gestionnaires d'événements du fragment (cf paragraphe 5.6.14, page 466) ;
- tester et observer les logs ;

Note 1 : l'exemple à suivre est le projet [Exemple-16B] du cours (cf paragraphe 2.8.3, page 316).

Note 2 : les URL à interroger et le type de leurs réponses ont été présentées au paragraphe 5.4.2, page 423.

Note 3 :

La classe [CommandsFragment] envoie une liste contenant une unique commande à exécuter par un ou plusieurs Arduinos. Cette commande sera encapsulée dans la classe [ArduinoCommand] suivante :

```

1. package android.arduinoss.dao;
2.
3. import java.util.Map;
4.
5. public class ArduinoCommand {
6.
7.     // data
8.     private String id;
9.     private String ac;
10.    private Map<String, Object> pa;
11.
12.    // constructeurs
13.    public ArduinoCommand() {
14.
15.    }
16.
17.    public ArduinoCommand(String id, String ac, Map<String, Object> pa) {
18.        this.id = id;
19.        this.ac = ac;
20.        this.pa = pa;
21.    }
22.
23.    // getters et setters
24.    ...
25. }
```

Dans l'interface [WebClient], la méthode pour exécuter cette liste d'une commande sera la suivante :

```

1.     // envoi de commandes JSON
2.     @Post("/arduinoss/commands/{idArduino}")
3.     Response<List<ArduinoResponse>> sendCommands(@Body List<ArduinoCommand> commands, @Path String idArduino);
```

- ligne 2 : l'URL est demandée avec un ordre HTTP POST ;
- ligne 3 : la valeur postée doit avoir l'annotation [@Body] ;

Note 4 : il est conseillé de faire ce travail de la façon suivante :

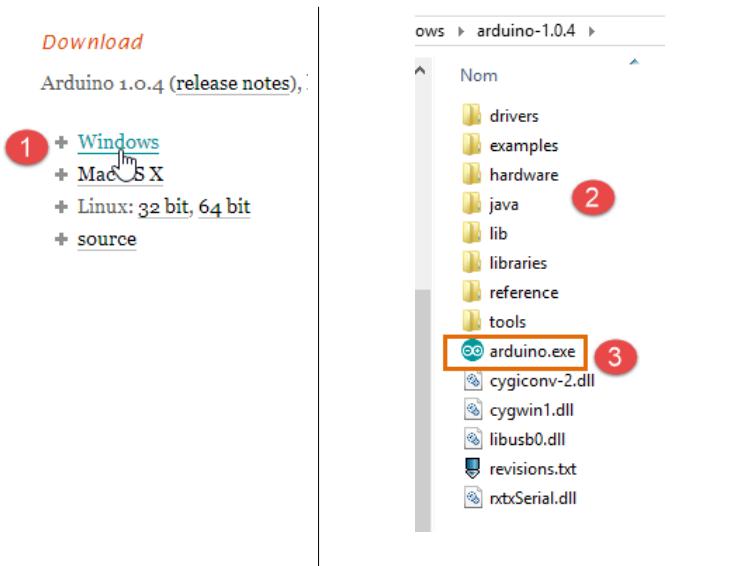
- ne passer à la vue suivante que lorsque la vue courante a été créée et testée ;
- ne gérer l'état des vues qu'après avoir obtenu une application fonctionnelle dans des conditions normales. Ensuite pour chaque vue, faites tourner le périphérique pour différents états de la vue et notez les informations perdues. Ce sont elles qu'il faut sauvegarder puis restaurer. Vérifiez ensuite la navigation : lorsqu'on quitte un onglet et qu'on y revient ultérieurement, on doit le retrouver dans l'état où on l'a laissé ;

6 Annexes

Nous présentons ici comment installer les outils utilisés dans ce document sur des machines windows 7 à 10. Le lecteur s'adaptera à son propre environnement.

6.1 Intallation de l'IDE Arduino

Le site officiel de l'Arduino est [<http://www.arduino.cc/>]. C'est là qu'on trouvera l'IDE de développement pour les Arduinos [<http://arduino.cc/en/Main/Software>] :



Le fichier téléchargé [1] est un zip qui une fois décompressé donne l'arborescence [2]. On pourra lancer l'IDE en double-cliquant sur l'exécutable [3].

6.2 Intallation du pilote (driver) de l'Arduino

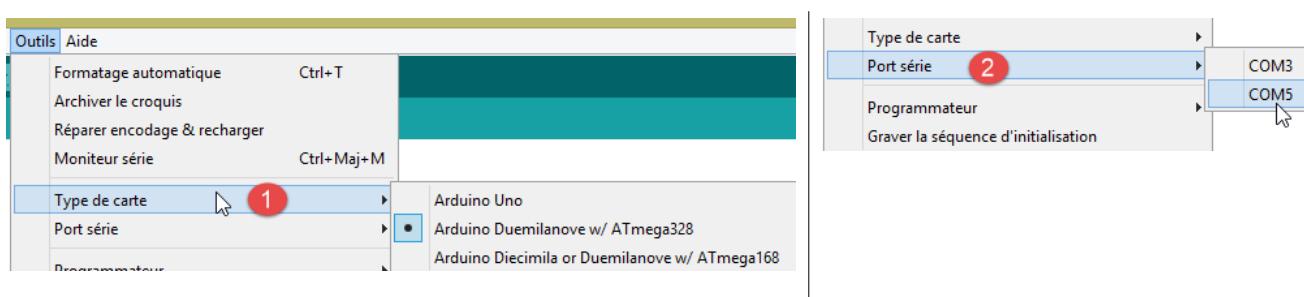
Afin que l'IDE puisse communiquer avec un Arduino, il faut que ce dernier soit reconnu par le PC hôte. Pour cela, on pourra procéder de la façon suivante :

- brancher l'Arduino sur un port USB de l'ordinateur hôte ;
- le système va alors tenter de trouver le pilote du nouveau périphérique USB. Il ne va pas le trouver. Indiquez alors que le pilote est sur le disque à l'endroit <arduino>/drivers où <arduino> est le dossier d'installation de l'IDE Arduino.

6.3 Tests de l'IDE

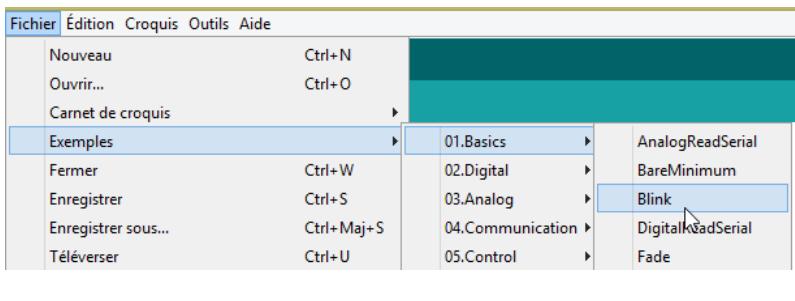
Pour tester l'IDE, on pourra procéder de la façon suivante :

- lancer l'IDE ;
- connecter l'Arduino au PC via son câble USB. Pour l'instant, ne pas inclure la carte réseau ;



- en [1], choisissez le type de la carte Arduino connectée au port USB ;
- en [2], indiquez le port USB sur lequel l'Arduino est connectée. Pour le savoir, débranchez l'Arduino et notez les ports. Rebranchez-le et vérifiez de nouveau les ports : celui qui a été rajouté est celui de l'Arduino ;

Exécutez certains des exemples inclus dans l'IDE :



L'exemple est chargé et affiché :

```

Fichier Édition Croquis Outils Aide
Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.

This example code is in the public domain.
*/

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}

```

Les exemples qui accompagnent l'IDE sont très didactiques et très bien commentés. Un code Arduino est écrit en langage C et se compose de deux parties bien distinctes :

- une fonction **[setup]** qui s'exécute une seule fois au démarrage de l'application, soit lorsque celle-ci est "téléversée" du PC hôte sur l'Arduino, soit lorsque l'application est déjà présente sur l'Arduino et qu'on appuie sur le bouton **[Reset]**. C'est là qu'on met le code d'initialisation de l'application ;
- une fonction **[loop]** qui s'exécute continuellement (boucle infinie). C'est là qu'on met le cœur de l'application.

Ici,

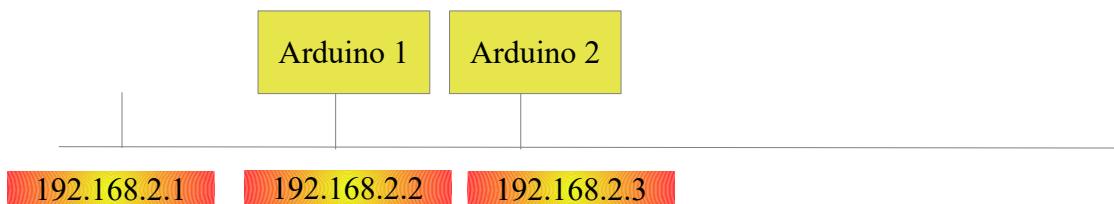
- la fonction **[setup]** configure la pin n° 13 en sortie ;
- la fonction **[loop]** l'allume et l'éteint de façon répétée : la led s'allume et s'éteint toutes les secondes.



Le programme affiché est transféré (téléversé) sur l'Arduino avec le bouton [1]. Une fois transféré, il s'exécute et la led n° 13 se met à clignoter indéfiniment.

6.4 Connexion réseau de l'Arduino

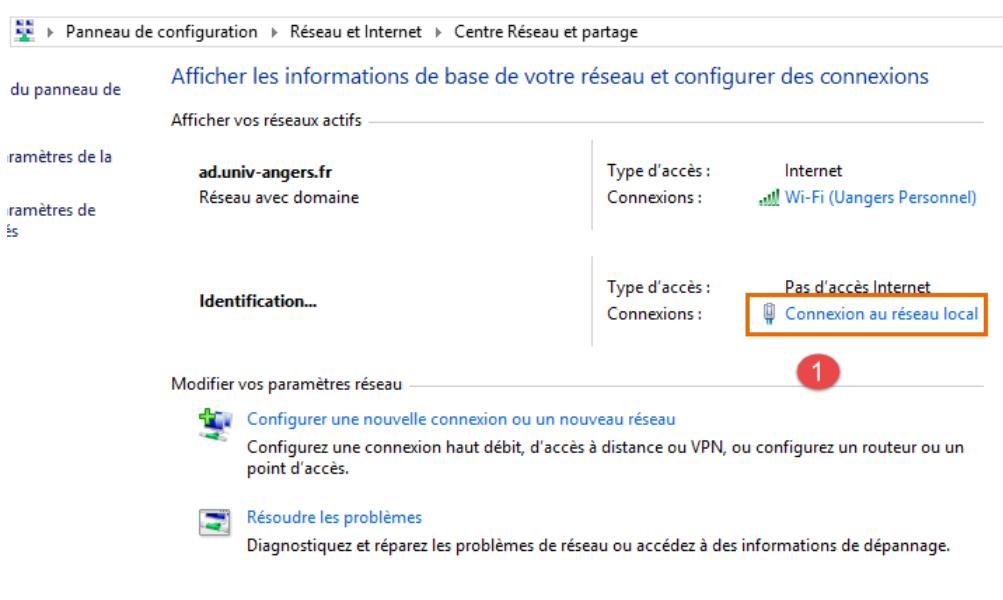
L'Arduino ou les Arduinos et le PC hôte doivent être sur un même réseau privé :



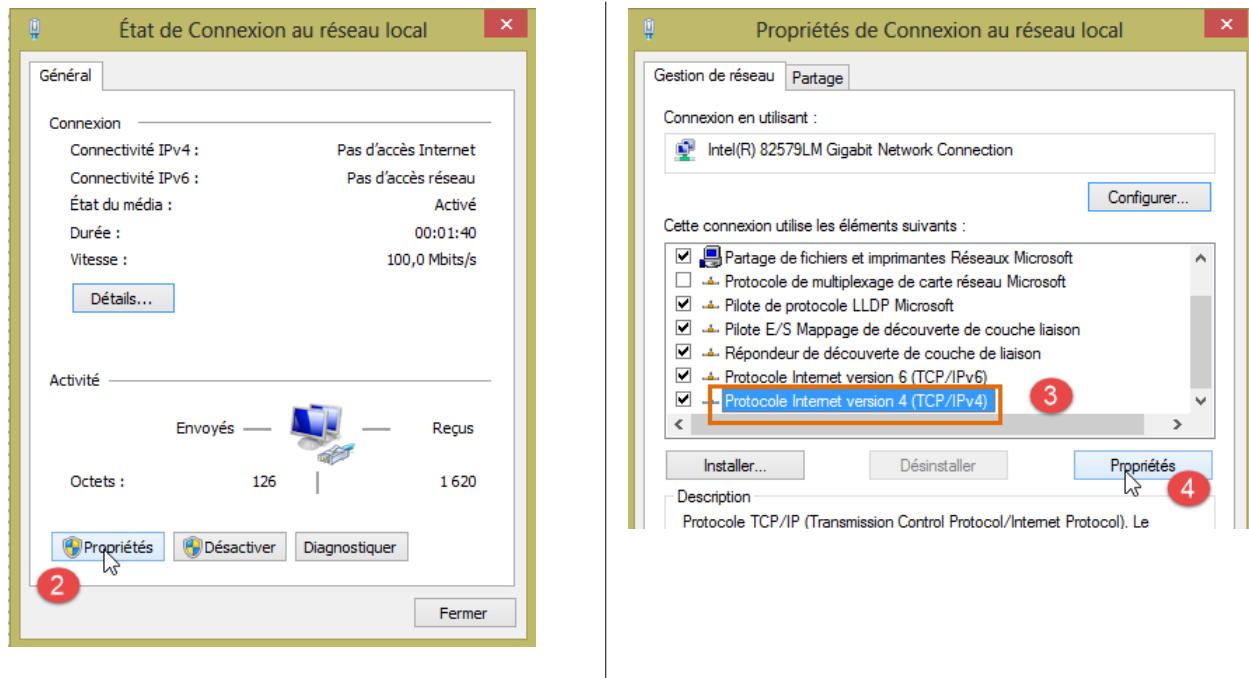
S'il n'y a qu'un Arduino, on pourra le relier au PC hôte par un simple câble RJ 45. S'il y en a plus d'un, le PC hôte et les Arduinos seront mis sur le même réseau par un mini-hub.

On mettra le PC hôte et les Arduinos sur le réseau privé **192.168.2.x**.

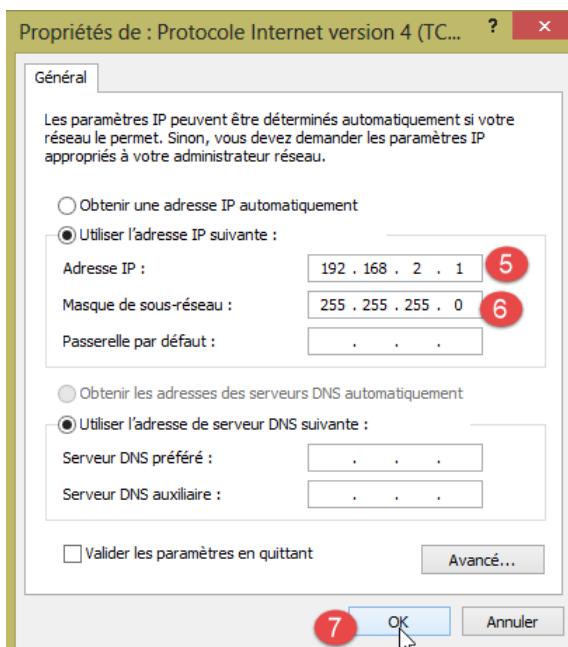
- l'adresse IP des Arduinos est fixée par le code source. Nous verrons comment ;
- l'adresse IP de l'ordinateur hôte pourra être fixée comme suit :
- prendre l'option [Panneau de configuration\Réseau et Internet\Centre Réseau et partage] :



- en [1], suivre le lien [Connexion au réseau local]



- en [2], visualiser les propriétés de la connexion ;
- en [4], visualiser les propriétés IP v4 [3] de la connexion ;



- en [5], donner l'adresse IP [192.168.2.1] à l'ordinateur hôte ;
- en [6], donner le masque [255.255.255.0] au réseau ;
- valider le tout en [7].

6.5 Test d'une application réseau

Avec l'IDE, chargez l'exemple [Exemples / Ethernet / WebServer] :

```
1. /*  
2.  Web Server  
3.
```

```

4. A simple web server that shows the value of the analog input pins.
5. using an Arduino Wiznet Ethernet shield.
6.
7. Circuit:
8. * Ethernet shield attached to pins 10, 11, 12, 13
9. * Analog inputs attached to pins A0 through A5 (optional)
10.
11. created 18 Dec 2009
12. by David A. Mellis
13. modified 9 Apr 2012
14. by Tom Igoe
15.
16. */
17.
18. #include <SPI.h>
19. #include <Ethernet.h>
20.
21. // Enter a MAC address and IP address for your controller below.
22. // The IP address will be dependent on your local network:
23. byte mac[] = {
24.   0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
25. IPAddress ip(192,168,1, 177);
26.
27. // Initialize the Ethernet server library
28. // with the IP address and port you want to use
29. // (port 80 is default for HTTP):
30. EthernetServer server(80);
31.
32. void setup() {
33.   // Open serial communications and wait for port to open:
34.   Serial.begin(9600);
35.   while (!Serial) {
36.     ; // wait for serial port to connect. Needed for Leonardo only
37.   }
38.
39.
40.   // start the Ethernet connection and the server:
41.   Ethernet.begin(mac, ip);
42.   server.begin();
43.   Serial.print("server is at ");
44.   Serial.println(Ethernet.localIP());
45. }
46.
47.
48. void loop() {
49.   // Listen for incoming clients
50.   EthernetClient client = server.available();
51.   if (client) {
52.     Serial.println("new client");
53.     // an http request ends with a blank line
54.     boolean currentLineIsBlank = true;
55.     while (client.connected()) {
56.       if (client.available()) {
57.         char c = client.read();
58.         Serial.write(c);
59.         // if you've gotten to the end of the line (received a newline
60.         // character) and the line is blank, the http request has ended,
61.         // so you can send a reply
62.         if (c == '\n' && currentLineIsBlank) {
63.           // send a standard http response header
64.           client.println("HTTP/1.1 200 OK");
65.           client.println("Content-Type: text/html");
66.           client.println("Connection: close");
67.           client.println();
68.           client.println("<!DOCTYPE HTML>");
69.           client.println("<html>");
70.             // add a meta refresh tag, so the browser pulls again every 5 seconds:
71.           client.println("<meta http-equiv=\"refresh\" content=\"5\">");
72.           // output the value of each analog input pin
73.           for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
74.             int sensorReading = analogRead(analogChannel);
75.             client.print("analog input ");
76.             client.print(analogChannel);
77.             client.print(" is ");
78.             client.print(sensorReading);
79.             client.println("<br />");
80.           }
81.           client.println("</html>");
82.           break;
83.         }
84.         if (c == '\n') {
85.           // you're starting a new line
86.           currentLineIsBlank = true;
87.         }
88.         else if (c != '\r') {
89.           // you've gotten a character on the current line
90.           currentLineIsBlank = false;

```

```

91.     }
92.   }
93.   // give the web browser time to receive the data
94.   delay(1);
95.   // close the connection:
96.   client.stop();
97.   Serial.println("client disconnected");
98. }
99. }
100. }
```

Cette application crée un serveur web sur le port 80 (ligne 30) à l'adresse IP de la ligne 25. L'adresse MAC de la ligne 23 est l'adresse MAC indiquée sur la carte réseau de l'Arduino.

La fonction [setup] initialise le serveur web :

- ligne 34 : initialise le port série sur lequel l'application va faire des logs. Nous allons suivre ceux-ci ;
- ligne 41 : le noeud TCP-IP (IP, port) est initialisé ;
- ligne 42 : le serveur de la ligne 30 est lancé sur ce noeud réseau ;
- lignes 43-44 : on logue l'adresse IP du serveur web ;

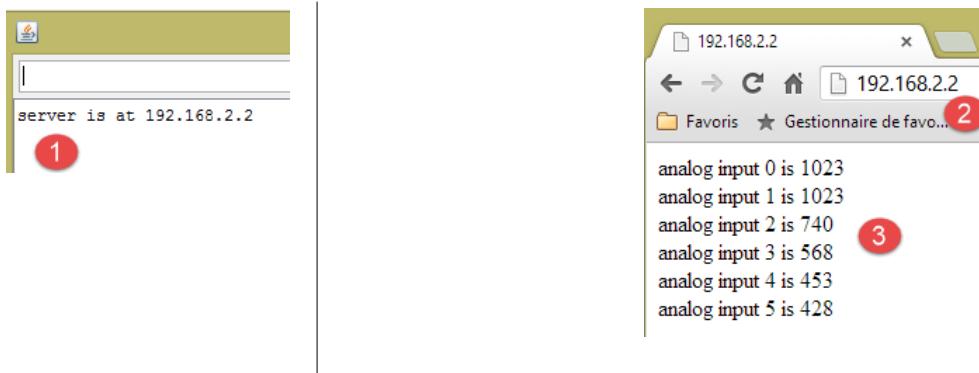
La fonction [loop] implémente le serveur web :

- ligne 50 : si un client se connecte au serveur web `[server].available` rend ce client, sinon rend `null` ;
- ligne 51 : si le client n'est pas `null` ;
- ligne 55 : tant que le client est connecté ;
- ligne 56 : `[client].available` est vrai si le client a envoyé des caractères. Ceux-ci sont stockés dans un buffer. `[client].available` rend vrai tant que ce buffer n'est pas vide ;
- ligne 57 : on lit un caractère envoyé par le client ;
- ligne 58 : ce caractère est affiché en écho sur la console de logs ;
- ligne 62 : dans le protocole HTTP, le client et le serveur échangent des lignes de texte.
 - le client envoie une requête HTTP au serveur web en lui envoyant une série de lignes de texte terminées par une ligne vide,
 - le serveur répond alors au client en lui envoyant une réponse et en fermant la connexion ;
- Ligne 62, le serveur ne fait rien avec les entêtes HTTP qu'il reçoit du client. Il attend simplement la ligne vide : une ligne qui contient le seul caractère `\n` ;
- lignes 64-67 : le serveur envoie au client les lignes de texte standard du protocole HTTP. Elles se terminent par une ligne vide (ligne 67) ;
- à partir de la ligne 68, le serveur envoie un document à son client. Ce document est généré dynamiquement par le serveur et est au format HTML (lignes 68-69) ;
- ligne 71 : une ligne HTML particulière qui demande au navigateur client de rafraîchir la page toutes les 5 secondes. Ainsi le navigateur va demander la même page toutes les 5 secondes ;
- lignes 73-80 : le serveur envoie au navigateur client, les valeurs des 6 entrées analogiques de l'Arduino ;
- ligne 81 : le document HTML est fermé ;
- ligne 82 : on sort de la boucle `while` de la ligne 55 ;
- ligne 97 : la connexion avec le client est fermée ;
- ligne 98 : on logue l'événement dans la console de logs ;
- ligne 100 : on reboucle sur le début de la fonction `loop` : le serveur va se remettre à l'écoute des clients. Nous avons dit que le navigateur client allait redemander la même page toutes les 5 secondes. Le serveur sera là pour lui répondre de nouveau.

Modifiez le code en ligne 25 pour y mettre l'adresse IP de votre Arduino, par exemple :

```
IPAddress ip(192,168,2,2);
```

Téléversez le programme sur l'Arduino. Lancez la console de logs (Ctrl-M) (M majuscule) :



- en [1], la console de logs. Le serveur a été lancé ;
- en [2], avec un navigateur, on demande l'adresse IP de l'Arduino. Ici [192.168.2.2] est traduit par défaut comme [<http://198.162.2.2:80>] ;
- en [3], les informations envoyées par le serveur web. Si on visualise le code source de la page du navigateur, on obtient :

```

1. <!DOCTYPE HTML>
2. <html>
3. <meta http-equiv="refresh" content="5">
4. analog input 0 is 1023<br />
5. analog input 1 is 1023<br />
6. analog input 2 is 727<br />
7. analog input 3 is 543<br />
8. analog input 4 is 395<br />
9. analog input 5 is 310<br />
10. </html>

```

On reconnaît là, les lignes de texte envoyées par le serveur. Du côté Arduino, la console de logs affiche ce que le client lui envoie :

```

1. new client
2. GET /favicon.ico HTTP/1.1
3. Host: 192.168.2.2
4. Connection: keep-alive
5. Accept: */*
6. User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.22 (KHTML, like Gecko) Chrome/25.0.1364.172 Safari/537.22
7. Accept-Encoding: gzip,deflate,sdch
8. Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
9. Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
10.
11. client disconnected

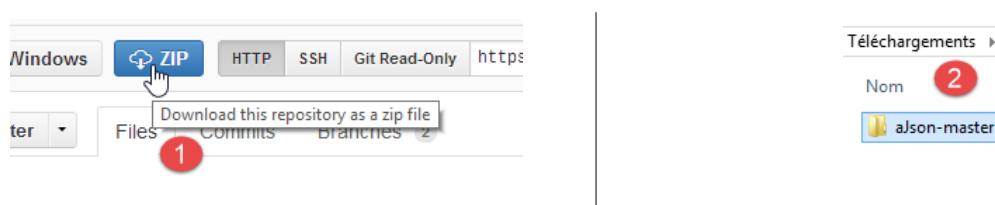
```

- lignes 2-9 : des entêtes HTTP standard ;
- ligne 10 : la ligne vide qui les termine.

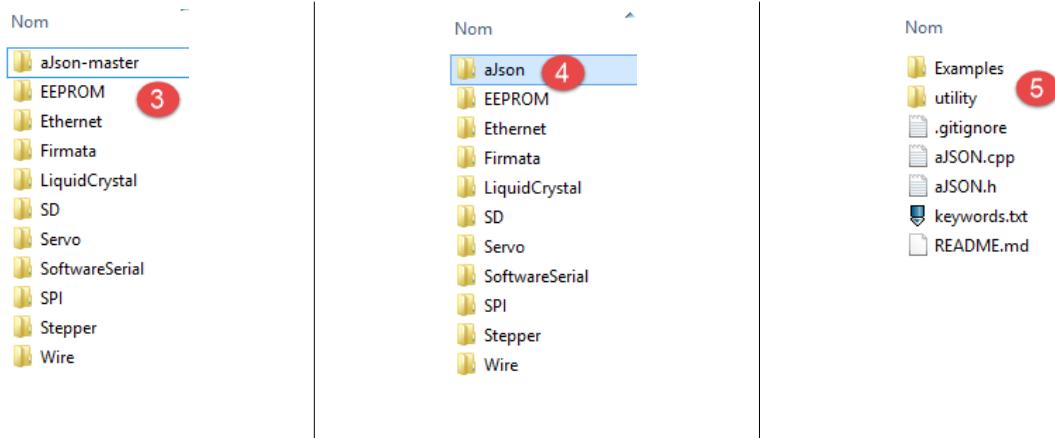
Etudiez bien cet exemple. Il vous aidera à comprendre la programmation de l'Arduino utilisé dans ce TP.

6.6 La bibliothèque aJson

Dans le TP à écrire, les Arduinos échangent des lignes de texte au format JSON avec leurs clients. Par défaut, l'IDE Arduino n'inclut pas de bibliothèque pour gérer le JSON. Nous allons installer la bibliothèque **aJson** disponible à l'URL <https://github.com/interactive-matter/aJson>.

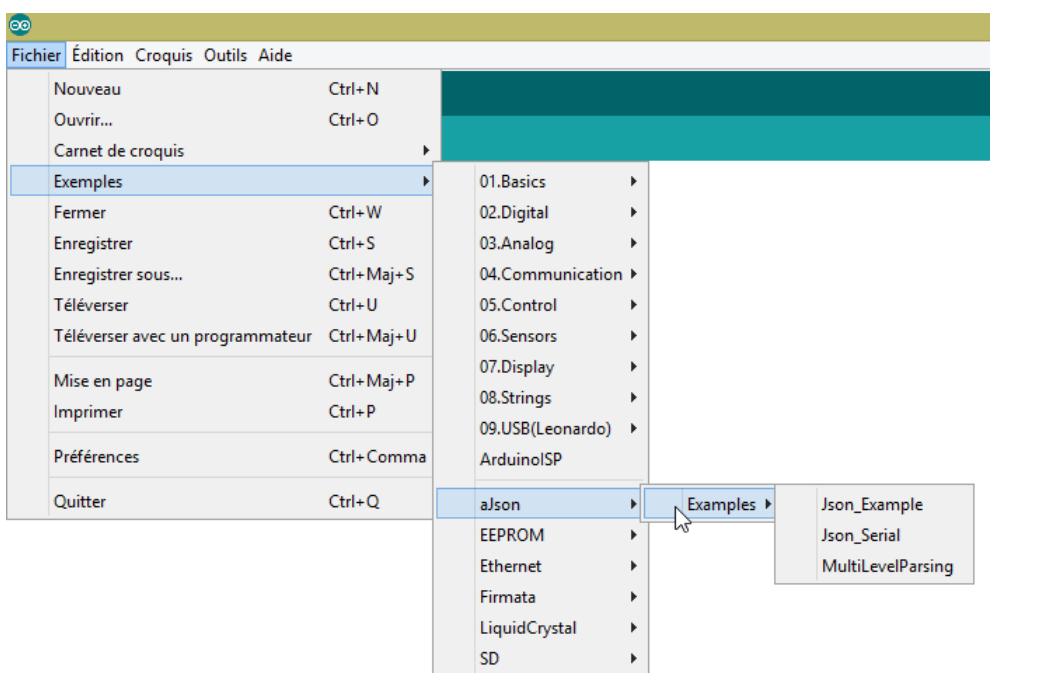


- en [1], téléchargez la version zippée du dépôt Github ;
- décompressez le dossier et copiez le dossier [aJson-master] [2] dans le dossier [<arduino>/libraries] [3] où <arduino> est le dossier d'installation de l'IDE Arduino ;



- en [4], renommez ce dossier [aJson] ;
- en [5], son contenu.

Maintenant, lancez l'IDE Arduino :



Vérifiez que dans les exemples, vous avez désormais des exemples pour la bibliothèque **aJson**. Exécutez et étudiez ces exemples.

6.7 La tablette Android

Les exemples ont été testés avec la tablette Samsung Galaxy Tab 2.

Pour tester les exemples avec une tablette, vous devez d'abord installer le driver de celle-ci sur votre machine de développement. Celui de la tablette Samsung Galaxy Tab 2 peut être trouvé à l'URL [<http://www.samsung.com/fr/support/usefulsoftware/KIES/>] :



TÉLÉCHARGEMENT POUR WINDOWS

Merci de vérifier le nom du modèle de votre produit avant le té

Kies

Modèle compatible : Caractéristique du mobile, Smartphone/ Tablette avant Galaxy Note II, etc.)

KIES DOWNLOAD



Pour tester les exemples, vous aurez besoin de connecter la tablette à un réseau (wifi probablement) et de connaître son adresse IP sur ce réseau. Voici comment procéder (Samsung Galaxy Tab 2) :

- allumez votre tablette ;
 - cherchez dans les applications disponibles sur la tablette (en haut à droite) celle qui s'appelle [paramètres] avec une icône de roue dentée ;
 - dans la section à gauche, activez le wifi ;
 - dans la section à droite, sélectionnez un réseau wifi ;
 - une fois connecté au réseau, faites une frappe courte sur le réseau sélectionné. L'adresse IP de la tablette sera affichée. Notez-la. Vous allez en avoir besoin ;

Toujours dans l'application [Paramètres],

- sélectionnez à gauche l'option [Options de développement] (tout en bas des options) ;
 - vérifiez qu'à droite l'option [Débogage USB] est cochée.

Pour revenir au menu, tapez dans la barre d'état en bas, l'icône du milieu, celle d'une maison. Toujours dans la barre d'état en bas,

- l'icône la plus à gauche est celle du retour en arrière : vous revenez à la vue précédente ;
 - l'icône la plus à droite est celle de la gestion des tâches. Vous pouvez voir et gérer toutes les tâches exécutées à un moment donné par votre tablette ;

Reliez votre tablette à votre PC avec le câble USB qui l'accompagne.

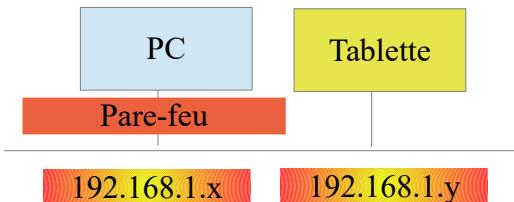
Installez la clé wifi sur l'un des ports USB du PC puis connectez-vous sur le même réseau wifi que la tablette. Ceci fait, dans une fenêtre DOS, tapez la commande `ipconfig` :

```
1. dos>ipconfig
2.
3. Configuration IP de Windows
4.
5. Carte Ethernet Connexion au réseau local :
6.
7.     Suffixe DNS propre à la connexion. . . . :
8.     Adresse IPv6 de liaison locale. . . . .: fe80::698b:455a:925:6b13%4
9.     Adresse IPv4. . . . . . . . . . . . . . .: 192.168.2.1
10.    Masque de sous-réseau. . . . . . . . . . .: 255.255.255.0
11.    Passerelle par défaut. . . . . . . . . .: .
12.
13. Carte réseau sans fil Wi-Fi :
14.
15.     Suffixe DNS propre à la connexion. . . . :
16.     Adresse IPv6 de liaison locale. . . . .: fe80::39aa:47f6:7537:f8e1%2
17.     Adresse IPv4. . . . . . . . . . . . . . .: 192.168.1.25
18.     Masque de sous-réseau. . . . . . . . . .: 255.255.255.0
19.     Passerelle par défaut. . . . . . . . . .: 192.168.1.1
```

Votre PC a deux cartes réseau et donc deux adresses IP :

- celle de la ligne 9 qui est celle du PC sur le réseau filaire ;
- celle de la ligne 17 qui est celle du PC sur le réseau wifi ;

Notez ces deux informations. Vous en aurez besoin. Vous êtes désormais dans la configuration suivante :



La tablette aura à se connecter à votre PC. Celui est normalement protégé par un pare-feu qui empêche tout élément extérieur d'ouvrir une connexion avec le PC. Il vous faut donc inhiber le pare-feu. Faites-le avec l'option [Panneau de configuration\Système et sécurité\Pare-feu Windows]. Parfois il faut de plus inhiber le pare-feu mis en place par l'antivirus. Cela dépend de votre antivirus.

Maintenant, sur votre PC, vérifiez la connexion réseau avec la tablette avec une commande [ping 192.168.1.y] où [192.168.1.y] est l'adresse IP de la tablette. Vous devez obtenir quelque chose qui ressemble à ceci :

```
1. dos>ping 192.168.1.26
2.
3. Envoi d'une requête 'Ping' 192.168.1.26 avec 32 octets de données :
4. Réponse de 192.168.1.26 : octets=32 temps=244 ms TTL=64
5. Réponse de 192.168.1.26 : octets=32 temps=199 ms TTL=64
6. Réponse de 192.168.1.26 : octets=32 temps=28 ms TTL=64
7. Réponse de 192.168.1.26 : octets=32 temps=88 ms TTL=64
8.
9. Statistiques Ping pour 192.168.1.26:
10.    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
11.    Durée approximative des boucles en millisecondes :
12.      Minimum = 28ms, Maximum = 244ms, Moyenne = 139ms
```

Les lignes 4-7 indiquent que la tablette d'adresse IP [192.168.1.y] a répondu à la commande [ping].

6.8 Installation d'un JDK

On trouvera à l'URL [<http://www.oracle.com/technetwork/java/javase/downloads/index.html>] (juin 2016), le JDK le plus récent. On nommera par la suite <jdk-install> le dossier d'installation du JDK.

Java SE Downloads

Java Platform (JDK) 8u91 / 8u92

Java SE 8u91 includes important security fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release. Java SE 8u92 is a patch-set update, including all of 8u91 plus additional features (described in the release notes). [Learn more](#)

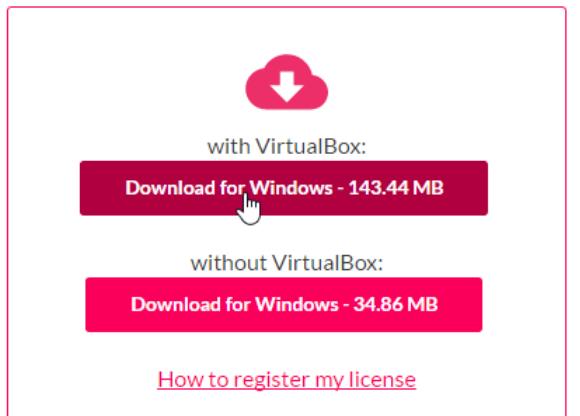
• Installation Instructions
• Release Notes

JDK DOWNLOAD

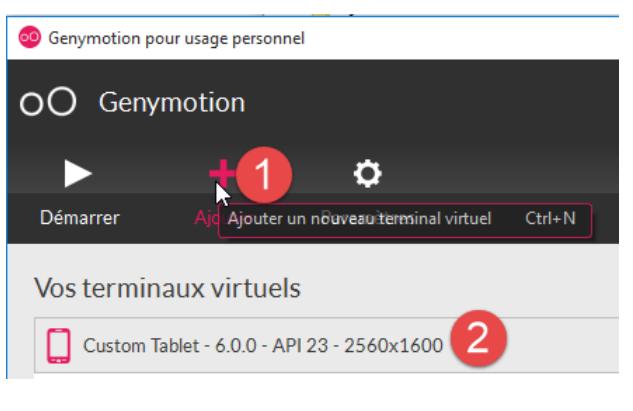
6.9 Installation du gestionnaire d'émulateurs Genymotion

L'entreprise [Genymotion] offre un émulateur Android performant. Celui-ci est disponible à l'URL [<https://cloud.genymotion.com/page/launchpad/download/>] (juin 2016).

Vous aurez à vous enregistrer pour obtenir une version à usage personnel. Téléchargez le produit [Genymotion] avec la machine virtuelle VirtualBox :



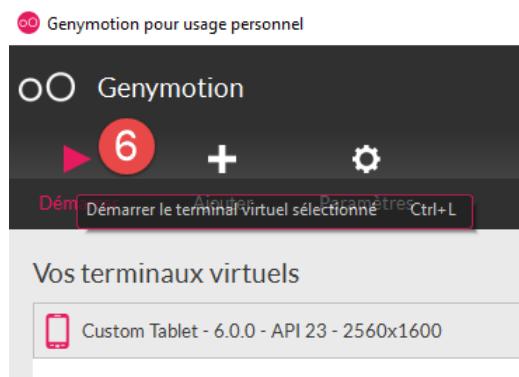
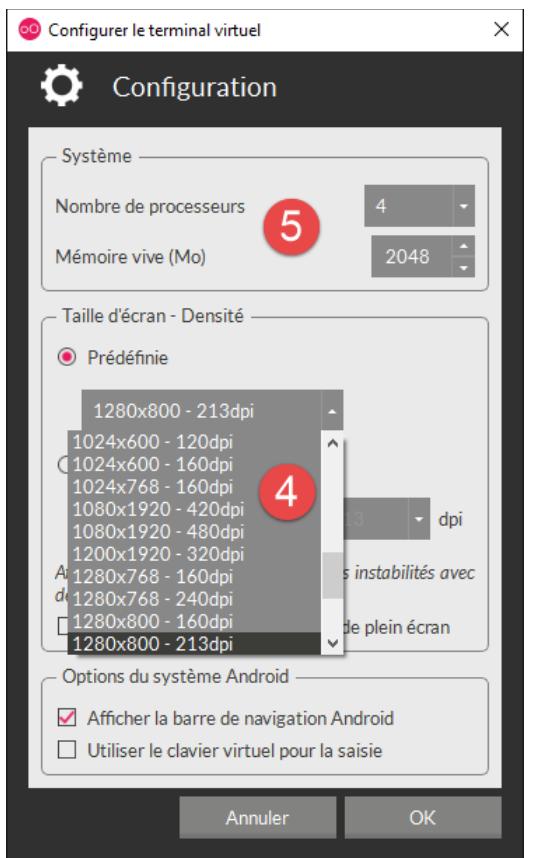
Nous appellerons par la suite <genymotion-install> le dossier d'installation de [Genymotion]. Lancez [Genymotion]. Téléchargez ensuite une image pour une tablette :



- en [1], ajoutez le terminal virtuel décrit en [2] ;

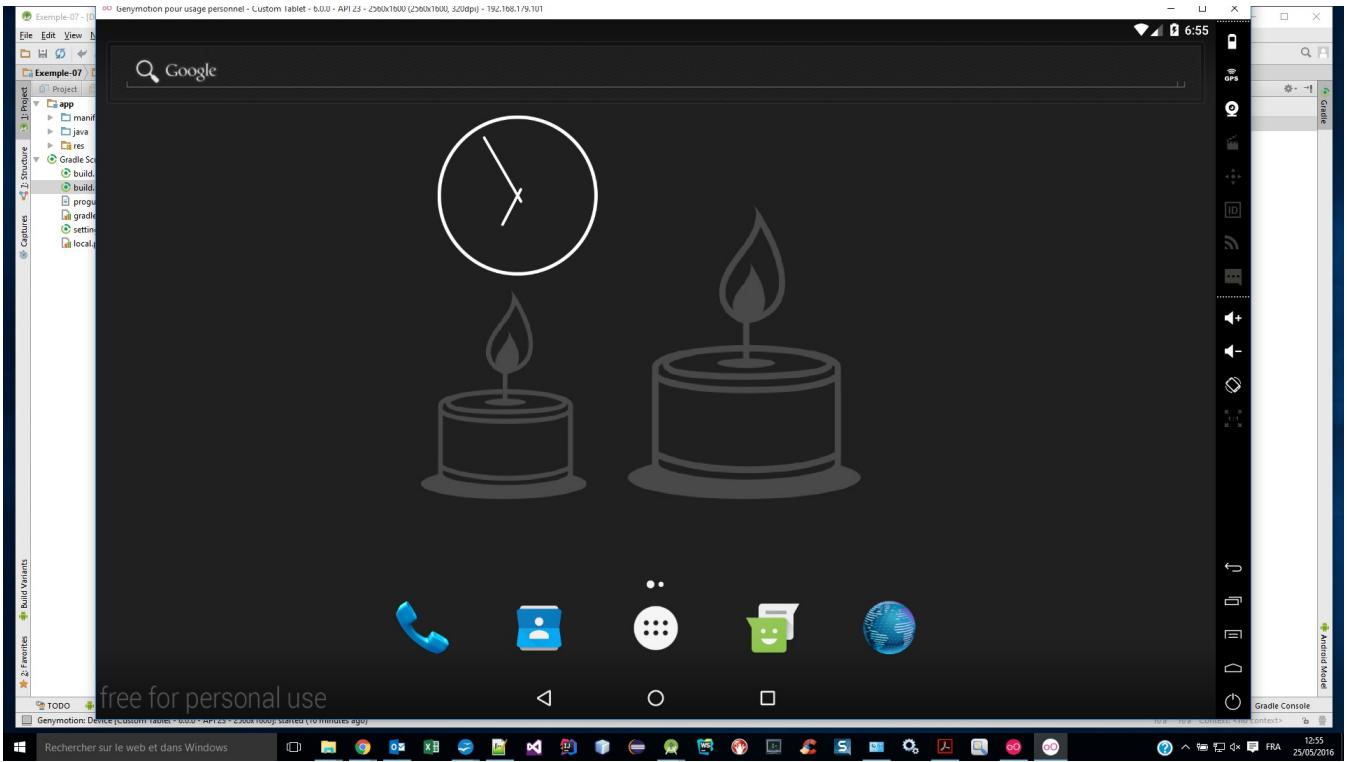


- en [3], configurez le terminal ;



- en [4-5], personnalisez le terminal pour votre environnement ;
- en [6], lancez le terminal virtuel ;

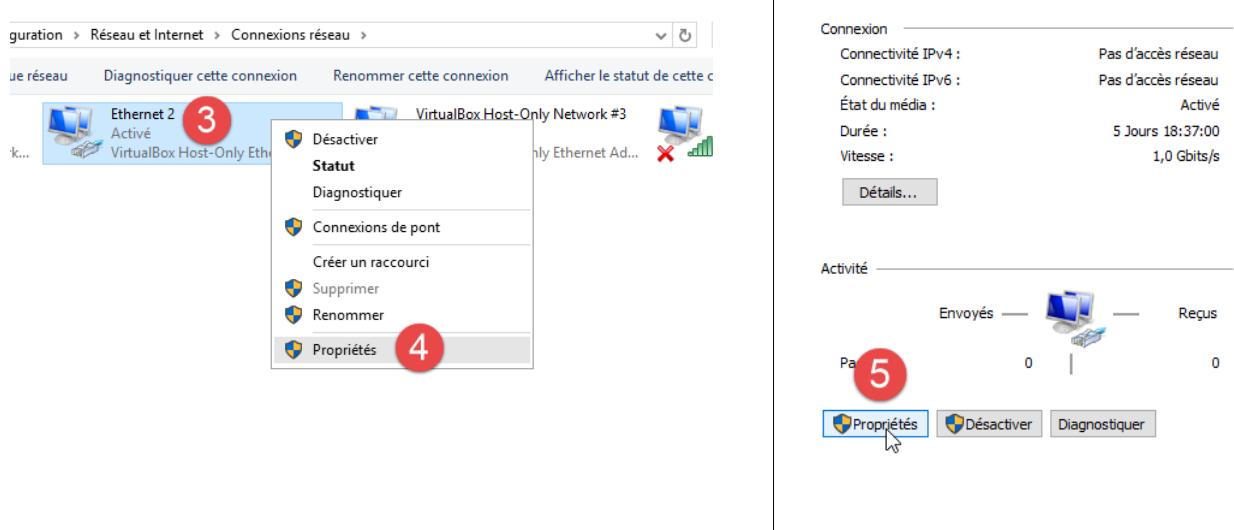
Si tout va bien, on obtient la fenêtre de l'émulateur Android :



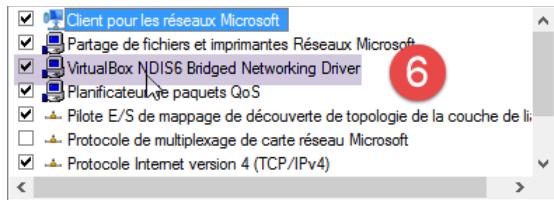
Parfois, l'émulateur Android n'est pas lancé. Avec une machine windows, on pourra regarder les deux points suivants :

- vérifier que la machine virtuelle [Hyper-V] n'est pas installée. Si besoin est, la désinstaller [1-2] ;

- puis dans l'assistant de configuration [Centre Réseau et partage] [1] :



- en [3], sélectionnez la ou les cartes associées à la machine virtuelle Virtual Box ;



- vérifiez qu'en [6], le pilote pour VirtualBox est bien coché. Refaites l'opération pour toutes les cartes associées à la machine virtuelle Virtual Box ;

6.10 Installation de Maven

Maven est un outil de gestion des dépendances d'un projet Java et plus encore. Il est disponible à l'URL [<http://maven.apache.org/download.cgi>].



Téléchargez et dézippez l'archive. Nous appellerons <maven-install> le dossier d'installation de Maven.



- en [1], le fichier [conf / settings.xml] configure Maven ;

On y trouve les lignes suivantes :

```
1.  <!-- localRepository
2.    | The path to the local repository maven will use to store artifacts.
3.    |
4.    | Default: ${user.home}/.m2/repository
5.  <localRepository>/path/to/local/repo</localRepository>
6.  -->
```

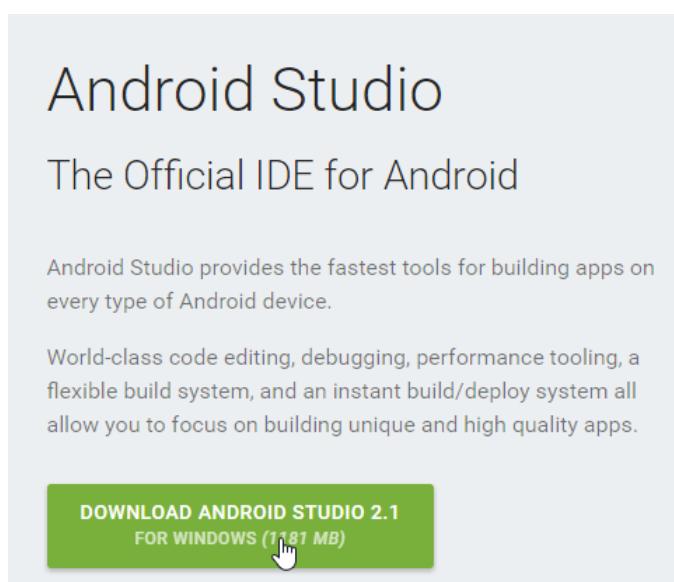
La valeur par défaut de la ligne 4, si comme moi votre {user.home} a un espace dans son chemin (par exemple [C:\Users\Serge Tahé]), peut poser problème à certains logiciels. On écrira alors quelque chose comme :

```
1.  <!-- localRepository
2.    | The path to the local repository maven will use to store artifacts.
3.    |
4.    | Default: ${user.home}/.m2/repository
5.  <localRepository>/path/to/local/repo</localRepository>
6.  -->
7.  <localRepository>D:\Programs\devjava\maven\.m2\repository</localRepository>
```

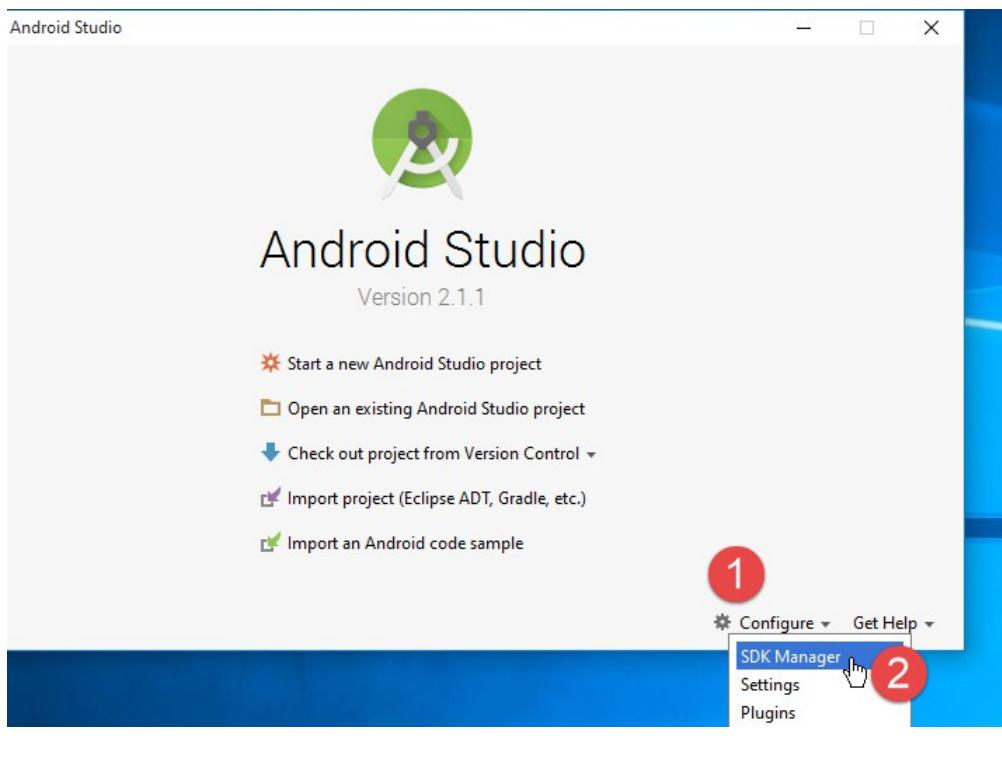
et on évitera, ligne 7, un chemin qui contient des espaces.

6.11 Installation de l'IDE Android Studio

L'IDE Android Studio Community Edition est disponible à l'URL [<https://developer.android.com/studio/index.html>] (juin 2016) :



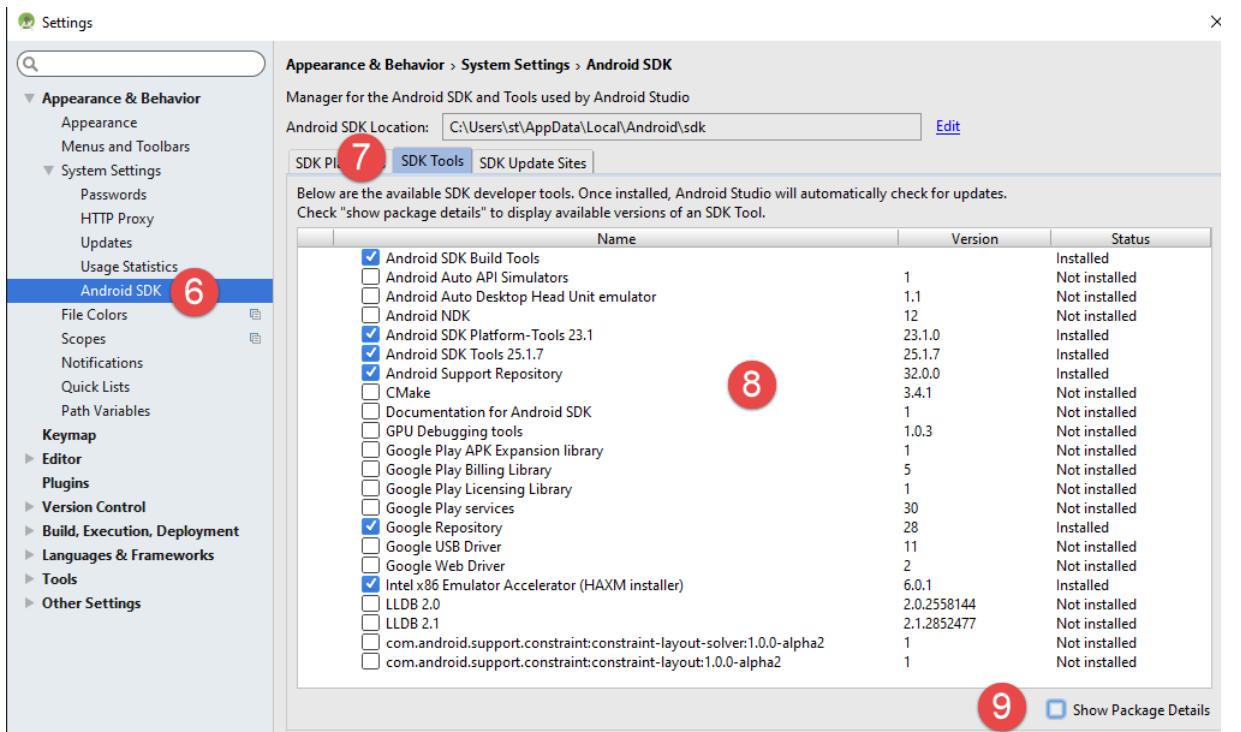
Installez l'IDE puis lancez-le. En suivant la procédure [1-8], installez les éléments du SDK Manager utilisés par les exemples qui vont suivre. Si vous décidez d'installer des éléments plus récents, vous aurez probablement des avertissements d'Android Studio comme quoi la configuration des exemples référence des éléments du SDK qui n'existent pas dans votre environnement. Vous pourrez alors suivre les suggestions faites par l'IDE.



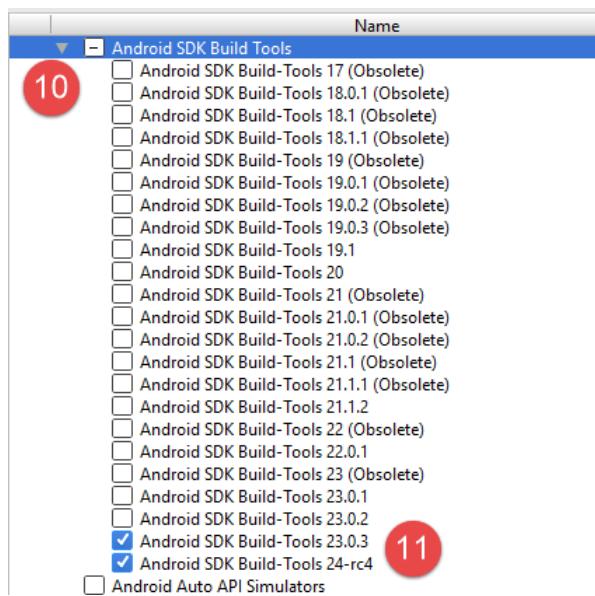
The screenshot shows the 'Default Settings' screen in the Android Studio settings. On the left, there's a sidebar with sections like 'Appearance & Behavior' (3) and 'System Settings'. In the main area, it shows 'Appearance & Behavior > System Settings > Android SDK'. It displays the 'Manager for the Android SDK and Tools used by Android Studio' and the 'Android SDK Location: C:\Users\st\AppData\Local\Android\sdk'. There are tabs for 'SDK Platforms' (4), 'SDK Tools', and 'SDK Update Sites'. The 'SDK Platforms' tab is selected. A table lists available platforms:

Name	API Level
<input type="checkbox"/> Android N Preview	N
<input checked="" type="checkbox"/> Android 6.0 (Marshmallow)	23
<input type="checkbox"/> Android 5.1 (Lollipop)	22
<input type="checkbox"/> Android 5.0 (Lollipop)	21

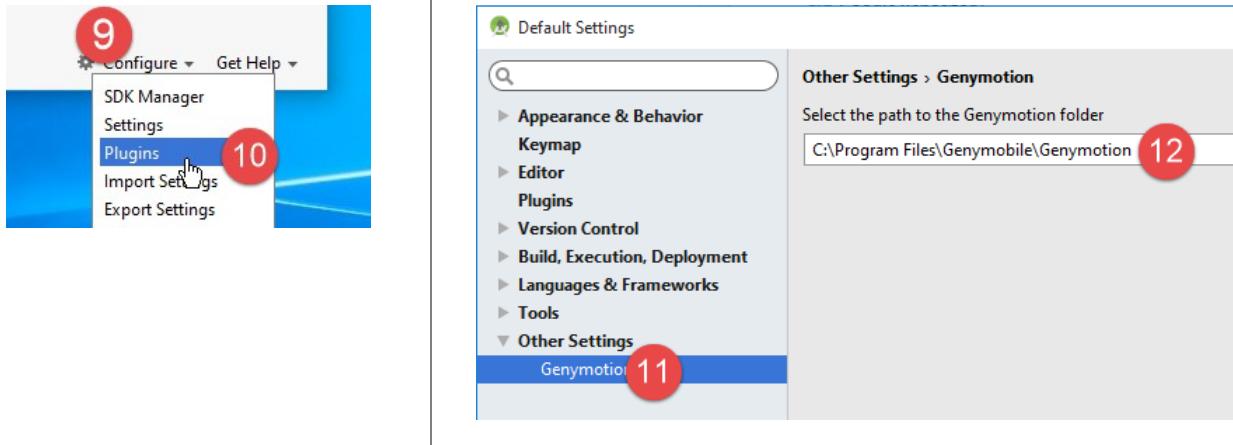
The 'Android 6.0 (Marshmallow)' row is highlighted with a red circle (5).



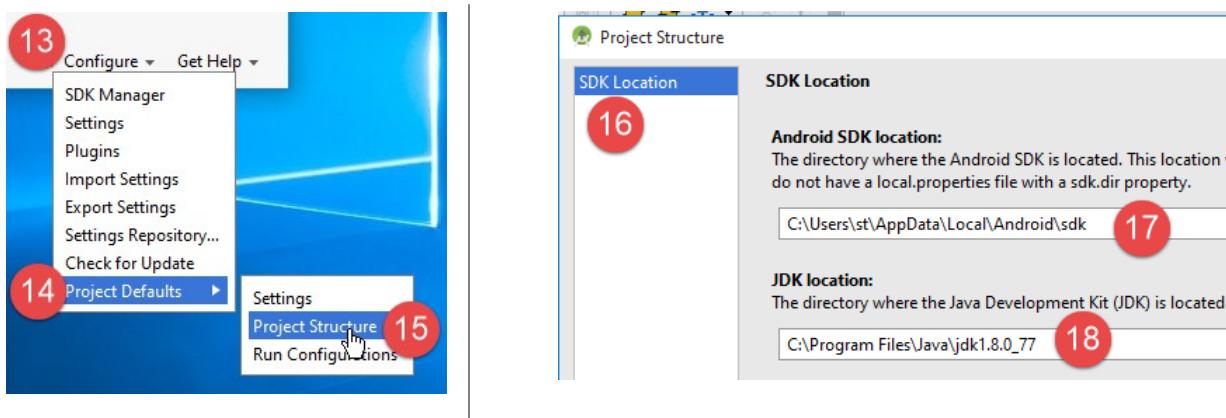
- en [9], demandez à voir le détail des packages :



- en [11] ci-dessus, les exemples ont utilisé le SDK Build-Tools 23.0.3 ;
- en [9-12] ci-dessous, indiquez le dossier où vous avez installé le gestionnaire d'émulateurs [Genymotion] ;

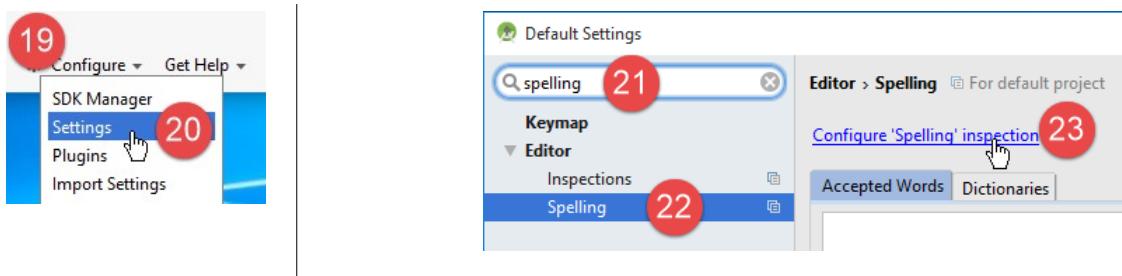


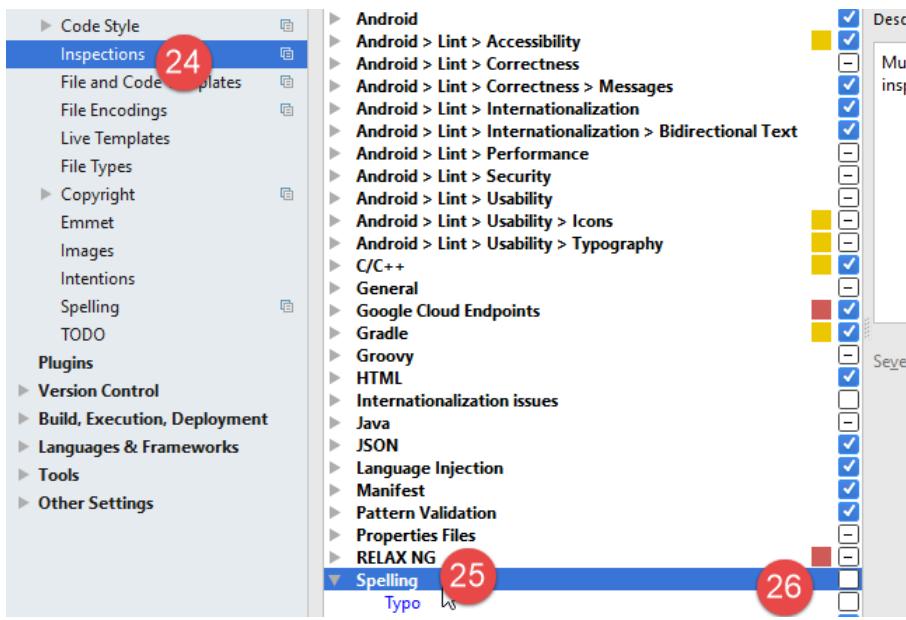
- en [13-18], on configure la nature par défaut des projets ;



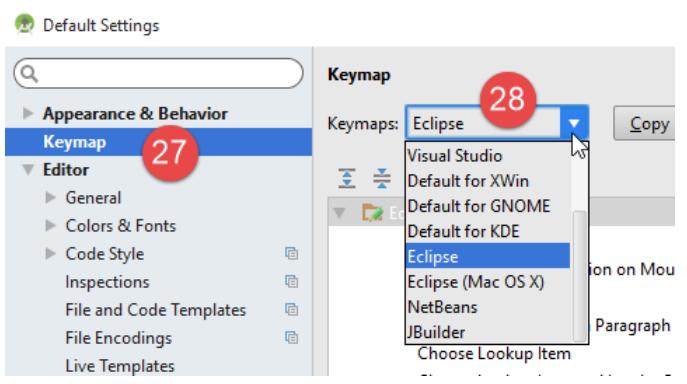
- en [17], la valeur proposée par défaut est normalement la bonne;
- en [18], assurez-vous d'avoir un JDK 1.8;

Ci-dessous, en [19-26], on désactive la correction orthographique qui par défaut est pour la langue anglaise ;

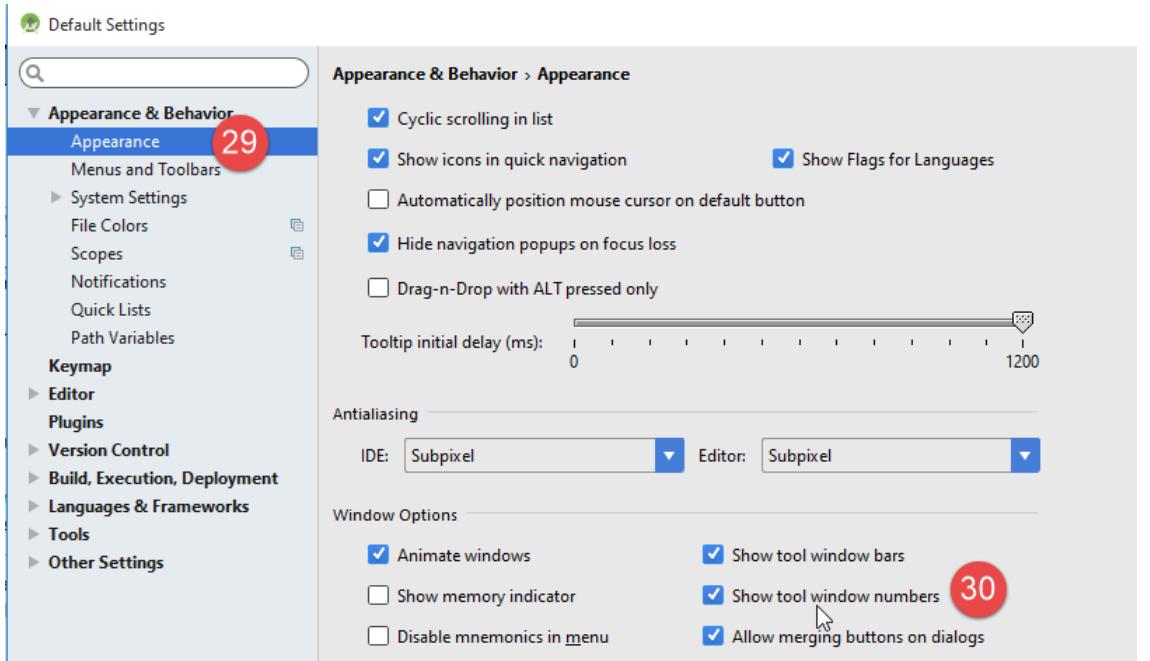




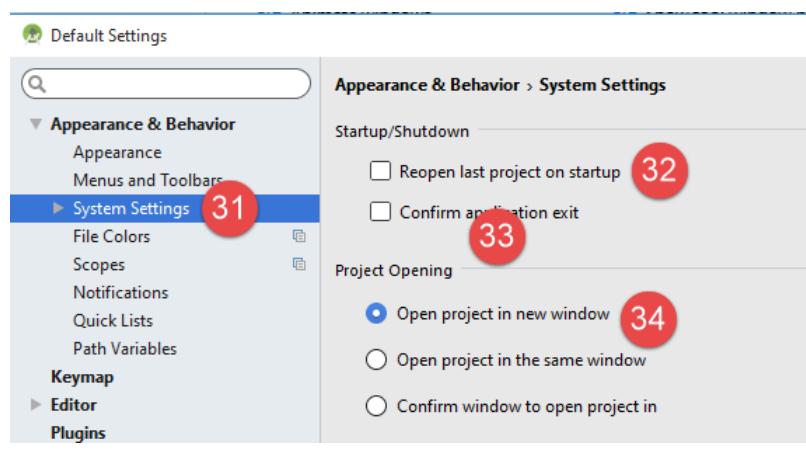
- ci-dessous, en [27-28], choisissez le type de raccourci clavier que vous souhaitez. Vous pouvez garder celui par défaut d'IntelliJ ou choisir celui d'un autre IDE auquel vous seriez plus habitués ;



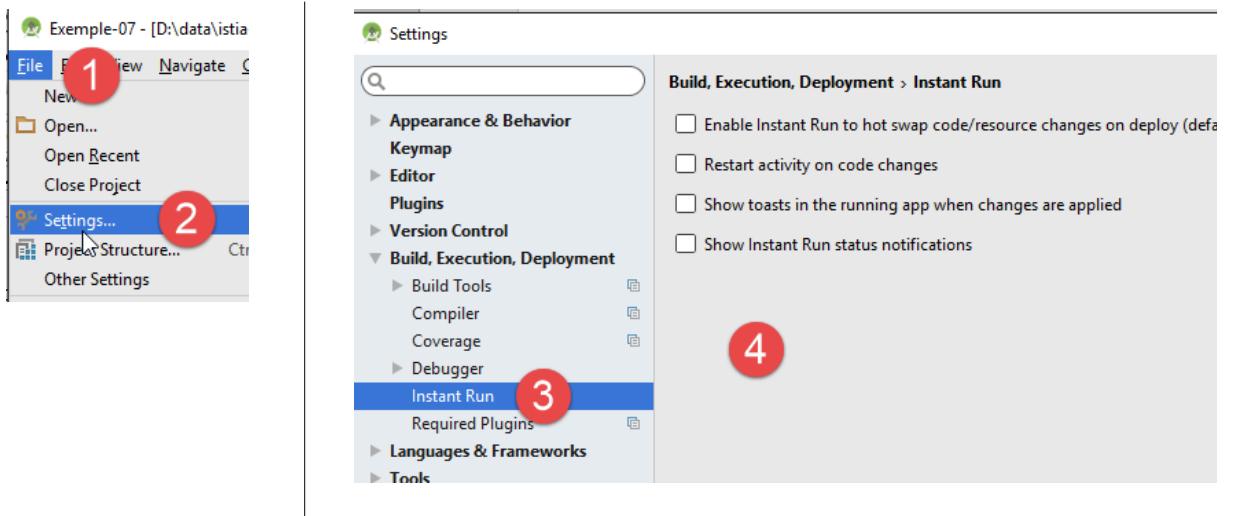
- ci-dessous, en [29-30], faites afficher les n°s de lignes du code;



- ci-dessous, en [31-34], indiquez comment vous souhaitez gérer le 1er projet au lancement de l'IDE puis les projets suivants;



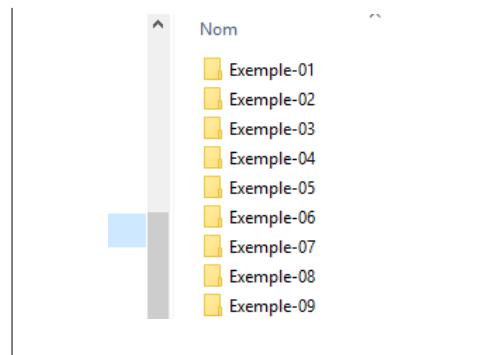
Avec Android 2.1 (mai 2016), la technologie [Instant Run] pose parfois des problèmes. Dans ce document, nous l'avons inhibée :



- en [3-4], tout a été désactivé ;

6.12 Utilisation des exemples

Les projets Android Studio des exemples sont disponibles à l'URL [<http://tahe.developpez.com/tutoriels-cours/programmation-android-avec-android-studio-debutant/documents/dvp-android-studio.rar>]. Téléchargez-les.



Les exemples ont été construits avec les éléments définis précédemment :

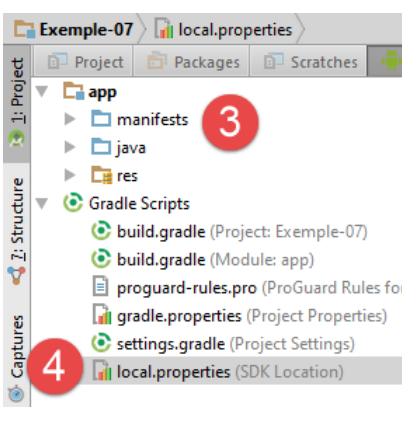
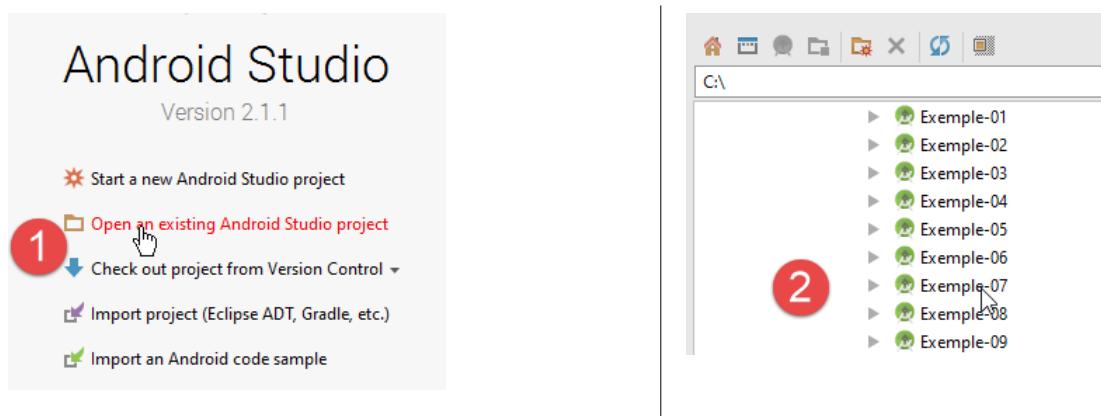
- JDK 1.8 ;
- Android SDK Platform 23 pour l'exécution;

Les outils du SDK suivants :

Android SDK Manager				
	Name	API	Rev.	Status
SDK Path: C:\Users\st\AppData\Local\Android\sdk				
	Packages			
	Name	API	Rev.	Status
✓	Tools			
	Android SDK Tools	25.1.6	25.1.6	Installed
	Android SDK Platform-tools	23.1	23.1	Installed
	Android SDK Build-tools	23.0.3	23.0.3	Installed
	Android SDK Build-tools	23.0.2	23.0.2	Installed
	Android SDK Build-tools	23.0.1	23.0.1	Not installed

Si votre environnement ne correspond pas au précédent, vous aurez à changer la configuration des projets. Cela peut être assez pénible. Dans un premier temps, il est probablement plus facile de reconstituer un environnement de travail analogue à celui ci-dessus.

Lancez Android Studio puis ouvrez le projet [exemple-07] par exemple :

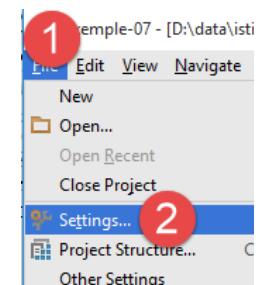


- en [1-3], on ouvre le projet [Exemple-07] ;
- en [4], on vérifie le fichier [local.properties] ;

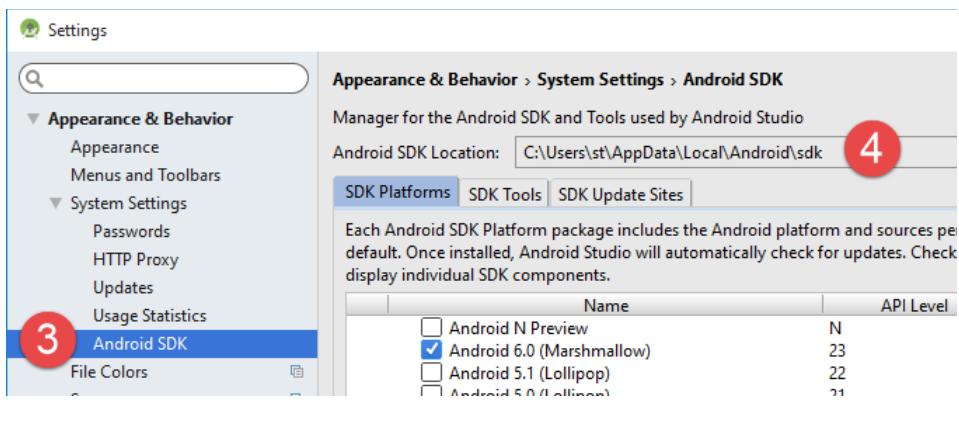
```

local.properties x
1 ## This file is automatically generated by Android Studio.
2 # Do not modify this file -- YOUR CHANGES WILL BE ERASED!
3 #
4 # This file must *NOT* be checked into Version Control Systems,
5 # as it contains information specific to your local configuration.
6 #
7 # Location of the SDK. This is only used by Gradle.
8 # For customization when using a Version Control System, please read the
9 # header note.
10 #Wed May 25 11:21:00 CEST 2016
11 sdk.dir=C:\\\\Users\\\\st\\\\AppData\\\\Local\\\\Android\\\\sdk
12

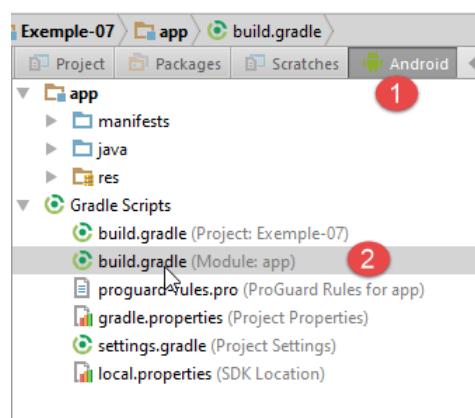
```



- ligne 11 ci-dessus, mettez l'emplacement du SDK Manager d'Android <sdk-manager-install>. Vous pouvez trouver celui-ci en suivant la procédure [1-4] :



Tous les exemples de ce document sont des projets Gradle configurés par un fichier [build.gradle] [1-2]:



Le fichier [build.gradle] de l'exemple 07 est le suivant :

```

1. buildscript {
2.     repositories {
3.         mavenCentral()
4.         mavenLocal()
5.     }
6.     dependencies {
7.         // replace with the current version of the Android plugin
8.         classpath 'com.android.tools.build:gradle:2.1.0'
9.         // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
10.        classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
11.    }
12. }
13.
14. apply plugin: 'com.android.application'
15. apply plugin: 'android-apt'
16. def AAVersion = '4.0.0'
17.
18. dependencies {
19.     apt "org.androidannotations:androidannotations:$AAVersion"
20.     compile "org.androidannotations:androidannotations-api:$AAVersion"
21.     compile 'com.android.support:appcompat-v7:23.4.0'
22.     compile 'com.android.support:design:23.4.0'
23.     compile fileTree(dir: 'libs', include: ['*.jar'])
24. }
25.
26. repositories {
27.     jcenter()
28. }
29.
30. apt {
31.     arguments {
32.         androidManifestFile variant.outputs[0].processResources.manifestFile
33.         resourcePackageName android.defaultConfig.applicationId
34.     }
35. }
```

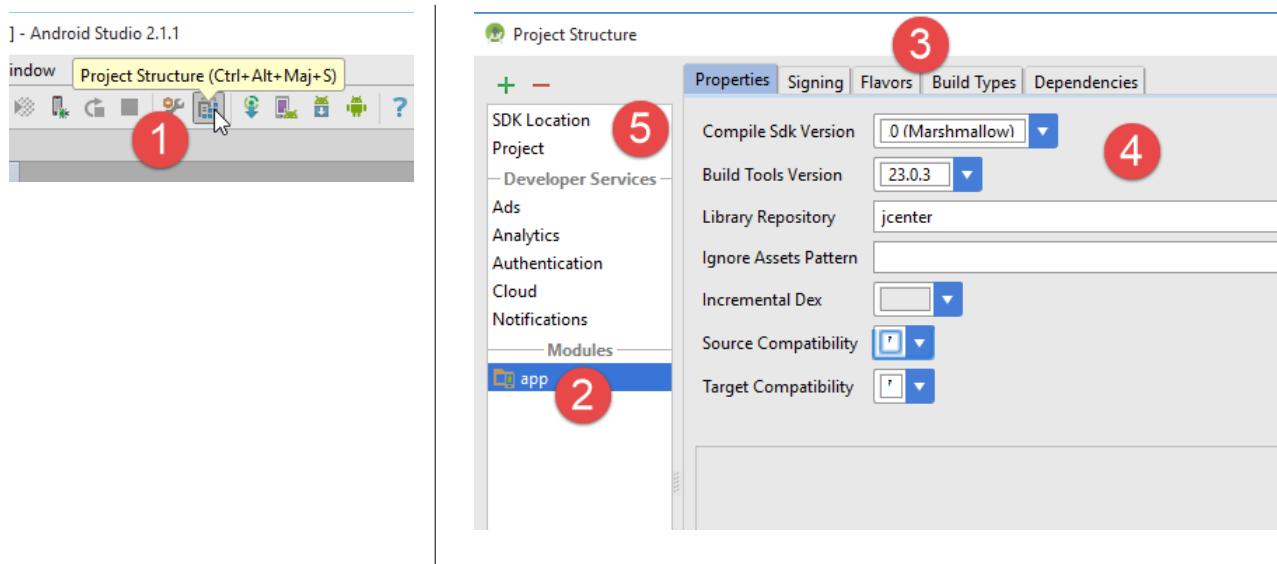
```

36.
37. android {
38.     compileSdkVersion 23
39.     buildToolsVersion "23.0.3"
40.     defaultConfig {
41.         applicationId "android.exemples"
42.         minSdkVersion 15
43.         targetSdkVersion 23
44.         versionCode 1
45.         versionName "1.0"
46.     }
47.
48.     buildTypes {
49.         release {
50.             minifyEnabled false
51.             proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
52.         }
53.     }
54.
55.     compileOptions {
56.         sourceCompatibility JavaVersion.VERSION_1_7
57.         targetCompatibility JavaVersion.VERSION_1_7
58.     }
59. }

```

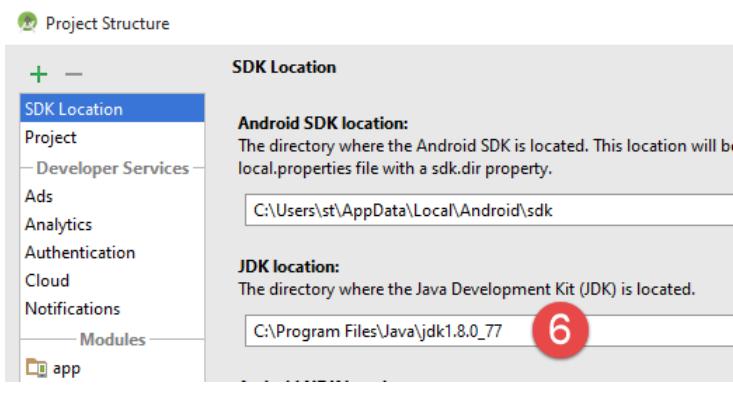
Selon l'environnement Android (SDK et outils) que vous avez construit, vous pouvez être amené à changer les versions des lignes 8, 21, 22, 38, 39. Android Studio vous aide en faisant des suggestions. Le plus simple est de suivre celles-ci.

Les éléments du fichier [build.gradle] sont accessibles d'une autre façon:



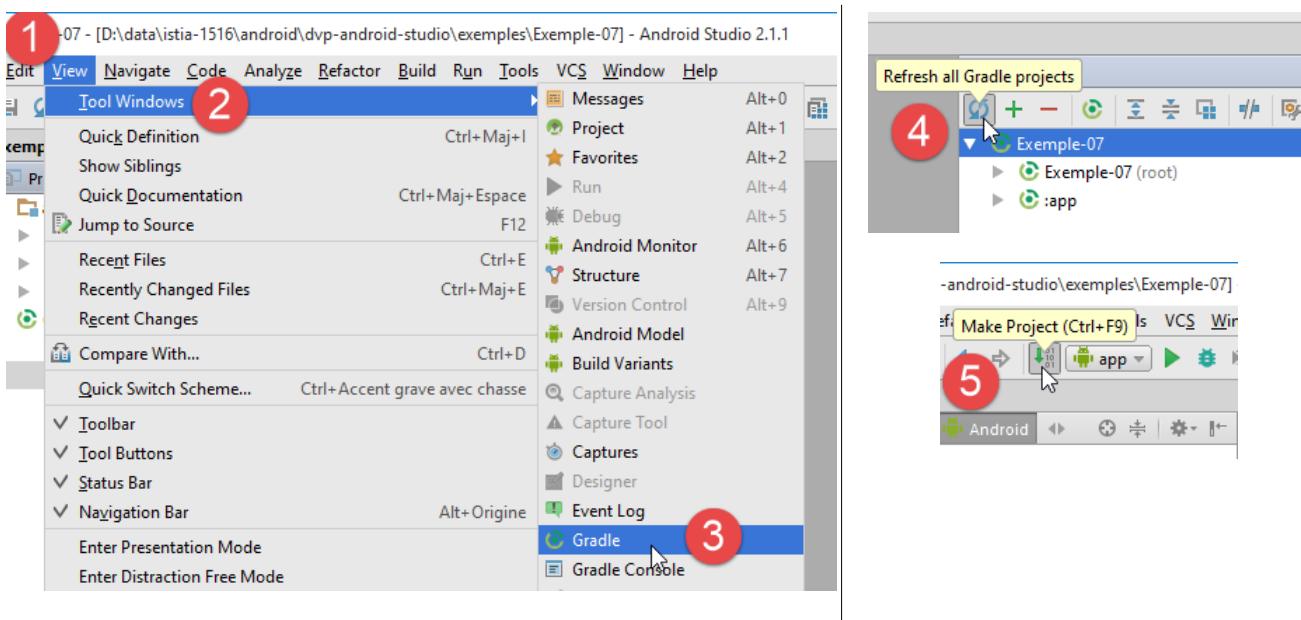
- les différents onglets de [3] reprennent les différentes valeurs du fichier [build.gradle]. On peut donc construire celui-ci de cette façon qui permet de s'affranchir des problèmes de syntaxe du fichier [build.gradle];

Un autre point à vérifier est le JDK utilisé par l'IDE [5] :



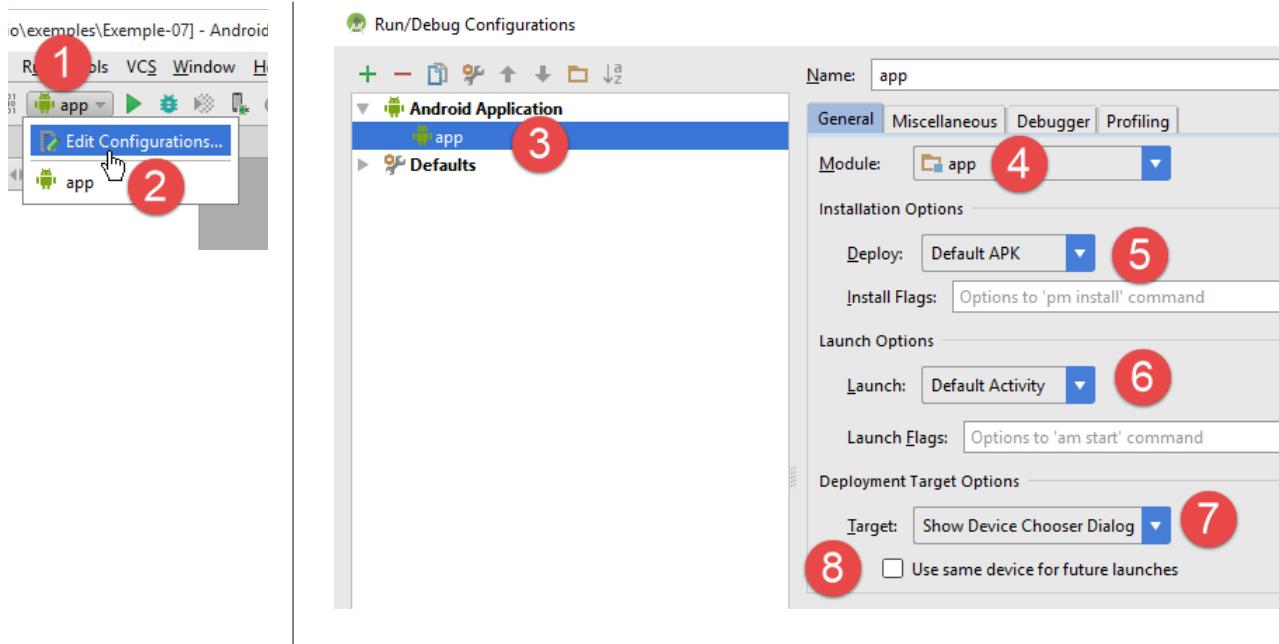
En [6], vérifiez le JDK.

Tous les exemples sont des projets Gradle ayant des dépendances à télécharger. Pour ce faire, on pourra procéder ainsi :

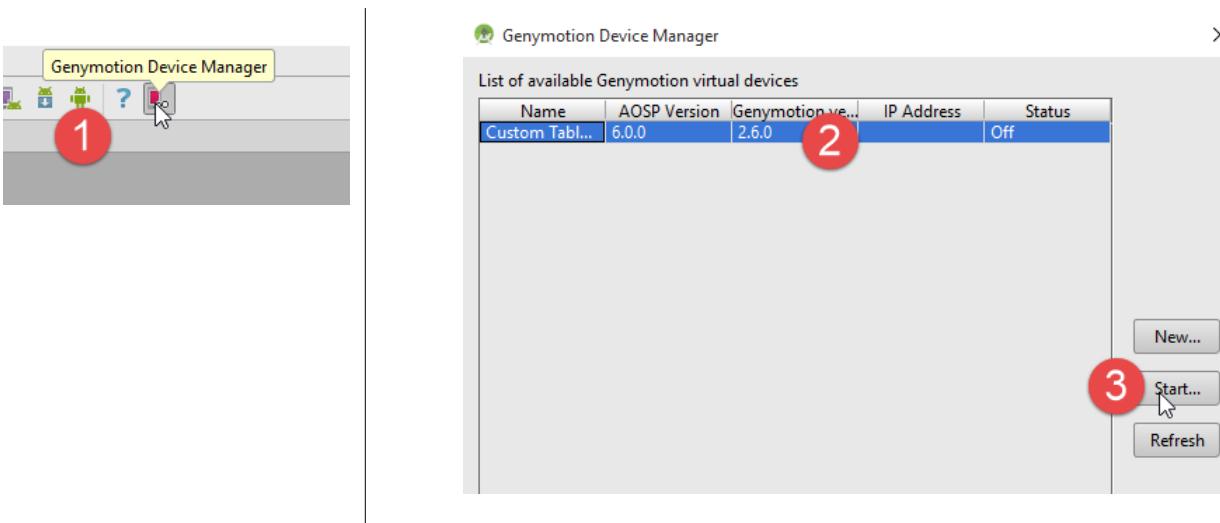


- en [5], on compile le projet ;

Une fois le projet sans erreurs, il faut créer une configuration d'exécution [1-8] :

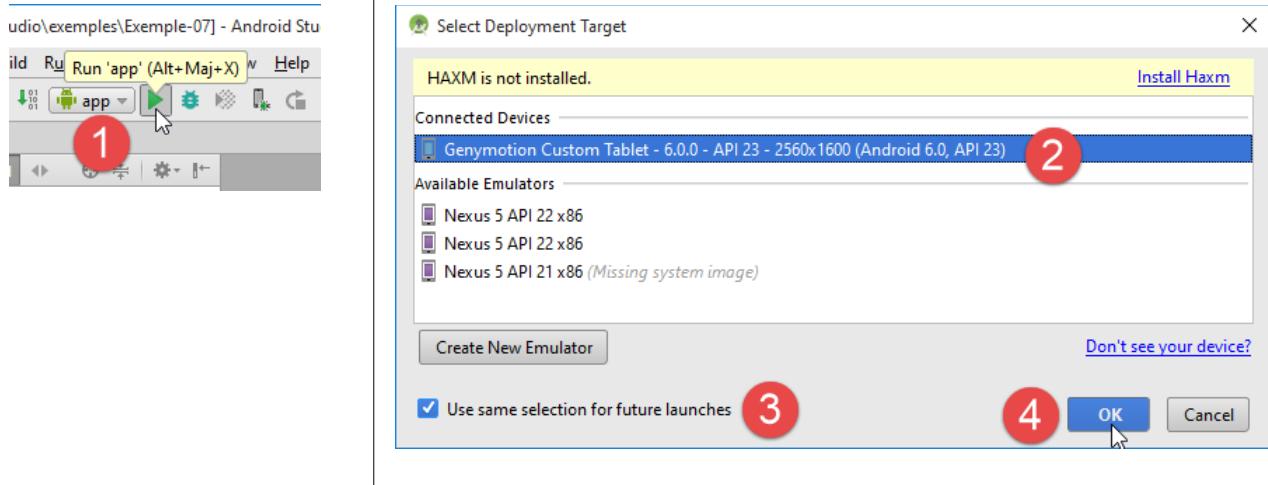


- en [8], on peut indiquer qu'on veut toujours utiliser le même terminal pour l'exécution. En général, cette option est cochée pour éviter d'avoir à préciser à chaque nouvelle exécution le terminal à utiliser. Ici, nous la laissons décochée parce que justement nous allons tester différents périphériques Android ;
- une fois la configuration d'exécution créée, on lance le gestionnaire des émulateurs Android [1-3] ;

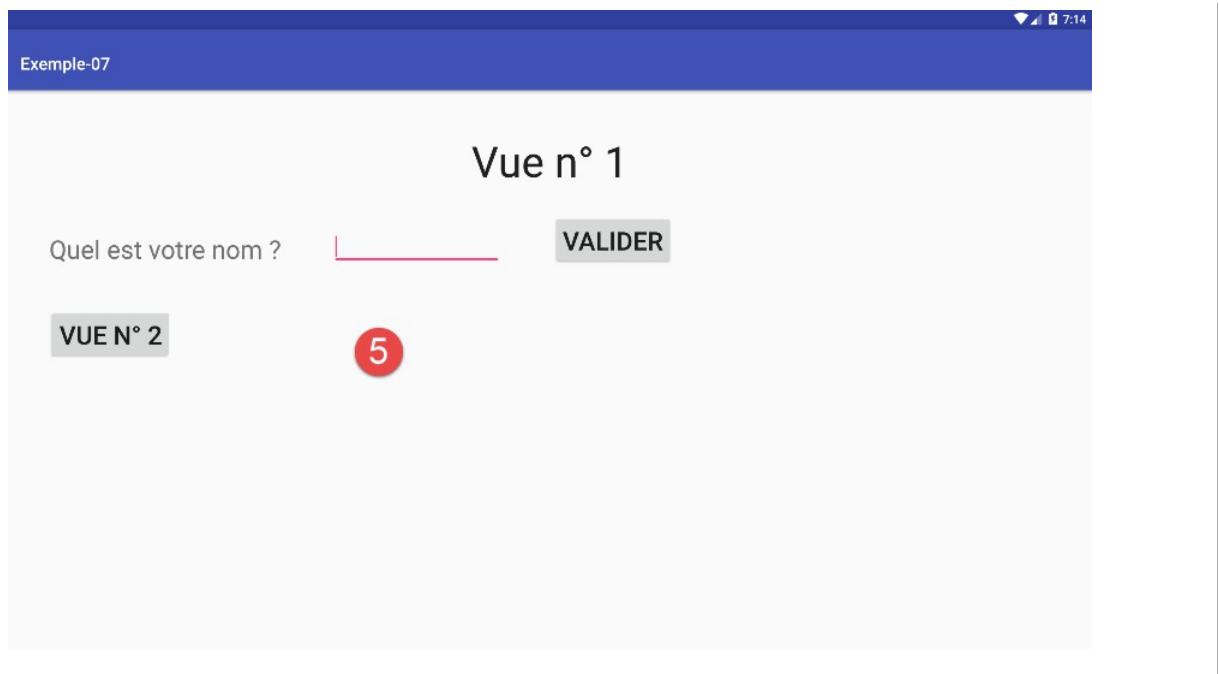


- si l'émulateur Android ne se lance pas, vérifiez les points mentionnés au paragraphe 6.11, page 484 ;

Pour lancer l'application sur l'émulateur, procédez comme suit [1-4] :

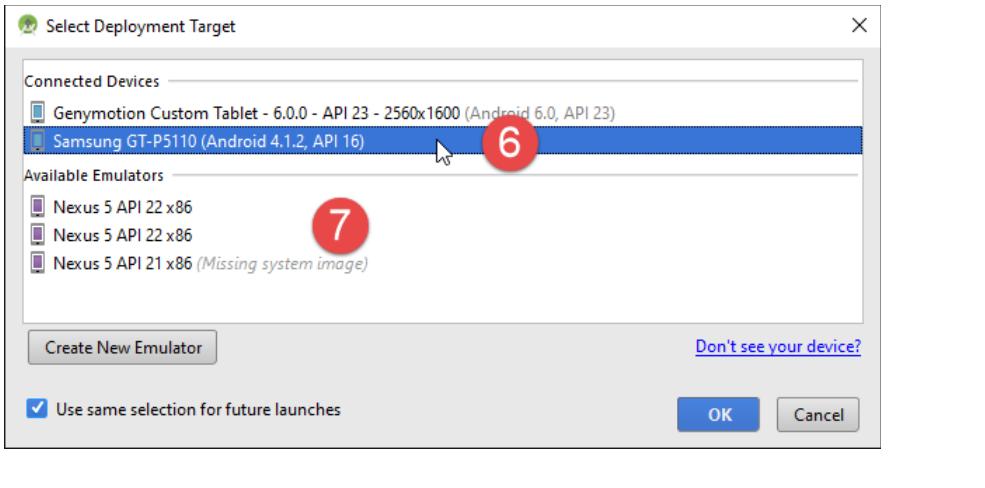


- en [2], vous devez voir apparaître l'émulateur que vous avez auparavant lancé ;



- en [5], la vue affichée par l'émulateur ;

Branchez maintenant une tablette Android sur un port USB du PC et exécutez l'application sur celle-ci :

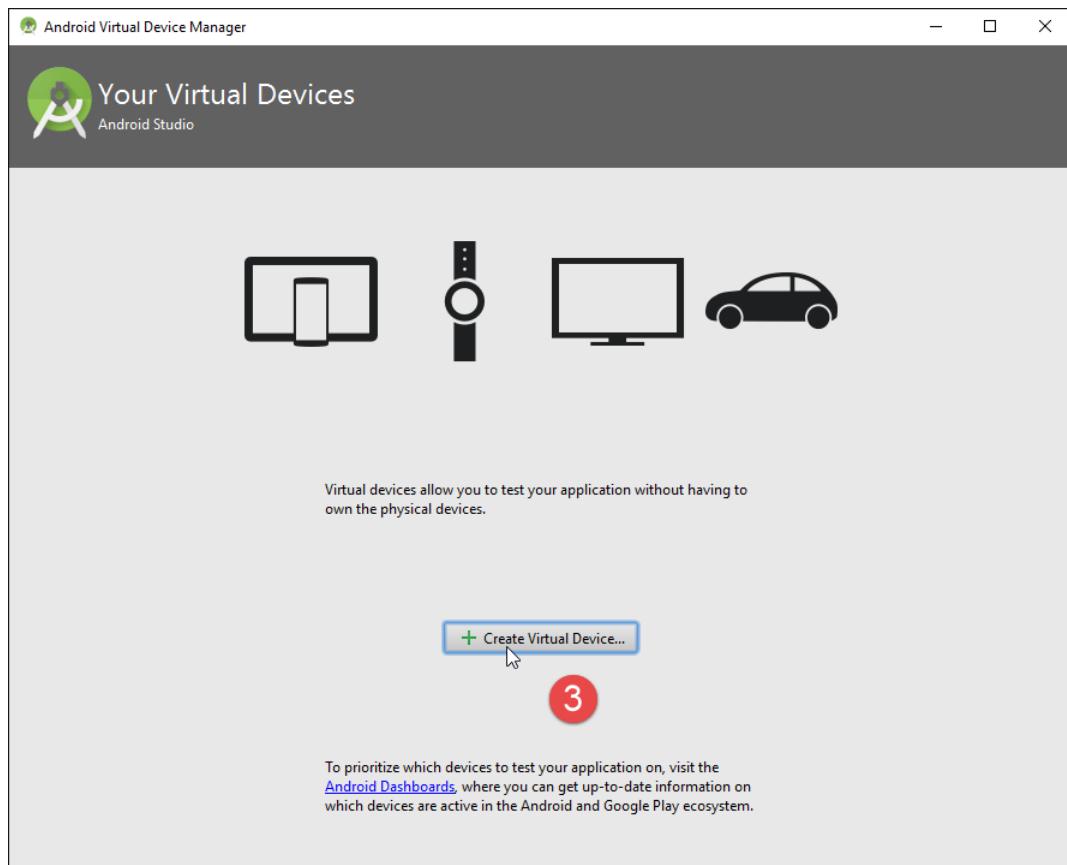


- en [6], sélectionnez la tablette Android et testez l'application.

En [7], nous avons des terminaux virtuels prédéfinis. Nous allons apprendre à en rajouter et à en éliminer.

Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
Tablet	Nexus 5 API 21 x86	1080 x 1920: xxhdpi	21	Android 5.0 (Googl...)	x86	750 MB	⚠ Download
Tablet	Nexus 5 API 22 x86	1080 x 1920: xxhdpi	22	Android 5.1 (Googl...)	x86	750 MB	▶ ✎ ▼
Tablet	Nexus 5 API 22 x86	1080 x 1920: xxhdpi	22	Android 5.1 (Googl...)	x86	750 MB	▶ ✎ ▼

Sur la copie d'écran ci-dessus, les terminaux proposés travaillent tous avec l'API 22. Nous les supprimons tous car nous voulons travailler avec l'API 23. Nous suivons la procédure [2-3] pour les terminaux à supprimer.



The screenshot shows the 'Select Hardware' screen. The left sidebar lists categories: TV, Wear, Phone, and Tablet. The 'Tablet' category is selected, indicated by a blue background and a red circle with the number '4' to its left. The main area displays a table of devices:

Category	Name	Size	Resolution	Density
TV	Nexus 9	8,86"	2048x1536	xhdpi
Wear	Nexus 7 (2012)	7,0"	800x1280	tvdpi
Phone	Nexus 7	7,02"	1200x1920	xhdpi
Tablet	Nexus 10	10,05"	2560x1600	xhdpi
	7" WSVGA (Tablet)	7,0"	600x1024	mdpi
	10.1" WXGA (Tablet)	10,1"	800x1280	mdpi

A red circle with the number '5' is overlaid on the 'Nexus 10' row in the table.

- en [4-5], on rajoute une tablette ;

Release Name	API Level	ABI	Target
Marshmallow	23	x86_64	Android 6.0 (with Google APIs)
Marshmallow Download	23	x86	Android 6.0 (with Google APIs)
Lollipop	22	x86	Android 5.1 (with Google APIs)
Lollipop Download	22	x86_64	Android 5.1 (with Google APIs)
KitKat Download	19	x86	Android 4.4 (with Google APIs)

- en [6], on choisit une API. Ci-dessus nous choisissons l'API 23 pour un Windows 64 bits ;

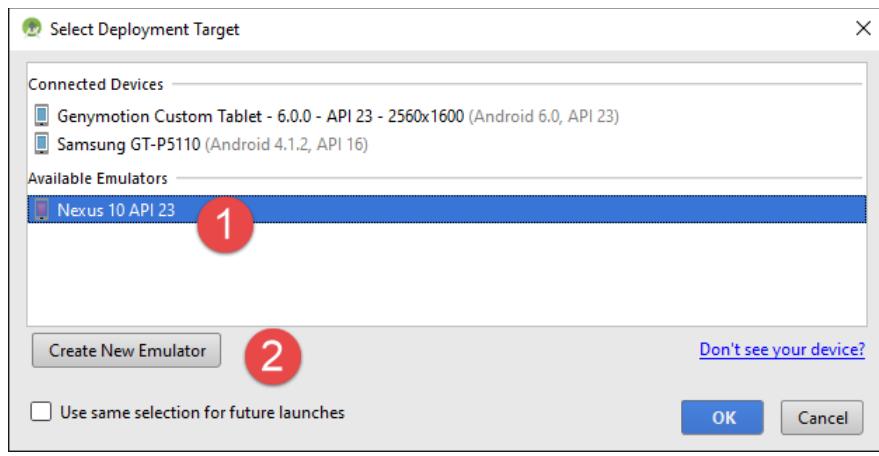


- en [7], le résumé de la configuration faite ;
- en [8], une configuration plus avancée du terminal virtuel peut être faite ;

Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
Nexus 10	Nexus 10 API 23	2560 x 1600: xhdpi	23	Android 6.0 (Googl...)	x86_64	650 MB	

Une fois l'assistant terminé, le terminal créé apparaît en [9].

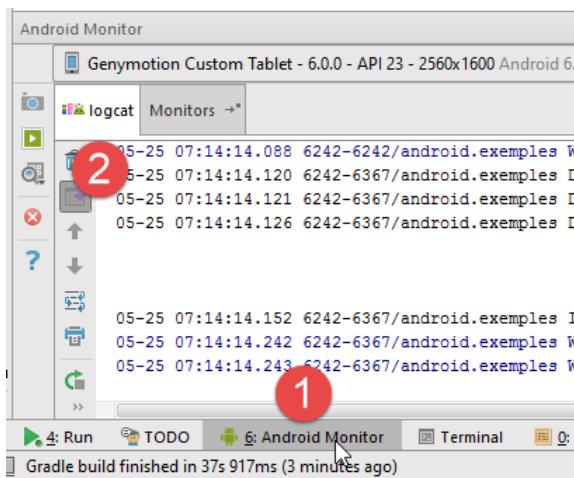
Ceci fait, si on relance l'exécution de [Exemple-07], on a maintenant la fenêtre suivante :



- en [1], le nouveau terminal virtuel apparaît ;
- en [2], on peut en créer de nouveaux ;

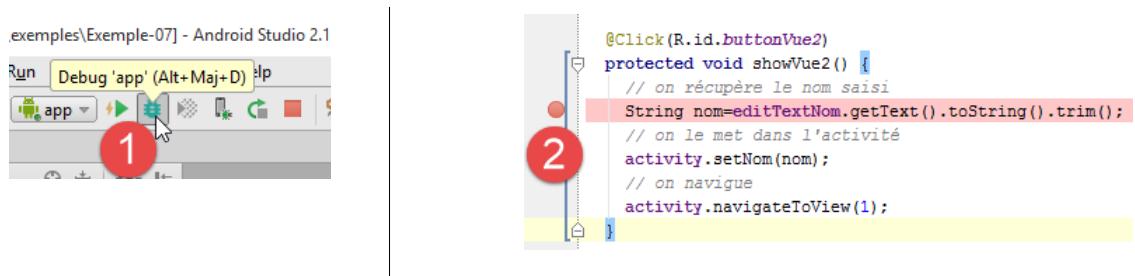
Faites des essais pour déterminer le terminal virtuel le plus adapté à votre poste. Dans ce document, les exemples ont été testés principalement avec l'émulateur Genymotion.

Quelque soit le terminal virtuel choisi, des logs sont affichés dans la fenêtre appelée [Logcat] [1-2] :

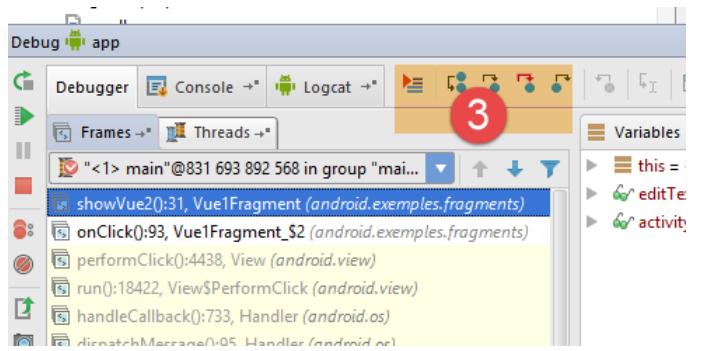


Consultez régulièrement ces logs. C'est là que seront signalées les exceptions qui ont fait planter votre programme.

Vous pouvez également déboguer votre programme avec les outils habituels du débogage :



- en [2], on met un point d'arrêt en cliquant une fois sur la colonne à gauche de la ligne cible. Un nouveau clic annule le point d'arrêt ;

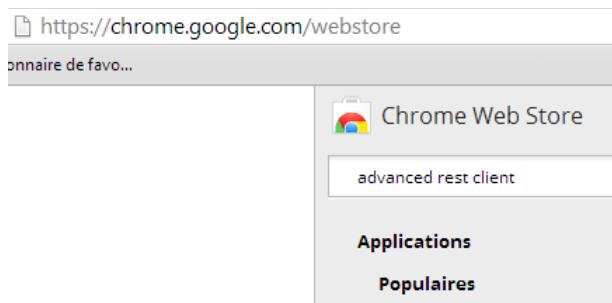


- en [3], au point d'arrêt faire :
 - [F6], pour exécuter la ligne sans entrer dans les méthodes si la ligne contient des appels de méthodes,
 - [F5], pour exécuter la ligne en entrant dans les méthodes si la ligne contient des appels de méthodes,
 - [F8], pour continuer jusqu'au prochain point d'arrêt ;
 - [Ctrl-F2] pour arrêter le débogage ;

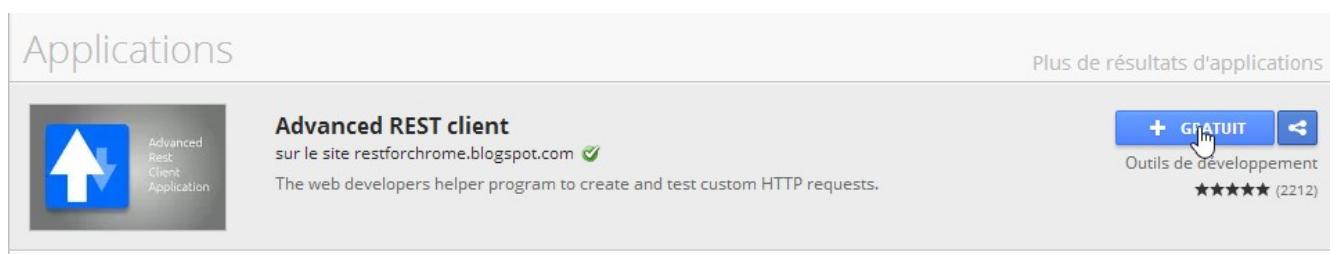
6.13 Installation du plugin Chrome [Advanced Rest Client]

Dans ce document, on utilise le navigateur **Chrome** de Google (<http://www.google.fr/intl/fr/chrome/browser/>). On lui ajoutera l'extension [Advanced Rest Client]. On pourra procéder ainsi :

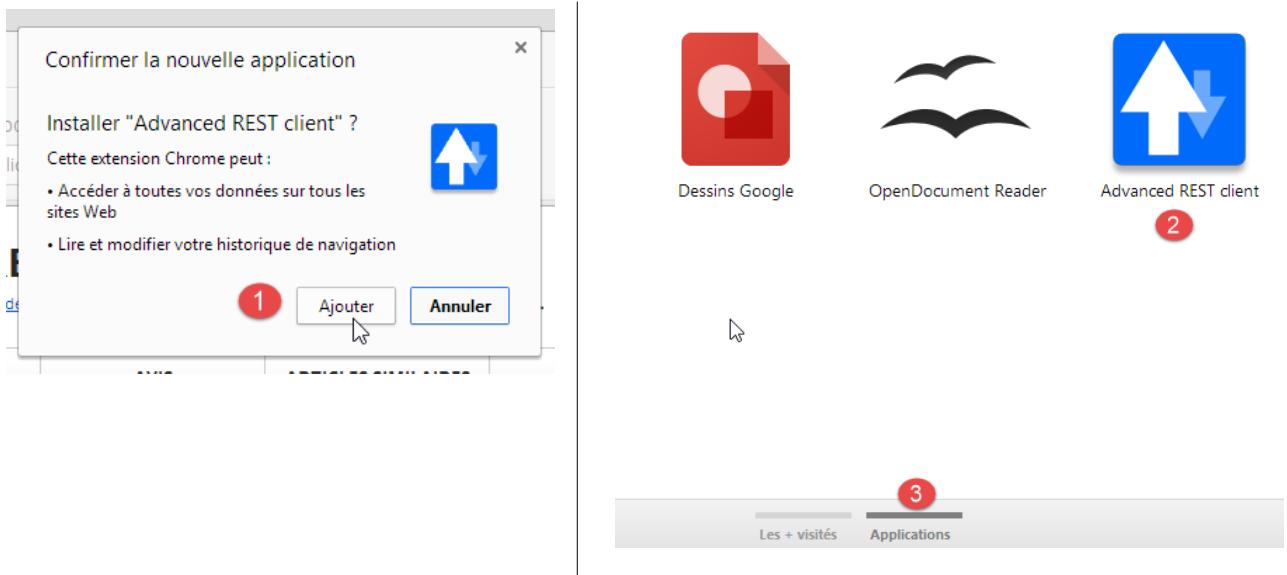
- aller sur le site de [Google Web store] (<https://chrome.google.com/webstore>) avec le navigateur Chrome ;
- chercher l'application [Advanced Rest Client] :



- l'application est alors disponible au téléchargement :



- pour l'obtenir, il vous faudra créer un compte Google. [Google Web Store] demande ensuite confirmation [1] :



- en [2], l'extension ajoutée est disponible dans l'option [Applications] [3]. Cette option est affichée sur chaque nouvel onglet que vous créez (CTRL-T) dans le navigateur.

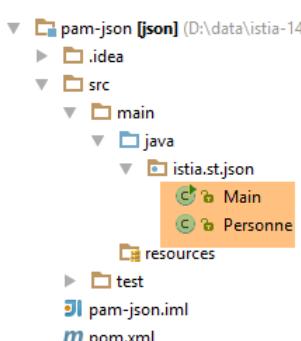
6.14 Gestion du JSON en Java

De façon transparente pour le développeur le framework [Spring MVC] utilise la bibliothèque JSON [Jackson]. Pour illustrer ce qu'est le JSON (JavaScript Object Notation), nous présentons ici un programme qui sérialise des objets en JSON et fait l'inverse en déserialisant les chaînes JSON produites pour recréer les objets initiaux.

La bibliothèque 'Jackson' permet de construire :

- la chaîne JSON d'un objet : `new ObjectMapper().writeValueAsString(object)` ;
- un objet à partir d'un chaîne JSON : `new ObjectMapper().readValue(jsonString, Object.class)`.

Les deux méthodes sont susceptibles de lancer une `IOException`. Voici un exemple.



Le projet ci-dessus est un projet Maven avec le fichier [pom.xml] suivant ;

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
6.
7. <groupId>istia.st.pam</groupId>
8. <artifactId>json</artifactId>
9. <version>1.0-SNAPSHOT</version>
10.
11. <dependencies>
12.   <dependency>
13.     <groupId>com.fasterxml.jackson.core</groupId>
```

```

14.      <artifactId>jackson-databind</artifactId>
15.      <version>2.3.3</version>
16.    </dependency>
17.  </dependencies>
18. </project>

```

- lignes 12-16 : la dépendance qui amène la bibliothèque 'Jackson' ;

La classe [Personne] est la suivante :

```

1. package istia.st.json;
2.
3. public class Personne {
4.     // data
5.     private String nom;
6.     private String prenom;
7.     private int age;
8.
9.     // constructeurs
10.    public Personne() {
11.
12.    }
13.
14.    public Personne(String nom, String prénom, int âge) {
15.        this.nom = nom;
16.        this.prenom = prénom;
17.        this.age = âge;
18.    }
19.
20.    // signature
21.    public String toString() {
22.        return String.format("Personne[%s, %s, %d]", nom, prenom, age);
23.    }
24.
25.    // getters et setters
26.    ...
27. }

```

La classe [Main] est la suivante :

```

1. package istia.st.json;
2.
3. import com.fasterxml.jackson.databind.ObjectMapper;
4.
5. import java.io.IOException;
6. import java.util.HashMap;
7. import java.util.Map;
8.
9. public class Main {
10.     // l'outil de sérialisation / désérialisation
11.     static ObjectMapper mapper = new ObjectMapper();
12.
13.     public static void main(String[] args) throws IOException {
14.         // création d'une personne
15.         Personne paul = new Personne("Denis", "Paul", 40);
16.         // affichage Json
17.         String json = mapper.writeValueAsString(paul);
18.         System.out.println("Json=" + json);
19.         // instantiation Personne à partir du Json
20.         Personne p = mapper.readValue(json, Personne.class);
21.         // affichage personne
22.         System.out.println("Personne=" + p);
23.         // un tableau
24.         Personne virginie = new Personne("Radot", "Virginie", 20);
25.         Personne[] personnes = new Personne[]{paul, virginie};
26.         // affichage Json
27.         json = mapper.writeValueAsString(personnes);
28.         System.out.println("Json personnes=" + json);
29.         // dictionnaire
30.         Map<String, Personne> hpersonnes = new HashMap<String, Personne>();
31.         hpersonnes.put("1", paul);
32.         hpersonnes.put("2", virginie);
33.         // affichage Json
34.         json = mapper.writeValueAsString(hpersonnes);
35.         System.out.println("Json hpersonnes=" + json);
36.     }
37. }

```

L'exécution de cette classe produit l'affichage écran suivant :

```

1. Json={"nom":"Denis","prenom":"Paul","age":40}
2. Personne=Personne[Denis, Paul, 40]

```

```

3. Json personnes=[{"nom":"Denis","prenom":"Paul","age":40},
   {"nom":"Radot","prenom":"Virginie","age":20}]
4. Json hpersonnes={"2":{"nom":"Radot","prenom":"Virginie","age":20},"1":
   {"nom":"Denis","prenom":"Paul","age":40}}

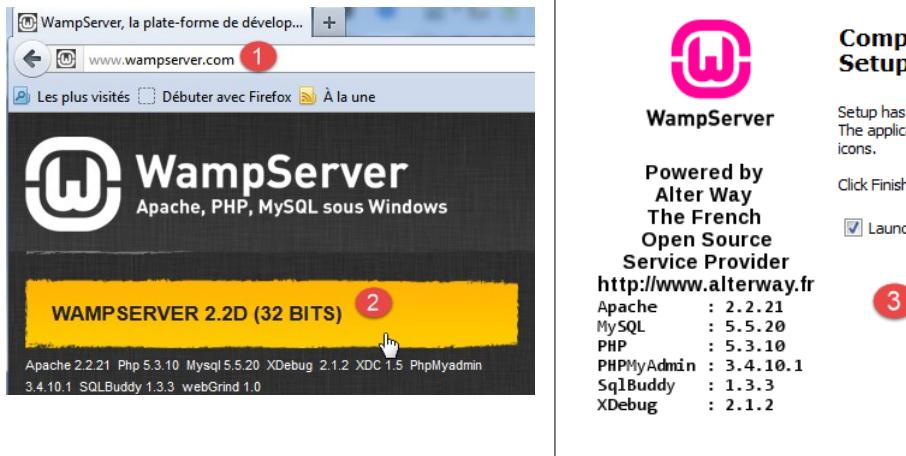
```

De l'exemple on retiendra :

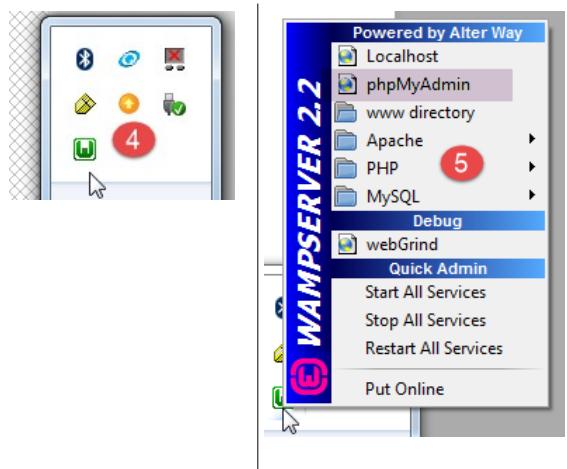
- l'objet [ObjectMapper] nécessaire aux transformations JSON / Object : ligne 11 ;
- la transformation [Personne] --> JSON : ligne 17 ;
- la transformation JSON --> [Personne] : ligne 20 ;
- l'exception [IOException] lancée par les deux méthodes : ligne 13.

6.15 Installation de [WampServer]

[WampServer] est un ensemble de logiciels pour développer en PHP / MySQL / Apache sur une machine Windows. Nous l'utiliserons uniquement pour le SGBD MySQL.



- sur le site de [WampServer] [1], choisir la version qui convient [2],
- l'exécutable téléchargé est un installateur. Diverses informations sont demandées au cours de l'installation. Elles ne concernent pas MySQL. On peut donc les ignorer. La fenêtre [3] s'affiche à la fin de l'installation. On lance [WampServer],



- en [4], l'icône de [WampServer] s'installe dans la barre des tâches en bas et à droite de l'écran [4],
- lorsqu'on clique dessus, le menu [5] s'affiche. Il permet de gérer le serveur Apache et le SGBD MySQL. Pour gérer celui-ci, on utiliser l'option [PhpPmyAdmin],
- on obtient alors la fenêtre ci-dessous,

The screenshot shows the PhpMyAdmin interface running in a Firefox browser. The title bar says "localhost / localhost | phpMyAdmin 3.4....". The left sidebar lists databases: information_schema, mysql, performance_schema, and test. The main menu at the top includes "Bases de données", "SQL", "État", "Log binaire", and "plus". Below the menu, there are two main sections: "Paramètres généraux" and "MySQL". The "Paramètres généraux" section shows the sorting method for MySQL connections as "utf8_general_ci". The "MySQL" section displays the following information:

- Serveur: localhost (localhost via TCP/IP)
- Version du serveur: 5.5.20-log
- Version du protocole: 10
- Utilisateur: root@localhost

Nous donnerons peu de détails sur l'utilisation de [PhpMyAdmin]. Nous montrons dans le document comment l'utiliser.

Table des matières

1 APPRENTISSAGE DE LA PROGRAMMATION ANDROID.....	9
 1.1 INTRODUCTION.....	9
1.1.1 CONTENU.....	9
1.1.2 PRÉ-REQUIS.....	10
1.1.3 LES OUTILS UTILISÉS.....	10
 1.2 EXEMPLE-01 : IMPORTATION D'UN EXEMPLE ANDROID.....	11
1.2.1 CRÉATION DU PROJET.....	11
1.2.2 QUELQUES POINTS SUR L'IDE.....	16
1.2.2.1 Les perspectives.....	16
1.2.2.2 Gestion de l'exécution.....	17
1.2.2.3 Gestion du cache.....	17
1.2.2.4 Gestion des logs.....	18
1.2.2.5 Gestion de l'émulateur [Genymotion].....	21
1.2.2.6 Gestion du binaire APK créé.....	21
 1.3 EXEMPLE-02 : UN PROJET ANDROID BASIQUE.....	23
1.3.1 CONFIGURATION GRADLE.....	25
1.3.2 LE MANIFESTE DE L'APPLICATION.....	26
1.3.3 L'ACTIVITÉ PRINCIPALE.....	29
1.3.4 EXÉCUTION DE L'APPLICATION.....	32
1.3.5 LE CYCLE DE VIE D'UNE ACTIVITÉ.....	34
 1.4 EXEMPLE-03 : RÉÉCRITURE DU PROJET [EXEMPLE-02] AVEC LA BIBLIOTHÈQUE [ANDROID ANNOTATIONS].....	36
 1.5 EXEMPLE-04 : VUES ET ÉVÉNEMENTS.....	42
1.5.1 CRÉATION DU PROJET.....	42
1.5.2 CONSTRUIRE UNE VUE.....	42
1.5.3 GESTION DES ÉVÉNEMENTS.....	49
 1.6 EXEMPLE-05 : NAVIGATION ENTRE VUES.....	51
1.6.1 CRÉATION DU PROJET.....	51
1.6.2 AJOUT D'UNE SECONDE ACTIVITÉ.....	51
1.6.3 NAVIGATION DE LA VUE N° 1 À LA VUE N° 2.....	54
1.6.4 CONSTRUCTION DE LA VUE N° 2.....	55
1.6.5 L'ACTIVITÉ [SECONDACTIVITY].....	57
1.6.6 NAVIGATION DE LA VUE N° 2 VERS LA VUE N° 1.....	59
1.6.7 CYCLE DE VIE DES ACTIVITÉS.....	60
 1.7 EXEMPLE-06 : NAVIGATION PAR ONGLETS.....	62
1.7.1 CRÉATION DU PROJET.....	62
1.7.2 CONFIGURATION GRADLE.....	65
1.7.3 LA VUE [ACTIVITY_MAIN].....	65
1.7.4 L'ACTIVITÉ.....	68
1.7.4.1 La gestion des fragments et des onglets.....	68
1.7.4.2 Les fragments affichés.....	70
1.7.4.3 Gestion du menu.....	71
1.7.4.4 Le bouton flottant.....	73
1.7.5 EXÉCUTION DU PROJET.....	74
1.7.6 CYCLE DE VIE DES FRAGMENTS.....	75
 1.8 EXEMPLE-07 : EXEMPLE-06 RÉÉCRIT AVEC LA BIBLIOTHÈQUE [AA].....	79
1.8.1 CRÉATION DU PROJET.....	79
1.8.2 CONFIGURATION GRADLE.....	79
1.8.3 AJOUT DES PREMIÈRES ANNOTATIONS AA.....	80
1.8.4 RÉÉCRITURE DES FRAGMENTS.....	81
1.8.5 EXAMEN DES LOGS.....	84
1.8.6 ONDESTROYVIEW.....	86
1.8.7 SETUSERVISIBLEHINT.....	88
1.8.8 SETOFFSCREENPAGELIMIT.....	89
1.8.9 ONDESTROY.....	92
 1.9 EXEMPLE-08 : MISE À JOUR D'UN FRAGMENT AVEC UNE ADJACENCE DE FRAGMENTS VARIABLE.....	94
1.9.1 CRÉATION DU PROJET.....	94
1.9.2 RÉÉCRITURE DU FRAGMENT [PLACEHOLDERFRAGMENT].....	94
1.9.3 COMMUNICATION INTER-FRAGMENTS.....	101
 1.10 EXEMPLE-09 : COMMUNICATION INTER-FRAGMENTS, SWIPE ET SCROLLING.....	104
1.10.1 CRÉATION DU PROJET.....	104
1.10.2 LA SESSION.....	104

1.10.3 L'ACTIVITÉ [MAINACTIVITY].....	105
1.10.4 LE FRAGMENT [PLACEHOLDERFRAGMENT].....	105
1.10.5 DÉSACTIVER LE SWIPE OU BALAYAGE.....	106
1.10.6 DÉSACTIVER LE SCROLLING ENTRE FRAGMENTS.....	108
1.10.7 UN NOUVEAU FRAGMENT.....	109
1.10.8 FAIRE DÉRIVER TOUS LES FRAGMENTS D'UNE MÊME CLASSE ABSTRAITE.....	111
1.11 EXEMPLE-10 : FAIRE DÉRIVER TOUS LES FRAGMENTS D'UNE CLASSE ABSTRAITE.....	112
1.11.1 CRÉATION DU PROJET.....	112
1.11.2 GESTION DU MODE DEBUG.....	112
1.11.3 LA CLASSE ABSTRAITE PARENTE DE TOUS LES FRAGMENTS.....	112
1.11.4 LA CLASSE [PLACEHOLDERFRAGMENT].....	115
1.11.5 LA CLASSE [VUE1FRAGMENT].....	116
1.11.6 ASSOCIATION ONGLETS / FRAGMENTS.....	117
1.12 EXEMPLE-11 : ONGLETS DISSOCIÉS DES FRAGMENTS.....	121
1.12.1 CRÉATION DU PROJET.....	121
1.12.2 OBJECTIFS.....	121
1.12.3 LA SESSION.....	122
1.12.4 LE MENU.....	122
1.12.5 LA CLASSE [MAINACTIVITY].....	123
1.12.6 AMÉLIORATIONS.....	125
1.13 EXEMPLE-12 : CODIFIER LES RELATIONS ENTRE ACTIVITÉ ET FRAGMENTS.....	126
1.13.1 CRÉATION DU PROJET.....	126
1.13.2 L'INTERFACE [IMAINACTIVITY].....	126
1.13.3 LA CLASSE ABSTRAITE [ABSTRACTFRAGMENT].....	127
1.13.4 MODIFICATION DU GESTIONNAIRE DE FRAGMENTS.....	127
1.13.5 MODIFICATION DE LA CLASSE [MAINACTIVITY].....	128
1.13.6 MODIFICATION DE L'AFFICHAGE DES FRAGMENTS DANS [MAINACTIVITY].....	128
1.13.7 CONCLUSION.....	128
1.14 EXEMPLE-13 : EXEMPLE-05 AVEC DES FRAGMENTS.....	129
1.14.1 CRÉATION DU PROJET.....	129
1.14.2 STRUCTURATION DU PROJET.....	129
1.14.3 NETTOYAGE DU PROJET.....	131
1.14.3.1 Nettoyage des fragments.....	131
1.14.3.2 Nettoyage de la session.....	132
1.14.3.3 Suppression des onglets, du bouton flottant et du menu.....	133
1.14.4 CRÉATION DES FRAGMENTS ET DES VUES ASSOCIÉES.....	137
1.14.5 MISE EN PLACE DES FRAGMENTS ET DE LA NAVIGATION ENTRE-EUX.....	138
1.14.6 DÉFINITION DE LA SESSION.....	140
1.14.7 ECRITURE FINALE DES FRAGMENTS.....	140
1.14.8 GESTION DU CYCLE DE VIE DES FRAGMENTS.....	142
1.14.9 CONCLUSION.....	143
1.15 EXEMPLE-14 : UNE ARCHITECTURE À DEUX COUCHES.....	144
1.15.1 CRÉATION DU PROJET.....	144
1.15.2 LA VUE [VUE1].....	144
1.15.3 LA SESSION.....	148
1.15.4 LE FRAGMENT [VUE1FRAGMENT].....	148
1.15.5 LA COUCHE [MÉTIER].....	150
1.15.6 L'ACTIVITÉ [MAINACTIVITY] REVISITÉE.....	152
1.15.7 LE FRAGMENT [VUE1FRAGMENT] REVISITÉ.....	153
1.15.8 EXÉCUTION.....	156
1.16 EXEMPLE-15 : ARCHITECTURE CLIENT / SERVEUR.....	157
1.16.1 LE SERVEUR [WEB / JSON].....	157
1.16.1.1 Création du projet.....	157
1.16.1.2 Configuration Gradle.....	159
1.16.1.3 Configuration du projet.....	165
1.16.1.4 La couche [métier].....	166
1.16.1.5 Le service web / JSON.....	167
1.16.1.6 Configuration du projet Spring.....	171
1.16.1.7 Exécution du service web / JSON.....	172
1.16.1.8 Génération du jar exécutables du projet.....	176
1.16.1.9 Gestion des logs.....	177
1.16.2 LE CLIENT ANDROID DU SERVEUR WEB / JSON.....	178
1.16.2.1 Création du projet.....	178

1.16.2.2	Configuration Gradle.....	179
1.16.2.3	Le manifeste de l'application Android.....	180
1.16.2.4	La couche [DAO].....	181
1.16.2.4.1	L'interface [IDao] de la couche [DAO].....	181
1.16.2.4.2	L'interface [WebClient].....	182
1.16.2.4.3	La classe [Response].....	183
1.16.2.4.4	Implémentation de la couche [DAO].....	183
1.16.2.5	Le package [architecture].....	186
1.16.2.5.1	L'interface [IMainActivity].....	187
1.16.2.5.2	La classe [Utils].....	187
1.16.2.5.3	La classe abstraite [AbstractFragment].....	188
1.16.2.6	La vue.....	188
1.16.2.6.1	La vue [vue1.xml].....	188
1.16.2.6.2	Le fragment [Vue1Fragment].....	191
1.16.2.7	L'activité [MainActivity].....	195
1.16.2.7.1	La vue [activity-main.xml].....	195
1.16.2.7.2	L'activité [MainActivity].....	196
1.16.2.8	Exécution du projet.....	196
1.16.2.9	Gestion de l'annulation.....	198
1.17	EXEMPLE-16 : GÉRER L'ASYNCRONISME AVEC RXANDROID.....	201
1.17.1	CRÉATION DU PROJET.....	201
1.17.2	CONFIGURATION GRADLE.....	201
1.17.3	LA COUCHE [DAO].....	202
1.17.4	L'INTERFACE [IDAO].....	202
1.17.5	LA CLASSE [ABSTRACTDAO].....	202
1.17.6	LA CLASSE [DAO].....	204
1.17.7	LA CLASSE [MAINACTIVITY].....	204
1.17.8	LA CLASSE [VUE1FRAGMENT].....	205
1.17.9	EXÉCUTION.....	207
1.17.10	GESTION DE L'ANNULATION.....	207
1.17.11	CONCLUSION.....	208
1.18	EXEMPLE-17 : COMPOSANTS DE SAISIE DE DONNÉES.....	210
1.18.1	CRÉATION DU PROJET.....	210
1.18.2	LA VUE XML DU FORMULAIRE.....	210
1.18.3	LES CHAÎNES DE CARACTÈRES DU FORMULAIRE.....	215
1.18.4	LE FRAGMENT DU FORMULAIRE.....	215
1.18.5	EXÉCUTION DU PROJET.....	218
1.19	EXEMPLE-18 : UTILISATION D'UN PATRON DE VUES.....	219
1.19.1	CRÉATION DU PROJET.....	219
1.19.2	LE PATRON DES VUES.....	219
1.20	EXEMPLE-19 : LE COMPOSANT [LISTVIEW].....	223
1.20.1	CRÉATION DU PROJET.....	223
1.20.2	LA SESSION.....	224
1.20.3	L'ACTIVITÉ [MAINACTIVITY].....	224
1.20.4	LA VUE [VUE1] INITIALE.....	225
1.20.5	LA VUE RÉPÉTÉE PAR LE [LISTVIEW].....	226
1.20.6	LE FRAGMENT [VUE1FRAGMENT].....	227
1.20.7	L'ADAPTATEUR [LISTADAPTER] DU [LISTVIEW].....	228
1.20.8	RETRIRER UN ÉLÉMENT DE LA LISTE.....	229
1.20.9	LA VUE XML [VUE2].....	230
1.20.10	LE FRAGMENT [VUE2FRAGMENT].....	231
1.20.11	EXÉCUTION.....	232
1.20.12	AMÉLIORATION.....	232
1.21	EXEMPLE-20 : UTILISER UN MENU.....	234
1.21.1	CRÉATION DU PROJET.....	234
1.21.2	LA DÉFINITION XML DES MENUS.....	234
1.21.3	LA GESTION DU MENU DANS LA CLASSE ABSTRAITE [ABSTRACTFRAGMENT].....	235
1.21.4	LA GESTION DU MENU DANS LE FRAGMENT [VUE1FRAGMENT].....	238
1.21.5	LA GESTION DU MENU DANS LE FRAGMENT [VUE2FRAGMENT].....	238
1.21.6	EXÉCUTION.....	239
1.22	EXEMPLE-21 : REFACTORING DE LA CLASSE ABSTRAITE [ABSTRACTFRAGMENT].....	240
1.22.1	CRÉATION DU PROJET.....	240
1.22.2	LE MENU DES FRAGMENTS.....	240

1.22.3	LES FRAGMENTS.....	241
1.22.4	EXÉCUTION.....	241
1.23	EXEMPLE-22 : SAUVEGARDE / RESTAURATION DE L'ÉTAT DE L'ACTIVITÉ ET DES FRAGMENTS.....	241
1.23.1	LE PROBLÈME.....	241
1.23.2	LES MÉTHODES DE SAUVEGARDE / RESTAURATION DE L'ACTIVITÉ ET DES FRAGMENTS.....	243
1.23.2.1	Solution 1 : sauvegarde manuelle.....	243
1.23.2.2	Solution 2 : sauvegarde automatique.....	244
1.23.3	LA MÉTHODE DE SAUVEGARDE / RESTAURATION DU PROJET [EXEMPLE-22].....	244
1.23.4	SAUVEGARDE DU FRAGMENT [VUE1FRAGMENT].....	247
1.23.5	SAUVEGARDE DU FRAGMENT [PLACEHOLDERFRAGMENT].....	247
1.23.6	RESTAURATION DU FRAGMENT [VUE1FRAGMENT].....	248
1.23.7	RESTAURATION DU FRAGMENT [PLACEHOLDERFRAGMENT].....	248
1.23.8	GESTION DES ONGLETS.....	249
2	SQUELETTE D'UN CLIENT ANDROID COMMUNICANT AVEC UN SERVICE WEB / JSON.....	251
2.1	ARCHITECTURE DU CLIENT ANDROID.....	251
2.2	LA CONFIGURATION GRADLE.....	252
2.3	LE MANIFESTE DE L'APPLICATION.....	253
2.4	L'ORGANISATION DU CODE JAVA.....	254
2.5	ÉLÉMENTS DE L'ACTIVITÉ.....	254
2.5.1	LA VUE ASSOCIÉE À L'ACTIVITÉ.....	255
2.5.2	LE CONTENEUR DE FRAGMENTS [MYPAGER].....	255
2.5.3	LA CLASSE [CORESTATE].....	256
2.5.4	L'INTERFACE [IMAINACTIVITY].....	257
2.5.5	L'INTERFACE [IDAO].....	258
2.5.6	LA SESSION.....	259
2.5.7	LA CLASSE ABSTRAITE [ABSTRACTACTIVITY].....	262
2.5.7.1	Squelette.....	262
2.5.7.2	Implémenter l'interface [IMainActivity].....	263
2.5.7.3	Sauvegarde de l'état de l'activité et de ses fragments.....	263
2.5.7.4	Restauration de l'état de l'activité et de ses fragments.....	264
2.5.7.5	Initialisation de la couche [DAO].....	264
2.5.7.6	Initialisation de la vue associée à l'activité.....	265
2.5.7.7	Gestion des onglets.....	266
2.5.7.8	Le gestionnaire d'onglets [CustomTabLayout].....	267
2.5.7.9	Dernières initialisations.....	268
2.5.7.10	Gestion de l'image d'attente.....	269
2.5.7.11	Implémentation de l'interface [IDao].....	269
2.5.7.12	Implémentation du gestionnaire de fragments.....	270
2.5.7.13	La méthode [onResume].....	270
2.5.7.14	Résumé.....	271
2.5.8	L'ACTIVITÉ [MAINACTIVITY].....	271
2.6	LA COUCHE [DAO].....	272
2.6.1	L'INTERFACE IDAO.....	273
2.6.2	L'INTERFACE [WEBCLIENT].....	273
2.6.3	L'INTERCEPTEUR D'AUTHENTIFICATION[MYAUTHINTERCEPTOR].....	274
2.6.4	LA CLASSE [ABSTRACTDAO].....	275
2.6.5	LA CLASSE [DAO].....	276
2.7	LES FRAGMENTS.....	278
2.7.1	LA CLASSE [MENUITEMSTATE].....	278
2.7.2	LA CLASSE [UTILS].....	278
2.7.3	LA CLASSE PARENT [ABSTRACTFRAGMENT].....	279
2.7.3.1	Le squelette.....	279
2.7.3.2	Le constructeur.....	280
2.7.3.3	Gestion du menu.....	281
2.7.3.4	Gestion de l'attente de la fin d'une tâche asynchrone.....	282
2.7.3.5	Gestion des exceptions.....	283
2.7.3.6	Gestion des opérations asynchrones.....	283
2.7.3.7	Gestion du cycle de vie du fragment.....	285
2.7.3.8	Mise à jour du fragment.....	287
2.7.4	UN EXEMPLE DE FRAGMENT.....	292
2.8	EXERCICES D'ILLUSTRATION.....	294
2.8.1	EXEMPLE-17B.....	294
2.8.1.1	Le projet [Exemple-17B].....	295

<u>2.8.1.2</u> L'état du fragment [Vue1Fragment].....	297
<u>2.8.1.3</u> Personnalisation du projet.....	297
<u>2.8.1.4</u> L'activité [MainActivity].....	299
<u>2.8.1.5</u> L'état du fragment [FragmentState].....	300
<u>2.8.1.6</u> Le fragment [AbstractFragment].....	301
<u>2.8.1.7</u> Tests.....	304
<u>2.8.2</u> EXEMPLE-23 : CLIENT MÉTÉO.....	304
<u>2.8.2.1</u> Le projet.....	304
<u>2.8.2.2</u> Personnalisation du projet.....	304
<u>2.8.2.3</u> La couche [DAO].....	306
<u>2.8.2.4</u> L'activité [MainActivity].....	309
<u>2.8.2.5</u> Le fragment[MeteoFragment].....	310
<u>2.8.2.6</u> Tests.....	314
<u>2.8.3</u> EXEMPLE-16B.....	316
<u>2.8.3.1</u> Le projet Exemple-16B.....	317
<u>2.8.3.2</u> Création d'un état pour le fragment [Vue1Fragment].....	319
<u>2.8.3.3</u> Personnalisation du projet.....	319
<u>2.8.3.4</u> La couche [DAO].....	321
<u>2.8.3.5</u> L'activité [MainActivity].....	324
<u>2.8.3.6</u> L'état du fragment[Vue1Fragment].....	325
<u>2.8.3.7</u> Le fragment[Vue1Fragment].....	326
<u>2.8.3.7.1</u> Gestion du clic sur le bouton [Exécuter].....	326
<u>2.8.3.7.2</u> Le cycle de vie du fragment.....	328
<u>2.8.3.8</u> Les tests.....	329
<u>2.8.4</u> EXEMPLE-22B.....	329
<u>2.8.4.1</u> Personnalisation du projet.....	331
<u>2.8.4.2</u> L'activité [MainActivity].....	333
<u>2.8.4.2.1</u> Implémentation des méthodes de la classe parent.....	333
<u>2.8.4.2.2</u> Gestion des onglets.....	334
<u>2.8.4.2.3</u> Gestion du menu.....	335
<u>2.8.4.3</u> Le fragment [Vue1Fragment].....	337
<u>2.8.4.4</u> L'état [PlaceHolderFragmentState].....	338
<u>2.8.4.5</u> Le fragment [PlaceHolderFragment].....	339
<u>2.8.4.6</u> Tests.....	340
<u>2.9</u> CONCLUSION.....	340
3 ÉTUDE DE CAS - GESTION DE RENDEZ-VOUS.....	341
<u>3.1</u> LE PROJET.....	341
<u>3.2</u> LES VUES DU CLIENT ANDROID.....	341
<u>3.3</u> L'ARCHITECTURE DU PROJET.....	343
<u>3.4</u> LA BASE DE DONNÉES.....	344
<u>3.4.1</u> LA TABLE [MEDECINS].....	344
<u>3.4.2</u> LA TABLE [CLIENTS].....	345
<u>3.4.3</u> LA TABLE [CRENEAUX].....	345
<u>3.4.4</u> LA TABLE [RV].....	346
<u>3.4.5</u> GÉNÉRATION DE LA BASE.....	346
<u>3.5</u> LE SERVEUR WEB / JSON.....	348
<u>3.5.1</u> MISE EN OEUVRE.....	349
<u>3.5.2</u> SÉCURISATION DU SERVICE WEB.....	350
<u>3.5.3</u> LISTE DES MÉDECINS.....	350
<u>3.5.4</u> LISTE DES CLIENTS.....	353
<u>3.5.5</u> LISTE DES CRÉNEAUX D'UN MÉDECIN.....	354
<u>3.5.6</u> LISTE DES RENDEZ-VOUS D'UN MÉDECIN.....	354
<u>3.5.7</u> L'AGENDA D'UN MÉDECIN.....	355
<u>3.5.8</u> OBTENIR UN MÉDECIN PAR SON IDENTIFIANT.....	355
<u>3.5.9</u> OBTENIR UN CLIENT PAR SON IDENTIFIANT.....	356
<u>3.5.10</u> OBTENIR UN CRÉNEAU PAR SON IDENTIFIANT.....	356
<u>3.5.11</u> OBTENIR UN RENDEZ-VOUS PAR SON IDENTIFIANT.....	356
<u>3.5.12</u> AJOUTER UN RENDEZ-VOUS.....	356
<u>3.5.13</u> SUPPRIMER UN RENDEZ-VOUS.....	358
<u>3.6</u> LE CLIENT ANDROID.....	359
<u>3.6.1</u> ARCHITECTURE DU PROJET ANDROID STUDIO.....	359
<u>3.6.2</u> PERSONNALISATION DU PROJET.....	360
<u>3.6.3</u> LA COUCHE [DAO].....	362

<u>3.6.3.1</u> Implémentation des échanges client / serveur.....	363
<u>3.6.3.2</u> L'interface [IDao].....	364
<u>3.6.3.3</u> La classe [Dao].....	365
<u>3.6.4</u> L'ACTIVITÉ [MAINACTIVITY].....	369
<u>3.6.5</u> LA SESSION.....	371
<u>3.6.6</u> GESTION DE LA VUE DE CONFIGURATION.....	372
<u>3.6.6.1</u> La vue.....	372
<u>3.6.6.2</u> Le fragment.....	372
<u>3.6.6.3</u> Gestion du cycle de vie du fragment.....	376
<u>3.6.7</u> GESTION DE LA VUE D'ACCUEIL.....	377
<u>3.6.7.1</u> La vue.....	377
<u>3.6.7.2</u> Le fragment.....	378
<u>3.6.7.3</u> Gestion du cycle de vie du fragment.....	381
<u>3.6.8</u> GESTION DE LA VUE AGENDA.....	383
<u>3.6.8.1</u> La vue.....	383
<u>3.6.8.2</u> Le fragment.....	384
<u>3.6.8.2.1</u> Méthode [updateAgenda].....	385
<u>3.6.8.2.2</u> Méthode [doSupprimer].....	389
<u>3.6.8.2.3</u> Méthode [doAnnuler].....	391
<u>3.6.8.2.4</u> Option de menu [Retour à la configuration].....	391
<u>3.6.8.2.5</u> Option de menu [Retour à l'accueil].....	391
<u>3.6.8.3</u> Gestion du cycle de vie du fragment.....	391
<u>3.6.9</u> GESTION DE LA VUE D'AJOUT D'UN RENDEZ-VOUS.....	393
<u>3.6.9.1</u> La vue.....	393
<u>3.6.9.2</u> Le fragment.....	394
<u>3.6.9.3</u> Gestion du cycle de vie du fragment.....	397
<u>3.7</u> EXÉCUTION.....	399
<u>4</u> TP 1 : GESTION BASIQUE D'UNE FICHE DE PAIE.....	401
<u>4.1</u> INTRODUCTION.....	401
<u>4.2</u> LA BASE DE DONNÉES.....	401
<u>4.2.1</u> DÉFINITION.....	401
<u>4.2.2</u> GÉNÉRATION.....	402
<u>4.2.3</u> MODÉLISATION JAVA DE LA BASE.....	403
<u>4.3</u> INSTALLATION DU SERVEUR WEB / JSON.....	404
<u>4.3.1</u> INSTALLATION.....	405
<u>4.3.2</u> LES URL DU SERVICE WEB/JSON.....	406
<u>4.3.3</u> LES RÉPONSES JSON DU SERVICE WEB/JSON.....	407
<u>4.4</u> TESTS DU CLIENT ANDROID.....	409
<u>4.5</u> TRAVAIL À FAIRE.....	412
<u>5</u> TP 2 - PILOTER DES ARDUINOS AVEC UNE TABLETTE ANDROID.....	414
<u>5.1</u> ARCHITECTURE DU PROJET.....	414
<u>5.2</u> LE MATÉRIEL.....	414
<u>5.2.1</u> L'ARDUINO.....	414
<u>5.2.2</u> LA TABLETTE.....	416
<u>5.2.3</u> L'ÉMULATEUR [GENYMOTION].....	417
<u>5.3</u> PROGRAMMATION DES ARDUINOS.....	417
<u>5.4</u> LE SERVEUR WEB / JSON.....	421
<u>5.4.1</u> INSTALLATION.....	421
<u>5.4.2</u> LES URL EXPOSÉES PAR LE SERVICE WEB / JSON.....	423
<u>5.4.3</u> LES TESTS DU SERVICE WEB / JSON.....	425
<u>5.5</u> TESTS DU CLIENT ANDROID.....	428
<u>5.6</u> LE CLIENT ANDROID DU SERVICE WEB / JSON.....	432
<u>5.6.1</u> L'ARCHITECTURE DU CLIENT.....	432
<u>5.6.2</u> LE PROJET ANDROID STUDIO DU CLIENT.....	432
<u>5.6.3</u> LES CINQ VUES XML.....	432
<u>5.6.4</u> LE MENU DES FRAGMENTS.....	433
<u>5.6.5</u> LES CINQ FRAGMENTS DE L'APPLICATION.....	434
<u>5.6.6</u> LES ÉTATS DES FRAGMENTS.....	435
<u>5.6.7</u> PERSONNALISATION DU PROJET.....	435
<u>5.6.7.1</u> L'interface [IMainActivity].....	435
<u>5.6.7.2</u> La classe [CoreState].....	436
<u>5.6.8</u> LA CLASSE [MAINACTIVITY].....	437
<u>5.6.9</u> LA VUE XML [CONFIG].....	439

5.6.10 LE FRAGMENT [CONFIGFRAGMENT].....	442
 5.6.10.1 Le bouton [Rafraîchir].....	444
 5.6.10.2 Vérification des saisies.....	445
 5.6.10.3 Affichage de la liste des Arduinos.....	446
 5.6.10.4 Un patron pour afficher un Arduino.....	449
 5.6.10.5 La session.....	453
 5.6.10.6 Gestion de l'état du fragment.....	454
 5.6.10.7 Gestion du cycle de vie du fragment.....	456
 5.6.10.8 Amélioration du code.....	458
 5.6.11 COMMUNICATION ENTRE VUES.....	459
 5.6.12 LA COUCHE [DAO].....	462
 5.6.12.1 L'interface IDao.....	462
 5.6.12.2 L'interface [WebClient].....	463
 5.6.12.3 La classe [Dao].....	464
 5.6.13 L'ACTIVITÉ [MAINACTIVITY].....	466
 5.6.14 LE FRAGMENT [CONFIGFRAGMENT] REVISITÉ.....	466
5.7 TRAVAIL À FAIRE.....	468
6 ANNEXES.....	470
 6.1 INTALLATION DE L'IDE ARDUINO.....	470
 6.2 INTALLATION DU PILOTE (DRIVER) DE L'ARDUINO.....	470
 6.3 TESTS DE L'IDE.....	470
 6.4 CONNEXION RÉSEAU DE L'ARDUINO.....	472
 6.5 TEST D'UNE APPLICATION RÉSEAU.....	473
 6.6 LA BIBLIOTHÈQUE AJSON.....	476
 6.7 LA TABLETTE ANDROID.....	477
 6.8 INSTALLATION D'UN JDK.....	479
 6.9 INSTALLATION DU GESTIONNAIRE D'ÉMULATEURS GENYMOTION.....	480
 6.10 INSTALLATION DE MAVEN.....	483
 6.11 INSTALLATION DE L'IDE ANDROID STUDIO.....	484
 6.12 UTILISATION DES EXEMPLES.....	490
 6.13 INSTALLATION DU PLUGIN CHROME [ADVANCED REST CLIENT].....	501
 6.14 GESTION DU JSON EN JAVA.....	502
 6.15 INSTALLATION DE [WAMPSERVER].....	504