GUIDE
ÉTAPE 2 : GESTION PROFESSIONNELLE DU PROJET

Contenu

1. Structure de l'application en MVC, tests unitaires et documentation technique

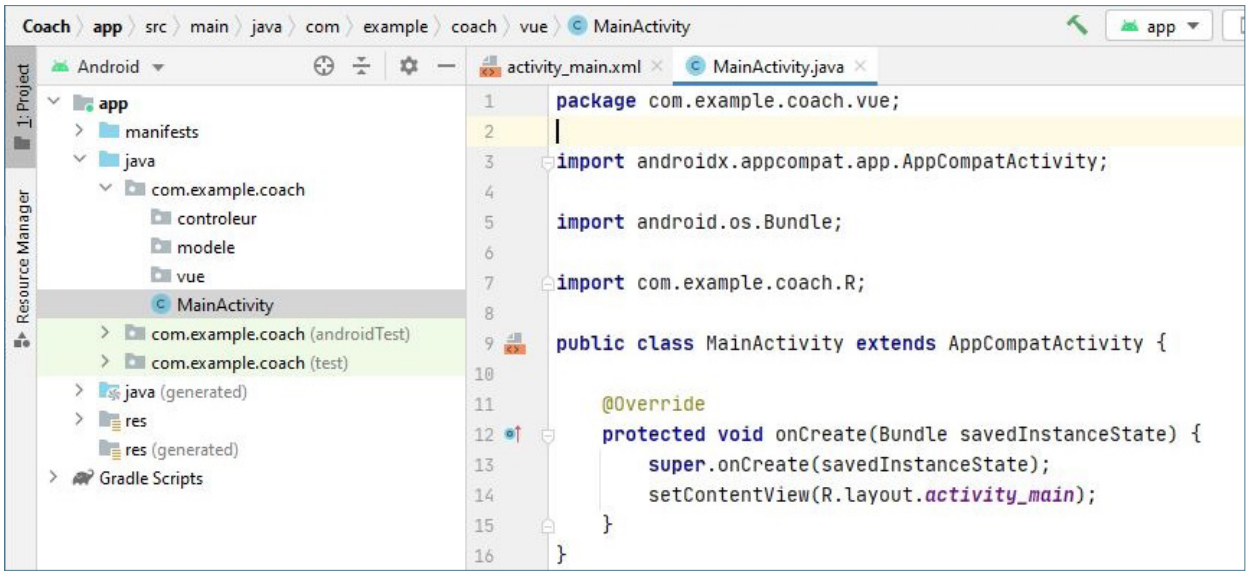
Maintenant que l'interface est créée et le code de base généré, il est temps de commencer à coder. Le but est de coder proprement dès le début, donc en respectant le modèle MVC.

Dans le package principal de l'application (com.example.coach), il faut donc créer les packages nécessaires pour gérer le MVC :

Faites un clic droit sur le package principal (com.example.coach) et choisissez "New > Package". Entrez le nom du package : "modele" (sans majuscule) et validez. Remarquez que le package s'est bien créé dans le package principal. En utilisant la même méthode, créez les 2 autres packages ("vue" et "controleur").

La classe MainActivity est une classe de visualisation. Donc elle doit aller dans la package "vue" : faites-la glisser dans le package. Vous remarquerez une fenêtre qui s'ouvre, vous proposant des options (ne modifiez rien) et vous offrant le bouton "refactor". Cliquez sur le bouton. Comprenez bien le principe du "refactoring" : déplacer une classe, une méthode, modifier le nom de quoi que ce soit dans un programme, peut avoir des incidences sur tout le reste du programme. La puissance d'un IDE est, entre autres, de gérer dans ces cas-là le refactoring, c'est-à-dire la recherche de tous les points du programme qui pourraient être concernés par ce changement, et proposer des options pour que ça se passe le mieux possible.

Au final, vous devriez obtenir cette arborescence :



Remarquez, dans la partie centrale, que pour MainActivity.java, le package a été automatiquement modifié : "package com.example.coach.vue". Maintenant que la structure est construite, il va être possible de créer et remplir les différentes classes en essayant d'être le plus "propre" possible. Le but est donc de remplir le modèle (qui gère les informations), le contrôleur (qui décide) et la vue (qui affiche et demande).

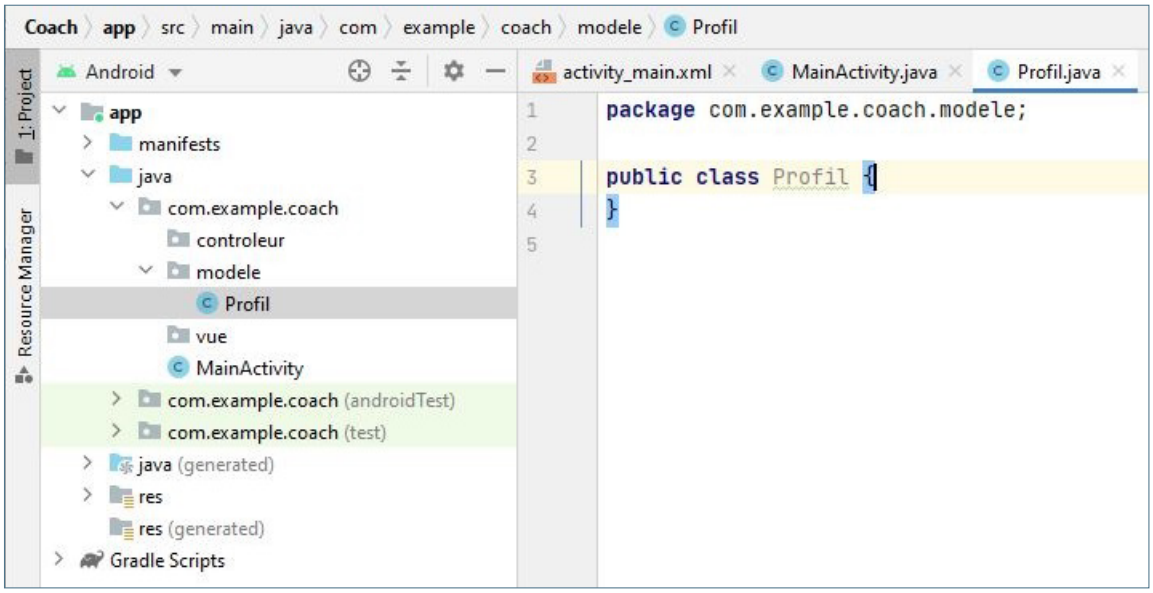
1A. Le modèle qui gère les informations

Le but de l'application est, dans un premier temps, de calculer l'IMG. Il parait naturel de créer une classe qui va permettre de mémoriser les informations nécessaires (celles qui vont être saisies, comme le poids, la taille...), de calculer l'IMG et de retourner les informations à afficher (trop maigre, trop de graisse...).

Tout ceci concerne le profil d'une personne, donc on va tout regrouper dans une classe métier qui va être placée dans le modèle :

Faites un clic droit sur la package "modele", choisissez "New > Java Class" et donnez le nom Profil et validez.

La classe est automatiquement créée dans le package modele.



1A1. Les propriétés

Cette classe doit mémoriser les informations du profil. Déclarez les 4 propriétés suivantes en Integer (et bien sûr en privé) : poids, taille, age et sexe. Le calcul de l'img sera aussi mémorisé : déclarez la propriété img en float. Enfin, il faudra mémoriser le message à retourner ("trop maigre"...) : déclarez la propriété message de type String.

1A2. Le constructeur et les getters/setters

Classiquement, le constructeur valorise les propriétés privées et on peut avoir besoin de setters et getters pour accéder à ces propriétés. L'IDE a prévu la génération automatique d'un tel constructeur et des getters/setters.

Pour cela, placez-vous sous la déclaration des propriétés, faites un clic droit "generate > constructor". Le constructeur n'a pas forcément besoin de tout valoriser, d'où la possibilité de sélectionner les propriétés. C'est notre cas : img et message vont être calculés, donc ne sélectionnez que les 4 autres propriétés (avec Shift ou Ctrl) et faites ok. Le constructeur est créé.

Avant de continuer, remarquez que le nom du constructeur (Profil) est en grisé. C'est d'ailleurs aussi le cas des propriétés déclarées juste au-dessus. Dans l'IDE, quand un nom est en grisé, c'est que la propriété ou la méthode n'est pas encore utilisée. C'est une indication qui peut être parfois bien pratique.

En utilisant le même principe que pour générer le constructeur, générez les 6 getters pour pouvoir récupérer les informations de toutes les propriétés. En revanche, il ne sera pas nécessaire de générer des setters (la valorisation ne va se faire que par le constructeur ou par calcul).

1A3. Les constantes

Il a été dit au début du TP que la détermination du niveau de l'img (normal, trop maigre, trop gras) se fait en fonction d'un seuil fixe, différent si c'est un homme ou une femme. Autant mémoriser ces 4 seuils (min et max pour homme et femme) dans des constantes de classes que vous allez déclarer en début de classe (donc déclarées en static, private et final) :

```
// constantes
private static final Integer minFemme = 15; // maigre si en dessous
private static final Integer maxFemme = 30; // gros si au dessus
private static final Integer minHomme = 10; // maigre si en dessous
private static final Integer maxHomme = 25; // gros si au dessus
```

Rappel

`static` signifie que la propriété est à portée de classe (sa valeur reste la même pour tous les objets générés à partir de cette classe). `final` signifie que la propriété ne peut être modifiée.

1A4. Les méthodes

Deux méthodes sont à écrire : une qui calcule l'img et une qui génère le résultat, donc le message.

Écrivez la méthode privée calculIMG qui ne reçoit aucun paramètre et qui ne retourne rien (void). Elle va juste valoriser la propriété img en lui affectant le calcul. Pour faire le calcul, utilisez le contenu des propriétés privées et aidez-vous de la formule donnée dans le guide 1 et rappelée ici :

IMG = (1,2 * Poids/Taille²) + (0,23 * age) - (10,83 * S) - 5,4

- avec S=0 pour une femme, =1 pour un homme ;
- taille en mètres mais que l'on saisira en cm pour éviter la saisie de la virgule.

Attention, vous aurez certainement besoin de transtyper (parser) le résultat du calcul avant de l'affecter à la propriété img, car vous utilisez des propriétés Integer et img est de type float. Attention aussi à la taille qui est en cm et qu'il faut convertir en m pour le calcul (là encore, transtypez d'abord la taille en float avant de diviser par 100). D'ailleurs, il est conseillé de faire le calcul de la taille en cm dans un premier temps, de le stocker dans une variable locale de type float, puis de s'en servir pour le calcul de l'img. Pour transtyper, il suffit de mettre "(float)" devant le calcul ou la valeur à transtyper (après avoir mis des parenthèses tout autour de l'ensemble du calcul à transtyper).

Écrivez la méthode privée resultIMG qui ne reçoit aucun paramètre et qui ne retourne rien. Elle va juste valoriser la propriété message. Pour cela, faites les tests nécessaires, en utilisant le contenu d'img et les constantes. Le but est d'obtenir les messages suivants : "normal", trop faible", "trop élevé". Les critères ont été donnés dans le guide 1. Les voici pour rappel :

- Femmes :
  - < 15 % : trop maigre ;
  - 15-30 % : normal ;
  - > 30 % : trop de graisse.
- Hommes :
  - < 10 % : trop maigre ;
  - 10-25 % : normal ;
  - > 25 % : trop de graisse.

Pourquoi ces 2 méthodes sont privées ? Parce qu'on ne va pas les appeler de l'extérieur. Inutile de refaire plusieurs fois le calcul. Elles servent à valoriser les propriétés correspondantes qui seront récupérées par des getters. Du coup, pour que la valorisation se fasse dès le début, appelez ces 2 méthodes en fin de constructeur (en appelant bien sûr calculIMG avant resultIMG).

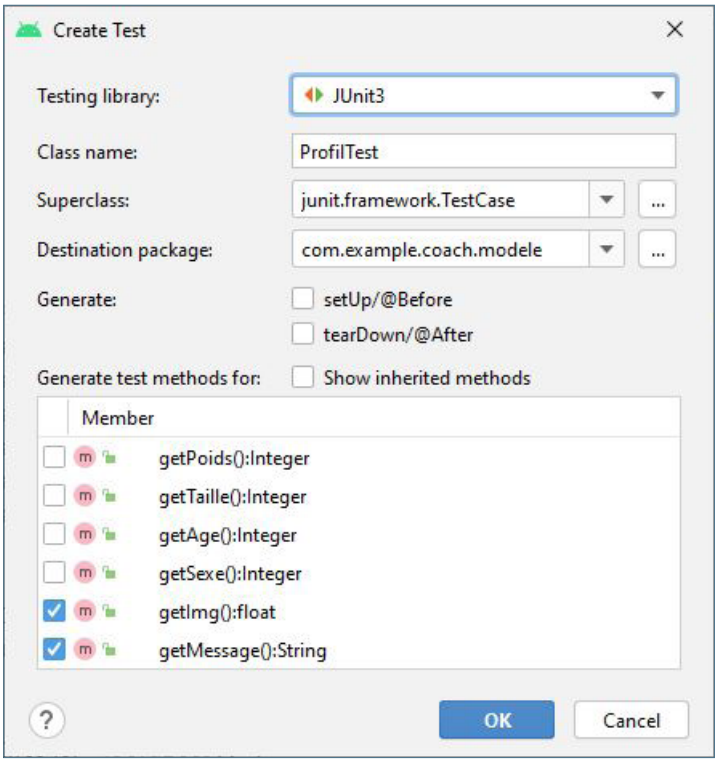
C'est aussi l'occasion de constater que des méthodes peuvent aussi être privées. On ne met en public que le strict minimum.

1A5. Tests unitaires

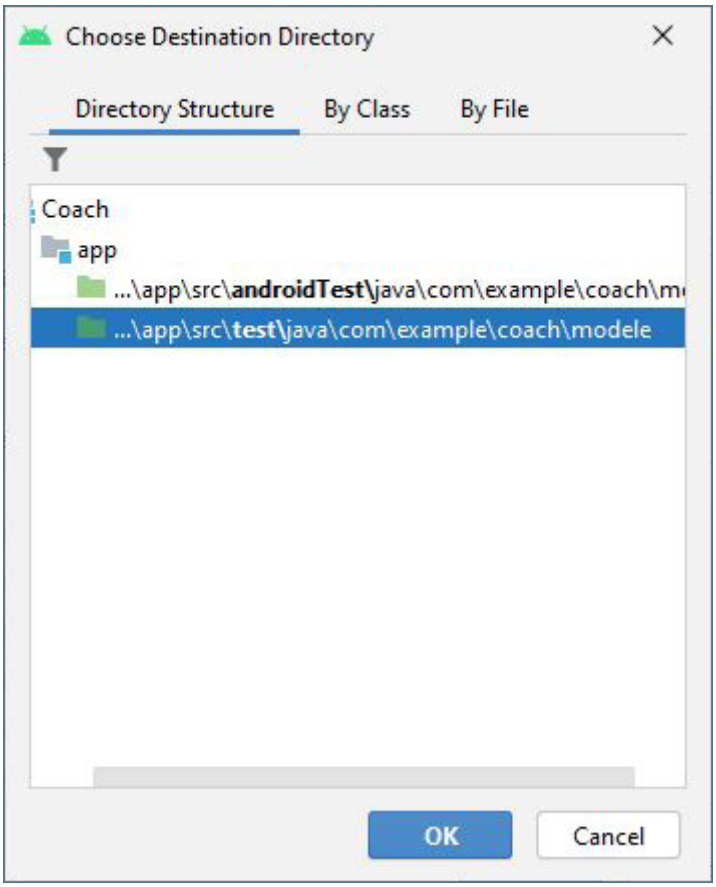
Voilà une classe qui se prête très bien à des tests unitaires pour contrôler si les 2 méthodes fonctionnent correctement. Donc vous allez tout de suite voir comment générer des tests unitaires.

Si entre temps vous avez fermé la classe Profil, rouvrez-là et positionnez-vous dessus.

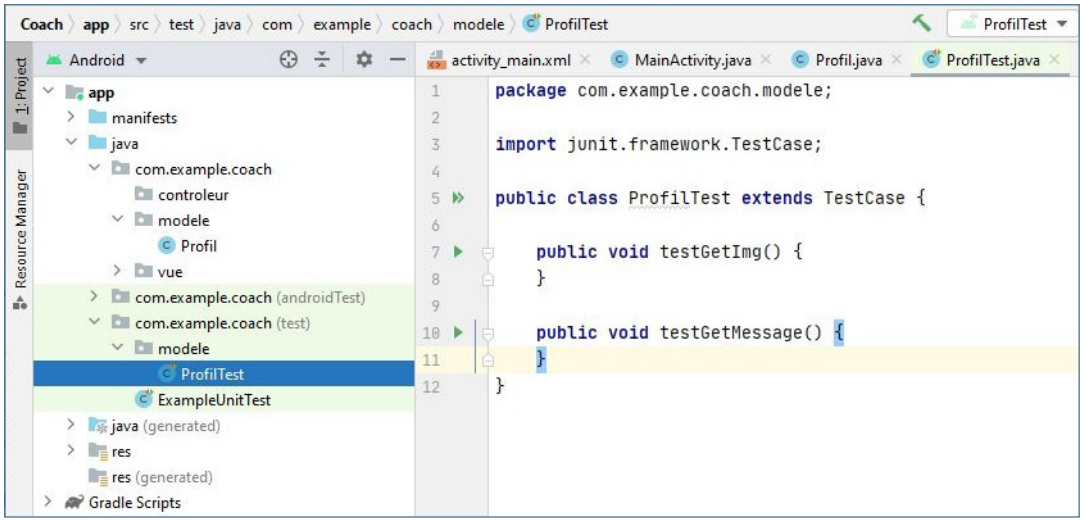
Allez dans le menu "Navigate > Test". Dans le menu flottant qui apparait, vérifiez que le titre est "Choose test for Profil" et sélectionnez "Create New Test". Dans la fenêtre qui apparait, sélectionnez les méthodes getImg() et getMessage() comme sur la capture. Le but est de tester si les valeurs retournées sont correctes.



Faites ok. Dans la nouvelle fenêtre, choisissez "\app\src\test..." et non "\app\src\androidTest" car les tests que l'on va écrire ne nécessitent pas l'émulateur, puis faites ok :



Une classe de test a été créée dans le package de test, et s'est ouverte.



Repérez où a été créée cette classe (dans le package "com.example.coach (test)" sous package "modele" (qu'il faut développer). En fait, en parallèle du projet, on peut avoir un ensemble de classes de tests avec la même structure que le projet.

Commençons par déclarer dans cette classe de tests, des propriétés qui vont servir d'exemple :

```
// création d'un profil : femme de 67kg, 1m65, 35 ans
private Profil profil = new Profil(67, 165, 35, 0);
// résultat de l'img correspondant
private float img = (float)32.4 ;
// message correspondant
private String message = "trop élevé" ;
```

Vous vous doutez que le but serait en réalité de tester tous les cas de figure les plus classiques. Là, nous allons nous contenter de faire un seul test pour montrer la démarche.

Maintenant, dans la méthode testGetMessage(), mettez la ligne de code :

```
assertEquals(message, profil.getMessage());
```

Et dans la méthode testGetImg(), ajoutez :

```
assertEquals(img, profil.getImg(), (float)0.1);
```

À noter que le 3e paramètre `((float)0.1)`, permet juste de dire que la comparaison entre les 2 nombres de type float va se faire à 0.1 près.

assertEquals permet de faire une comparaison d'égalité entre 2 valeurs.

Enregistrez.

Vous l'avez compris, on va tester si les méthodes appliquées à l'objet retournent les valeurs attendues.

Maintenant que la classe est terminée, on va la tester. Faites un clic droit directement sur le nom de classe ProfilTest et cliquez sur "Run ProfilTest". Le test se lance. Vous verrez le résultat en bas. En bas à gauche, vous voyez le nom des 2 méthodes. À côté de chaque méthode vous retrouvez le logo correspondant, qui vous permet de savoir si la méthode marche ou non (vert ou orange). J'ai bien sûr volontairement fait en sorte qu'un test marche et pas l'autre. Dans la partie centrale du bas, vous avez directement l'explication de l'erreur. Vous pouvez aussi cliquer en bas à gauche sur chaque méthode pour voir les messages correspondants (aucun message sur testGetMessagequi marche, un message précis sur testGetImg qui ne marche pas).

Voici l'affichage obtenu en sélectionnant testGetImg en bas à gauche :

Le test sur le message est passé, pas celui sur le calcul de l'img. Et l'information est clairement affichée :

```
Expected :32.4
Actual   :32.181683
```

Effectivement, si vous regardez la capture qui était donnée au début guide 1, qui correspond justement aux valeurs exemple du test, le résultat de l'img était de 32.2 (arrondi à un chiffre après la virgule).

Le but est de corriger l'erreur : dans la classe de test, modifiez 32.4 par 32.2 et enregistrez.

Pour relancer le test, il suffit de cliquer sur la flèche verte en bas à gauche (que vous pouvez voir sur la capture juste au-dessus). Cette fois, les 2 tests ont réussi.

Si vous obtenez une erreur différente sur le calcul de l'img, c'est que votre formule dans la classe Profil n'est pas bonne : à vous de la corriger.

Voilà : c'était pour vous donner une idée du fonctionnement des tests unitaires.

Attention, lors de votre prochain test de l'application (en cliquant sur la flèche verte tout en haut) il y a des chances que ce soit le test unitaire qui soit lancé. Pour éviter ce problème, vous passerez par le menu "Run > Run... > app". Faites le maintenant pour ne pas oublier puis stoppez l'application avec le carré rouge.

Pour éviter de tout mélanger, je vous conseille de réduire le package de test quand vous ne l'utilisez pas, afin de ne voir que le package de l'application déployé.

1B. Le contrôleur qui décide

Le contrôleur doit gérer les demandes provenant de la vue ou du modèle.

Commençons par créer le contrôleur : faites un clic droit sur le package controleur, "New > JavaClass" et tapez "Controle" et validez. La classe est créée.

1B1. Démarrage de l'application

Classiquement une application démarre sur le contrôleur. Mais ce n'est pas possible avec Android car c'est forcément une Activity qui se lance en premier.

Pour vous en persuader, ouvrez le fichier AndroidManifest.xml qui se trouve à gauche, dans "manifests". Repérez la ligne suivante :

```
<action android:name="android.intent.action.MAIN" />
```

Vous remarquez le "MAIN" qui permet de repérer le démarrage. Cette ligne se trouve dans une balise activity. C'est donc une Activity qui va démarrer comme MAIN. Dans ce fichier, vous pourrez retrouver les

autres Activity que vous ajouterez plus tard dans votre application. C'est aussi dans ce fichier que vous ajouterez des paramétrages, du genre l'autorisation d'accès à internet. Pour le moment, vous pouvez fermer ce fichier.

1B2. Contrôleur singleton

L'Activity principale doit donc être en mesure de créer le contrôleur. Les éventuelles autres Activity qui seront créées doivent être capable de récupérer l'instance du contrôleur déjà créé, pour pouvoir faire appel à lui. Il faut donc une seule instance de contrôleur. Pour cela, il faut utiliser le pattern singleton qui ne permet qu'une seule instance pour une classe.

Commencez par rendre la classe Controle comme finale pour empêcher tout héritage :

```
public final class Controle {
```

Ajoutez la propriété "instance", du type de la classe (Controle), en la mettant en privé et static, et en l'initialisant à null. C'est cette propriété qui contiendra l'unique instance de la classe. C'est bien sûr forcément une propriété à portée de classe (static) car elle est unique et va être manipulée par une méthode statique (les méthodes statiques ne peuvent manipuler que les propriétés statiques).

Ajoutez le constructeur Controle sans paramètre, mais attention, en le mettant en privé (pas question de pouvoir instancier directement la classe). Dans le constructeur, mettez l'instruction :

```
super() ;
```

C'est pour éviter de laisser le constructeur vide : il fait appel au constructeur de la classe mère (ici il n'y a pas de classe mère, excepté bien sûr la classe Object, mère de toutes les classes).

Il reste maintenant à créer la méthode qui va permettre soit de créer l'instance (si elle n'existe pas encore) soit de retourner l'instance déjà créée. Créez la méthode getInstance() qui est publique, finale (pas le droit de la redéfinir) et bien sûr statique (on y accèdera directement par le nom de la classe). Cette méthode doit retourner une instance, donc un objet de type Controle.

Dans cette méthode, commencez par faire un test sur le contenu de Controle.instance. Si le contenu est null, alors affectez à Controle.instance une instance de la classe :

```
Controle.instance = new Controle();
```

Le constructeur privé sera alors sollicité.

Après la fermeture du test, donc dans tous les cas, retournez Controle.instance.

Vous l'avez compris, dans une Activity, il ne sera pas possible de faire directement "new Controle()" car le constructeur est privé. En revanche, on pourra faire "Controle.getInstance()" qui permettra de récupérer la seule instance de Controle (soit une nouvelle si aucune instance n'est encore créée, soit l'instance déjà créée).

1B3. Actions du contrôleur

Dans l'application, le but est pour le moment de réagir au clic du bouton "calcul" pour obtenir l'affichage de l'IMG et du message. Donc, le contrôleur va être sollicité par la vue en recevant les valeurs saisies et en retournant l'IMG et le message.

Les méthodes suivantes qui vont être créées dans la classe Controle ne sont pas statiques : elles seront appelées à partir de l'instance créée de Controle.

Dans la classe Controle, créez la méthode creerProfil qui ne retourne rien et qui reçoit en paramètre les 4 informations saisies (poids, taille, age, sexe). Le but va être de gérer un profil avec ces informations. Au même endroit de la déclaration de la propriété instance, déclarez en privé et static la propriété profil de

type Profil. Normalement Profil apparait en rouge car cette classe est inconnue ici. Il manque l'import. Si vous cliquez sur Profil et laissez la souris dessus, il devrait apparaître souligné avec un message vous invitant à faire "alt+Entrée". Faites-le : l'import devrait s'insérer automatiquement et Profil n'est plus en rouge. Si vous n'y arrivez pas, insérez par vous-même l'import (juste en dessous de ligne précisant le package) :

```
import com.example.coach.modele.Profil;
```

Dans la méthode creerProfil, valorisez l'objet profil en lui affectant une instance de la classe Profil, avec les bons paramètres.

Créez la méthode getImg() qui ne reçoit aucun paramètre et qui retourne l'img récupéré dans le profil. De même, créez la méthode getMessage() qui ne reçoit aucun paramètre et qui retourne le message récupéré dans le profil.

1B4. Documentation technique

Pour le moment le contrôleur est terminé. Nous avons vu précédemment l'intérêt des tests unitaires et la puissance de l'IDE qui est capable de les générer automatiquement. Voyons ici la même chose pour la documentation technique.

Placez le curseur une ligne au-dessus du nom de la méthode "creerProfil" (sur une ligne vierge) et validez pour que le curseur se place juste au-dessus du "p" de "public", donc au même niveau d'indentation, puis tapez :

```
/**
```

Et validez. Vous allez voir que la structure du commentaire va se créer automatiquement juste au-dessus de la méthode :

```
/**
 *
 * @param poids
 * @param taille
 * @param age
 * @param sexe
 */
```

Les paramètres se sont placés. Vous pouvez maintenant compléter le bloc de commentaires, par exemple comme ceci :

```
/**
 * Création du profil
 * @param poids
 * @param taille en cm
 * @param age
 * @param sexe 1 pour homme, 0 pour femme
 */
```

Faites de même pour les autres méthodes de la classe ainsi que la classe Profil.

Il est temps de voir comment la documentation technique peut alors se générer automatiquement. Il faut d'abord réserver un emplacement pour la documentation. Sur votre disque, créez un dossier (par exemple "CoachDoc" où vous voulez mais pas dans votre dossier de projet : personnellement je l'ai mis au même niveau hiérarchique que le dossier de projet Coach). Ce dossier va recevoir les pages html de la documentation technique qui va être générée automatiquement.

Dans Android Studio, partie gauche, sélectionnez le package principal (juste en cliquant sur com.example.coach). Effectivement, on peut générer une documentation à n'importe quel niveau, par exemple juste pour une classe si on ne sélectionne qu'une seule classe. Ici on veut la documentation de toute l'application.

Sélectionnez "Tools > Generate javadoc". Dans la fenêtre qui s'ouvre, dans "Output directory", cliquez sur le dessin du petit dossier pour aller sélectionner votre dossier CoachDoc.

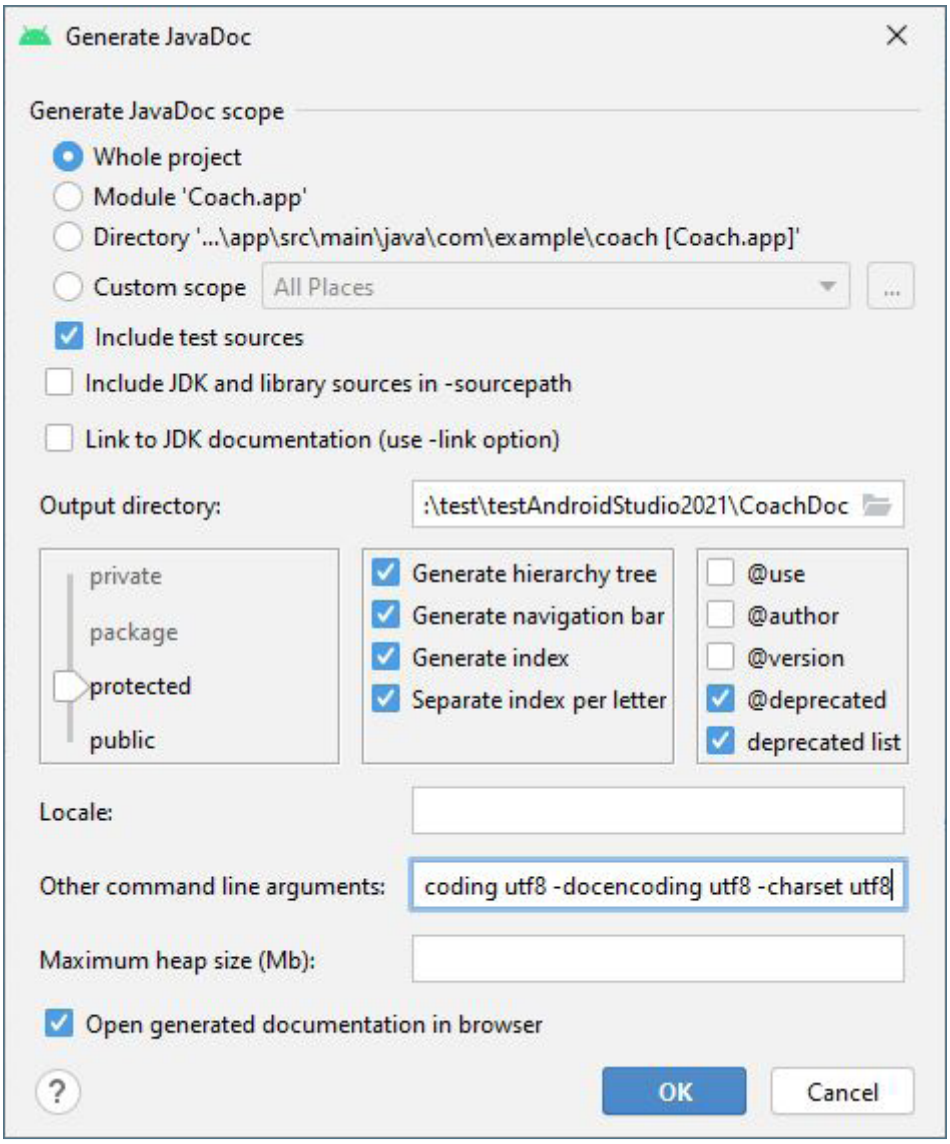
Remarquez le niveau qui est par défaut sur "protected". En effet, normalement une documentation technique ne laisse apparaitre que les membres publics ou protégés. Ceci dit, si vous aviez envie de générer une documentation par exemple pour vos collègues qui devraient travailler sur l'application, là il serait possible de mettre le niveau private.

Au niveau de la ligne "Other commande line arguments", tapez le code suivant :

```
-encoding utf8 -docencoding utf8 -charset utf8
```

Cela va permettre de ne pas avoir de souci au niveau des accents, dans la documentation.

Voici l'écran que vous devriez avoir (votre chemin output pouvant être évidemment différent) :



Contrôlez que la case tout en bas, "Open generated..." est cochée : cela vous permettra de voir directement la documentation dans le navigateur. Faites ok.

La documentation est générée. Il est possible que des erreurs ou warnings apparaissent, souvent des oublis de commentaires, mais normalement vous obtenez tout de même la documentation. Vous devriez donc la voir. Si elle ne s'ouvre pas automatiquement, allez dans le dossier CoachDoc et lancez le fichier index.html. Si vous obtenez une erreur sur R non reconnu, alors, dans MainActivity, remplacez "R" par "*" dans l'import : `import com.example.coach.*;`

Enregistrez et recommencez la génération de la doc.

Baladez-vous dans la documentation pour voir comment elle est organisée et surtout pour retrouver les informations que vous avez saisies dans les zones de commentaires. Donc en particulier, allez dans le package "com.example.coach.controleur" et, en bas à gauche, cliquez sur la classe Controle. Vous allez retrouver, au milieu, les différentes méthodes créées, et en descendant, remarquez les commentaires que vous aviez mis.

Pour la suite, prenez l'habitude d'ajouter systématiquement les commentaires normalisés au fur et à mesure de l'écriture du code. Surtout ne faites pas l'erreur de dire "j'écris le code et quand j'aurais tout fini, je mettrai les commentaires". La documentation technique doit pouvoir être générée à tout moment, sans avoir à intervenir sur le code.

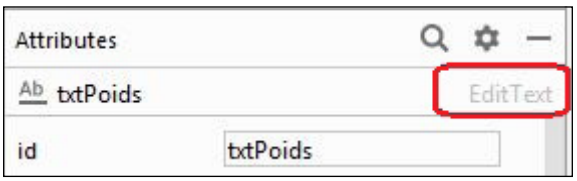
1C. La vue qui affiche et demande

Maintenant que le modèle est construit pour mémoriser les informations, et que le contrôleur permet de fournir tous les services nécessaires, il reste à gérer le code de la vue.

1C1. Gestion des objets graphiques

Dans un premier temps, le but est d'accéder aux objets graphiques pour pouvoir récupérer leur contenu (getter) et si nécessaire modifier le contenu (setter).

Dans la classe MainActivity, déclarez en privé les propriétés txtPoids, txtTaille, txtAge, rdHomme, lblIMG, imgSmiley et btnCalc, en respectant leurs types qui ont été donnés dans activity_main.xml (par exemple, txtPoids est de type EditText, lblIMG de type TextView...).



Faites les imports nécessaires s'ils n'ont pas été faits automatiquement. On a donné aux propriétés les mêmes noms que les id des objets graphiques correspondants, pour que ce soit plus clair, mais ce n'est pas du tout une obligation.

Le but est de faire le lien entre ces variables et les objets graphiques. Créez une méthode privée init() qui ne reçoit aucun paramètre et qui ne retourne rien, mais qui permet de faire ce lien en valorisant les propriétés concernées. Pour cela, ajoutez cette première ligne :

```
txtPoids = (EditText) findViewById(R.id.txtPoids) ;
```

La méthode findViewById permet de récupérer un objet graphique à partir de son id. findViewById peut récupérer n'importe quel type d'objet graphique, et du coup retourne un objet de type View. Voilà pourquoi il faut ensuite transtyper pour obtenir le bon type.

En suivant la même logique, faites le lien avec les 6 autres objets graphiques.

Cette méthode init() doit bien sûr être appelée à un moment ou à un autre. Le mieux est de l'appeler en fin de méthode onCreate().

1C2. Lien avec le contrôleur

La vue doit pouvoir solliciter le contrôleur.

Au niveau des propriétés, commencez par déclarer en privé un objet controle de type Controle.

En fin de méthode init(), vous allez créer le controleur en affectant à la propriété controle, l'appel de la méthode statique getInstance de la classe Controle. Je vous rappelle qu'on n'instancie pas directement la classe avec new car le constructeur est privé et que le but est de limiter le nombre d'instances à une.

1C3. Gestion d'un événement

Le seul événement à gérer pour le moment est le clic sur le bouton "Calculer". Pour gérer cet événement, commencez par créer en privé une méthode ecouteCalcul() qui ne reçoit aucun paramètre et qui ne retourne rien. Appelez cette méthode en fin de méthode init().
Dans la méthode ecouteCalcul(), ajoutez le code suivant :

```
btnCalc.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {

    }
});
```

Pensez à faire les imports demandés.

Explication de ce code partie par partie :

```
btnCalc.setOnClickListener(new Button.OnClickListener() { ... });
```

Sur le bouton, on applique la méthode setOnClickListener qui permet d'affecter un listener (donc une écoute) afin de capturer l'événement du clic sur le bouton.

La méthode setOnClickListener attend en paramètre un objet qui implémente l'interface Button.OnClickListener, d'où ce raccourci d'écriture très puissant, avec "new Button.OnClickListener" qui crée un objet... Mais remarquez bien qu'après les (), la parenthèse fermante du setOnClickListener (ici en rouge) n'apparait pas tout de suite. En effet, on remarque une accolade ouvrante, et plus loin l'accolade fermante suivi cette fois de la fameuse parenthèse qui doit fermer les paramètres du setOnClickListener.

Ce raccourci d'écriture permet de créer l'objet et, à la volée, de redéfinir une méthode imposée par l'interface OnClickListener : c'est la méthode OnClick.

```
public void onClick(View v) {

}
```

Il ne reste plus qu'à mettre dans cette méthode le code que vous voulez voir s'exécuter lorsque l'utilisateur clique sur le bouton. Pour faire un essai, dans cette méthode, ajoutez la ligne de code suivante :

```
Toast.makeText(MainActivity.this, "test", Toast.LENGTH_SHORT).show();
```

Et ajoutez l'import nécessaire pour accéder à Toast.

La méthode statique makeText de la classe Toast permet d'afficher quelques instants un message sur le téléphone.

Essayez tout de suite d'exécuter l'application. Attention, pour exécuter l'application, si vous cliquez directement sur la flèche verte, vous risquez de lancer le test unitaire précédent, car l'IDE garde en mémoire la dernière exécution. Donc pour lancer l'application, faites "Run > run..." et sélectionner "app". N'oubliez pas aussi de faire "stop" avant chaque nouvelle exécution.

Une fois que l'application s'exécute, cliquez sur le bouton : vous devriez voir apparaitre quelques instants le mot "test".

Si tout marche correctement, vous allez pouvoir enlever cette ligne de code pour passer à la suite. Pensez à stopper l'application.

1C4. Gestion de l'affichage du résultat

La suite du code va donc se faire dans cette méthode.

Le but va être de récupérer les informations saisies et de les convertir en Integer. Le problème est que les zones des objets graphiques peuvent très bien ne pas être remplies par l'utilisateur. Ce que l'on veut récupérer, c'est une bonne valeur, ou à défaut 0 si rien n'est saisi.

Pour cela, commencez par déclarer 3 variables locales (poids, taille, age) de type Integer et initialisez-les à 0.

Ensuite, on va tenter de récupérer les saisies, mais attention, uniquement si elles sont convertibles en Integer (ceci dit, avec la nouvelle version d'Android Studio qu'on a utilisée, les zones de saisies sont obligatoirement numériques, mais c'est l'occasion de voir comment éviter les erreurs dans un cas similaire). Il faut donc placer ces récupérations dans un try/catch qui ne se fera que si les 3 saisies sont correctes. Voici la structure :

```
try {
    poids = Integer.parseInt(txtPoids.getText().toString());
    taille = Integer.parseInt(txtTaille.getText().toString());
    age = Integer.parseInt(txtAge.getText().toString());
}catch(Exception e){}
```

Vous remarquez que le catch, qui capture l'erreur, ne fait rien car on veut juste laisser à 0 les trois variables et ne rien faire d'autre.

Une rapide petite explication sur la récupération :

txtPoids étant un objet graphique, on peut lui appliquer la méthode getText() qui permet de récupérer le contenu. La méthode toString() permet d'obtenir la version String de ce contenu. Et enfin, l'ensemble est parsé (transtypé) en integer pour être affecté à la variable poids. On a besoin d'avoir un integer pour pouvoir gérer les calculs.

Ces 3 variables ne sont pas des propriétés de la classe mais juste des variables locales à votre méthode. Elles ne vont vous servir que dans cette méthode.

En ce qui concerne le sexe, c'est un peu plus compliqué puisque c'est géré par des boutons radio. Déclarez une variable sexe de type Integer et initialisez-là à 0 (femme par défaut). Faites ensuite un test sur rdHomme en lui appliquant la méthode isChecked() qui retourne vrai si le bouton radio a été sélectionné. Dans ce cas, c'est un homme, donc affectez 1 à la variable sexe.

Une fois ces affectations gérées, faites un test sur poids, taille et age : si l'une de ces variables contient 0, cela veut dire qu'elle n'a pas été remplie. Du coup c'est inutile de faire le calcul : dans ce cas, affichez un message de quelques instants (avec la classe Toast, comme vous l'avez vu plus haut) en marquant "Veuillez saisir tous les champs".

Lancez l'application pour voir si, dans le cas où vous ne remplissez pas les 3 zones ou que vous ne saisissez pas des informations correctes, vous obtenez bien le message d'avertissement attendu.

Dans le cas contraire (donc dans le else du test précédent), vous allez pouvoir gérer l'affichage de l'image et des messages. Le mieux est de le faire dans une méthode séparée de ecouteCalcul(), donc vous allez créer une méthode privée affichResult que vous allez appeler dans votre test. Cette méthode reçoit en paramètre poids, taille, age et sexe et ne retourne rien.

Écrivez le contenu de cette méthode qui doit :

- créer le profil en utilisant la méthode creerProfil de la classe Controle (attention, il ne faut pas créer un objet profil mais juste appeler la bonne méthode sur l'objet controle pour demander au contrôleur de créer le profil, car la vue ne doit pas avoir directement accès au modele) ;
- récupérer dans des variables locales le message généré et le résultat du calcul de l'img (toujours en faisant appel au contrôleur) ;
- en faisant des tests sur le contenu de message (trois valeurs possibles), afficher l'image qui correspond en modifiant l'image de l'objet imgSmiley avec la méthode setImageResource qui attend en paramètre un drawable, par exemple R.drawable.normal pour l'image qui porte le nom "normal") ;
- afficher aussi la valeur de l'img (avec la méthode setText sur lblIMG), suivi de l'information " : IMG trop faible" (ou " : IMG normal", ou " : IMG trop élevé", suivant le cas) ;
- formater l'affichage de l'img pour qu'il n'y ait qu'un chiffre après la virgule : pour cela, utilisez la méthode statique format de la classe String, en mettant en premier paramètre "%.01f" et en second paramètre, la variable à formater, donc la variable qui a récupérée (l'img) ;
- pour finir, mettez le texte en vert si l'img est normal, en rouge dans les autres cas. Pour cela, appliquez à lblIMG la méthode setTextColor, en mettant en paramètre soit Color.GREEN, soit Color.RED.

Testez pour voir si tout marche correctement. Bon, j'avoue ne pas être persuadée du bon fonctionnement de la formule qui donne normal pour des valeurs parfois très faibles... Cependant, si vous saisissez des valeurs extrêmes, vous devriez obtenir les 3 possibilités de résultats avec les images et couleurs correspondantes.

Si l'application ne fonctionne pas correctement, vous pouvez utiliser le mode debug et/ou les affichages console avec Log.d.

Voilà, la première phase de l'application est terminée ! Vous avez, l'air de rien, vu les bases d'une application Android.

Si vous pouvez, faites un test de déploiement sur un support mobile.

2. Gestion de versions

Avant de poursuivre le codage sous Android, on va mettre en place la gestion de versions, qui d'ailleurs aurait dû être gérée depuis le début.

En rappel, ce n'est pas juste un outil de sauvegarde. La gestion de versions permet de mémoriser les différentes versions d'un projet, au fur et à mesure de sa création, de revenir sur une version antérieure si nécessaire, de comparer des versions mais aussi de travailler à plusieurs, avec des branches différentes, et de fusionner les travaux avec gestion de conflits, si nécessaire.

Pour la gestion de versions, on peut se contenter d'utiliser les outils locaux offerts par l'IDE : ceci n'est bien sûr valable que pour un projet individuel et cela n'offre pas la sécurité d'une sauvegarde distante. On peut installer et configurer un serveur de gestion de versions, sur son ordinateur ou sur un ordinateur séparé. Enfin, il est possible d'utiliser des sites spécialisés comme Github ou Bitbucket.

Android Studio permet toutes les possibilités : la gestion de versions locale mais aussi distante.

2A. Historique en local

Dans les dernières versions d'Android Studio, le dépôt en local est géré automatiquement.

Dans Android Studio, sélectionnez l'application (cliquez sur app, à gauche). VCS enregistre tout en local. Allez dans "VCS > Local history > Show history". Vous allez voir l'historique de l'évolution de vos différentes versions, au fur et à mesure que vous avez modifié le code. Si vous cliquez à gauche sur une ancienne version et que vous double cliquez au centre sur un fichier, vous verrez dans une nouvelle fenêtre la version "avant" et la version "après" la modification.

Vous pouvez fermer les fenêtres.

2B. Dépôt sur internet

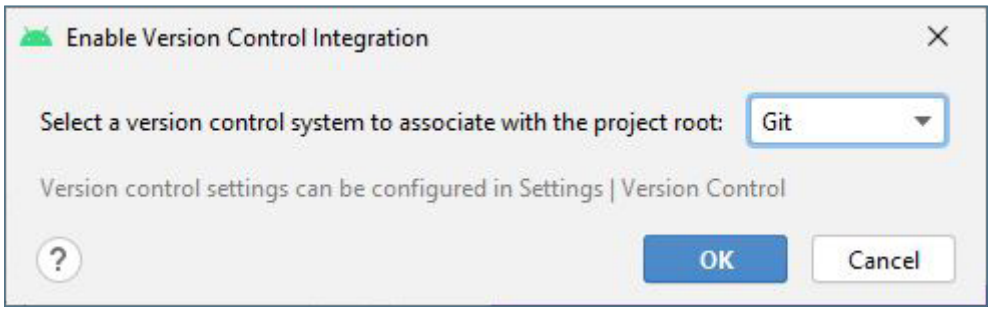
Il est possible de gérer le dépôt sur un serveur distant, et pourquoi pas sur internet. Pour tester, on va utiliser un site de gestion de dépôts pour Git : Bitbucket. C'est l'occasion de voir une autre solution en ligne que GitHub.

Voici les étapes à suivre.

Avant de suivre toutes ces étapes, il est fortement conseillé de faire une sauvegarde du projet dans un dossier séparé. Ainsi, si vous rencontrez un problème, vous pouvez revenir à la version précédente (après avoir fermé le projet sous Android Studio, fermé Android Studio, supprimé le projet qui pose problème, recopié les bons fichiers sur le disque, réouvert Android Studio et recherché le projet sur le disque).

Configuration de Git dans Android Studio :

- allez sur le site de GIT (http://git-scm.com/downloads) et récupérez la version correspondant à votre système ;
- installez GIT avec les options par défaut ;
- sous Android, à gauche, sélectionnez "app". Allez dans le menu "VCS > Enable Version Control Integration...". Dans la fenêtre qui s'ouvre, sélectionnez "Git" comme ci-dessous puis faites OK.

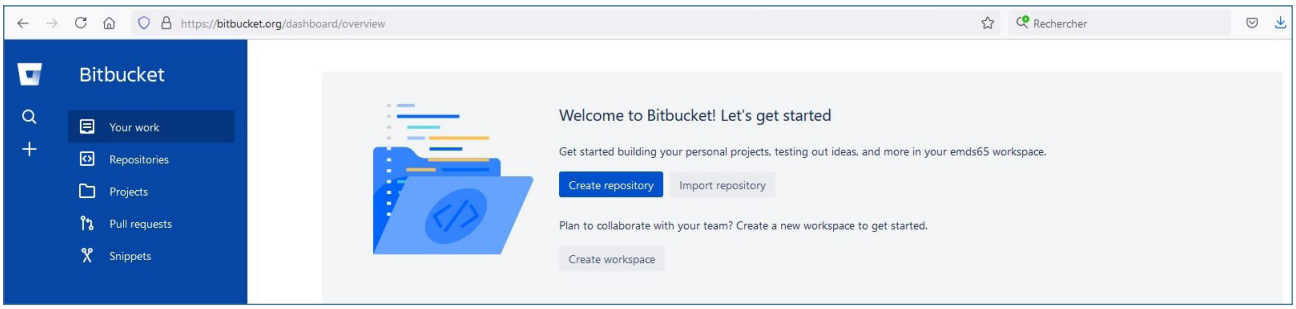


- remarquez que plusieurs raccourcis Git ont été ajoutés et que le contenu du menu VCS a changé ;
- vérifiez directement sur le disque que le dossier ".git" a été créé dans le dossier de projet (attention c'est un dossier caché : si vous ne le voyez pas, allez dans les paramètres des dossiers pour faire en sorte de voir les fichiers cachés) ;
- dans Android Studio, remarquez que les fichiers du projet sont en rouge : c'est pour signaler qu'ils n'ont pas été ajoutés au dépôt et commités ;
- sélectionnez l'application (cliquez sur "app", à gauche) ;
- allez dans "VCS > Git > Add" (les fichiers sont maintenant en vert) puis "VCS > Git > Commit directory..." La partie gauche a changé. Remarquez que vous êtes maintenant sur "Commit" (message vertical à gauche) et non sur "Project" (comme précédemment). En bas à gauche se trouve la zone de saisie du message du commit. Il est conseillé de mettre à chaque fois un message précis quand un commit est réalisé. Par exemple, vous pourriez mettre ici : "Application fonctionnelle en MVC avec une activity et le calcul de l'IMG. Tests unitaires." Cela vous permet de mieux repérer à quoi correspond le commit. Ensuite cliquez sur "Commit". Il est possible qu'une petite fenêtre s'ouvre signalant des erreurs ou warnings. Ce n'est pas grave : cliquez sur "Commit" (et non pas "Review") ;
- android a fait un commit en local (voir le message vert que vous avez en bas de l'écran) :

Revenez sur "Project" (en cliquant à gauche). Remarquez aussi que les fichiers ne sont plus en rouge (non ajoutés pour la préparation au commit) ni en vert (non commités)..

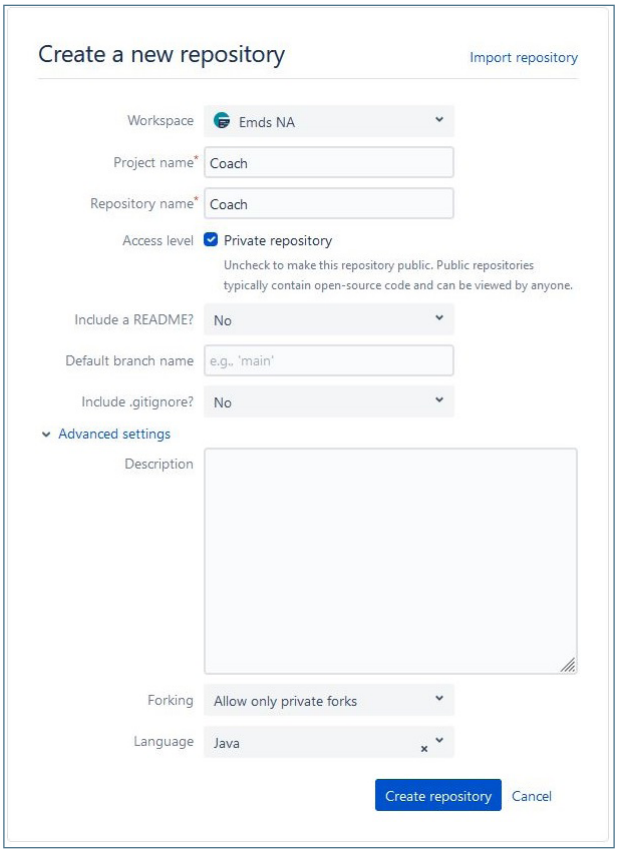
Création du dépôt sous Bitbucket :

— allez sur https://bitbucket.org/, créez un compte "free" et suivez les étapes. Après toutes les étapes d'inscription, entre autres la validation de votre adresse mail, vous devriez obtenir l'écran suivant (ou quelque chose de similaire) :



Jusqu'à 5 utilisateurs pourront accéder à ce compte et travailler en même temps.

— Sur le site, une fois le compte créé, choisissez "Create repository". Donnez un nom au projet (Coach), un nom au repository (Coach), laissez coché "private repository". Dans le combo "Include a README?", sélectionnez "No". Dans le combo "Include .gitignore?", sélectionnez "No". Cliquez sur "Advanced settings". Dans language, sélectionnez Java. La fenêtre devrait ressembler à ceci :



Surtout, n'oubliez pas de choisir "No" pour "Include a README?"et "No" pour "Include .gitignore?" car sinon Bitbucket va créer un dépôt avec un premier commit contenant un readme et/ou le fichier ".gitignore" et du coup vous ne pourrez plus lier votre dépôt local à Bitbucket.

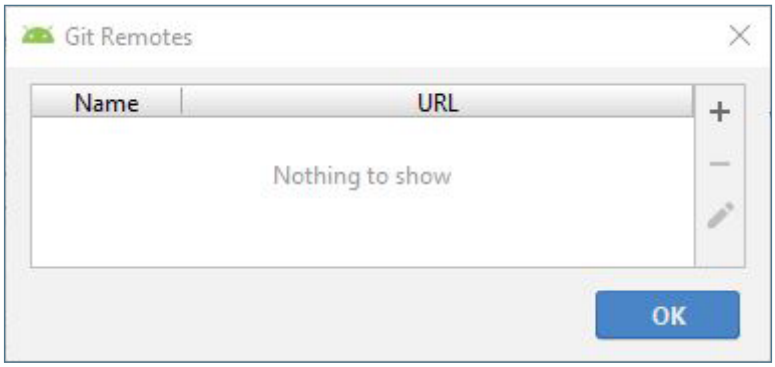
— Cliquez sur "Create repository".

— Vous devriez arriver sur une page qui contient la démarche pour faire le lien entre le dépôt local et celui sur BitBucket, en ligne de commandes. Cependant, Android Studio offre les commandes nécessaires pour faire le lien. Dans un premier temps, repérez et copiez l'adresse de votre reposi-

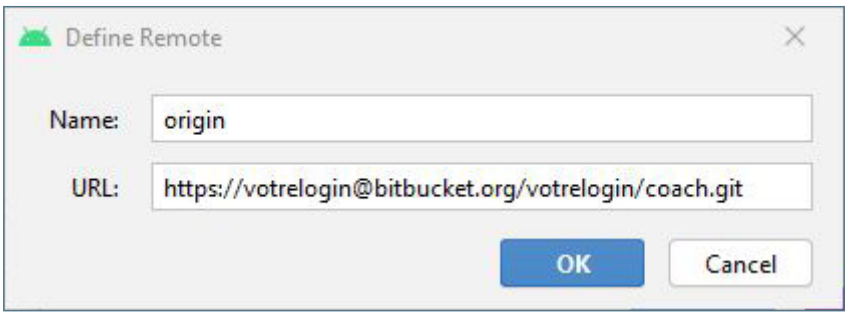
tory, donné dans une des lignes, qui doit ressembler à ceci :
https://votrelogin@bitbucket.org/votrelogin/coach.git
avec bien sûr "votrelogin" à remplacer par le login que vous avez donné lors de l'inscription.

Création du lien entre le git local sous Android Studio et Bitbucket :

- sous Android Studio, sélectionnez à gauche "app" ;
- Menu "VCS > Git > Remotes...". Vous obtenez la fenêtre suivante :



Vous allez pouvoir donner l'adresse du serveur en lien avec le Git. Pour cela, cliquez sur "+". Vous obtenez la fenêtre suivante, dans laquelle vous allez copier l'url de votre dépôt (que vous avez repéré un peu plus haut, sur le site de bitbucket).



Faites OK. Normalement le mot de passe de connexion à Bitbucket est demandé. Saisissez-le puis "Log In".

- Vous retombez sur la fenêtre précédente, mais cette fois il y a une ligne dans la liste (le dépôt sur Bitbucket a été trouvé). Faites OK.
- Menu "VCS > Git > Push...". Une fenêtre s'ouvre. L'idée est d'envoyer (push) vers Bitbucket, le dépôt local. Cliquez sur Push. Normalement vous devriez obtenir en bas à droite, le message "Pushed mater to new branch origin/master" (au bout d'un petit moment).
- Dans Bitbucket, rafraichissez la page : vous devriez voir cette fois le dossier app qui contient tout le projet.

Gestion des commits :

- Par la suite, sous Android Studio, quand vous voulez commiter (en local et à distance), il suffira de se placer sur "app" puis de faire "VCS > Git > Commit Directory...", de mettre un "Commit message" (sachant que par défaut c'est l'ancien message qui est marqué) puis cette fois, plutôt que de faire "commit" et ensuite "Push", vous pourrez directement cliquer sur la petite flèche à côté du bouton "Commit" et sélectionner "Commit and Push".

Si vous ajoutez de nouveaux fichiers, il faudra avant tout faire un clic droit sur app (à gauche) ou sur le fichier concerné, et "Git > Add" avant de faire le "commit and push".

CNED, BP 60200, 86980 Futuroscope Chasseneuil Cedex, France

87D22TDWB1A21