



Testez !

- Cf. projet Demo_cycledevie_Activity
 - <https://gitlab.com/m2eservices/democycledevie.git>
- Lancez l'appli et une fois lancée, tournez l'écran !



Ressources

- Pour accéder aux ressources, il suffit de connaître leur type et leur identifiant

`([android.]R.type_de_ressource.nom_ressource).`

- Exemple **OK** :

```
// Fixe la mise en page d'une activité
setContentView(R.layout.ecran_de_demarrage);
```

- Exemple **pas OK** car **on renvoie toujours un ID** et pas la string attendue !

```
// Création par copie d'une chaîne de caractères
String titre = new String(R.string.texte_titre_ecran);
```

- **Solution** :

```
Resources resources = getResources();
String nom = resources.getString(R.string.texte_titre_ecran);
```



Ressources référencées par d'autres ressources

- Vous pouvez également utiliser vos ressources comme valeurs d'attributs dans d'autres ressources sous forme XML.
- Cette possibilité est très utilisée dans les mises en page par exemple.
 - Texte affiché
 - Style utilisé
 - Dimension utilisée
 - Couleur utilisée...
- La notation pour faire référence à une autre ressource est la suivante :
attribute="@[package_name:]resource_type/resource_identifiant"
- Exemple :
`<TextView android:text="@string/table_contenu_cellule_gauche" />`
- Vous pouvez aussi écrire vos propres ressources
 - Par exemple pour définir des configurations différentes de mise en page



Utilisation de ressources systemes

- Il suffit d'utiliser la classe **android.R**.
 - Exemple : **android.R.drawable.ic_dialog_alert**
- Pour accéder à ces ressources dans un fichier XML, il faut spécifier « android » comme espace de nommage.
 - Exemple : `<... android:text="@android:string/unknownName"/>`



Créer des ressources

- Par convention, on sépare les types de ressources, par exemple res/values/**strings.xml** pour les **strings** etc.
- Exemple (strings.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="nom_application">Suivi des SMS</string>
    <string name="auteur_application">JCT</string>
</resources>
```
- Rappel : Internationalisation « facile »
 - Créer des dossiers pour chaque langue et Android se charge du reste



Définir des couleurs

- Une couleur définit une valeur RVB (rouge, vert et bleu) et une transparence.
- Il existe différents formats dont la syntaxe globale est la suivante :
`<color name=nom_couleur>valeur_de_la_couleur</color>`
- Les différents formats d'une couleur sont les suivants :
 - #RGB (Rouge-Vert-Bleu/valeur hexadécimale sur 1 caractère [0,16])
 - #ARGB (Alpha (transparence)-Rouge-Vert-Bleu)
 - #RRGGBB (Rouge-Vert-Bleu/valeur hexadécimale sur 2 caractères [0,255])
 - #AARRGGBB

```
<resources>
    <color name="bleu_transparent">#50FF00FF</color>
</resources>
```

- Exemple d'utilisation des couleurs :
 - En Java : `R.color.bleu_transparent`
 - En XML : `@[package:]color/bleu_transparent`



Définir des chaînes de caractères

- Syntaxe :
`<string name=nom_chaine>valeur_de_la_chaine</string>`
- Ce format permet d'utiliser trois balises HTML standard ``, `<i>` et `<u>`
`<string name="ex1">Un texte mis en forme</string>`
- Note : si vous utilisez des guillemets ou des apostrophes, vous devez les 'échapper' en les faisant précéder du caractère slash ('\\').

```
<resources>
    <string name="app_name">Exemple Android</string>
    <string name="menu_principal">Menu Principal</string>
</resources>
```

- Utilisation des chaînes de caractères :
 - Java : `R.string.le_nom`
 - XML : `@[package:]string/le_nom`

Pensez à convertir vos chaînes de caractères en ressources au fur et à mesure de votre développement



Définir des unités de mesure

- Les unités prises en charge par Android sont:
 - px (pixels), in (pouces), mm (millimètres), pt (points), dp («density-independant» pixel), sp («scale-independant pixel»).

```
<resources>
    <dimen name="taille_texte">5sp</dimen>
</resources>
```

- Utilisation des dimensions :
 - Java : `R.dimen.un_nom`
Exemple : `Resources.getDimen(R.dimen.taille_texte);`
 - XML : `@[package:]dimen/un_nom`

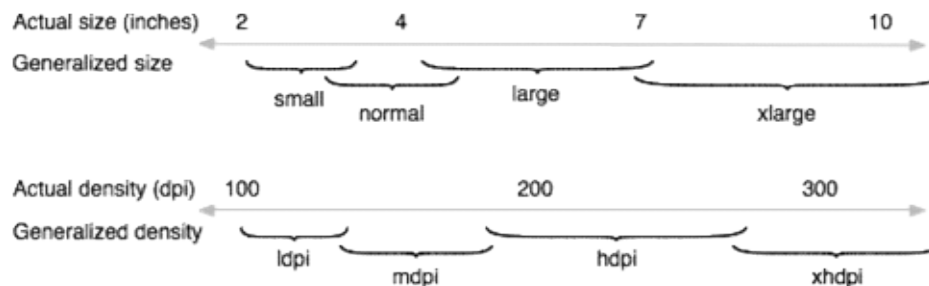
```
<TextView android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="@dimen/taille_texte"/>
```




Autres ressources

- Ressources images : PNG, JPG et GIF
 - Java : `R.drawable.fichier_bitmap`
 - XML : `@[package:]drawable/fichier_bitmap`
- Balise `<supports-screens>` qui grâce aux attributs `android:smallScreens`, `android:normalScreens` et `android:largeScreens` permet de spécifier quelle(s) taille(s) d'écran votre application supporte.

« deprecated »...
Pensez aux « swxxx »...



- *xlarge* screens are at least 960dp x 720dp
- *large* screens are at least 640dp x 480dp
- *normal* screens are at least 470dp x 320dp
- *small* screens are at least 426dp x 320dp

Cf.
http://developer.android.com/guide/practices/screens_support.html pour gérer des tailles d'écrans différentes de façon efficace

- Autres ressources : menu (définis en XML), layouts (en XML)



Autres ressources

- Ressources images : PNG, JPG et GIF
 - Java : `R.drawable.fichier_bitmap`
 - XML : `@[package:]drawable/fichier_bitmap`
- **Note:** Place all your **launcher icons in the `res/mipmap-[density]/` folders, rather than the `res/drawable-[density]/` folders**. The Android system retains the resources in these density-specific folders, such as `mipmap-xxxhdpi`, regardless of the screen resolution of the device where your app is installed. This behavior allows launcher apps to pick the best resolution icon for your app to display on the home screen. For more information about using the mipmap folders, see [Managing Projects Overview](#).

For example, the following application resource directories provide different layout designs for different screen sizes and different drawables. Use the `mipmap/` folders for launcher icons.

```
res/layout/my_layout.xml           // layout for normal screen size ("default")
res/layout-large/my_layout.xml     // layout for large screen size
res/layout-xlarge/my_layout.xml    // layout for extra-large screen size
res/layout-xlarge-land/my_layout.xml // layout for extra-large in landscape orientation

res/drawable-mdpi/graphic.png      // bitmap for medium-density
res/drawable-hdpi/graphic.png      // bitmap for high-density
res/drawable-xhdpi/graphic.png     // bitmap for extra-high-density
res/drawable-xxhdpi/graphic.png    // bitmap for extra-extra-high-density

res/mipmap-mdpi/my_icon.png        // launcher icon for medium-density
res/mipmap-hdpi/my_icon.png        // launcher icon for high-density
res/mipmap-xhdpi/my_icon.png       // launcher icon for extra-high-density
res/mipmap-xxhdpi/my_icon.png      // launcher icon for extra-extra-high-density
res/mipmap-xxxhdpi/my_icon.png     // launcher icon for extra-extra-extra-high-density
```



Autres ressources

- On peut définir à la volée des attributs (**utilisé 100% du temps**)
 - exemple : un « id » auquel on fera référence + tard.

```
<TextView android:id="@+id/monText"/>
```

Utilisation : **R.id.monText**

```
TextView monTexte =  
    (TextView) findViewById(R.id.monText) ;  
monTexte.setText("Coucou c'est moi !") ;
```



Manifest.xml

- Android Studio : Fichier placé dans app/src/main du projet.
- Il décrit le contexte de l'application, les activités, les services, les récepteurs d'Intents (Broadcast receivers), les fournisseurs de contenu et les permissions.



Structure de Manifest

Détails à

<http://developer.android.com/guide/topics/manifest/manifest-intro.html>

Cf. slide suivant

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
  <uses-permission /> _
  <permission />
  <permission-tree />
  <permission-group />
  <instrumentation />
  <uses-sdk />
  <uses-configuration />
  <uses-feature />
  <supports-screens />
  <application> _
    <activity> _
      <intent-filter>
        <action />
        <category />
        <data />
      </intent-filter>
      <meta-data />
    </activity>
    <activity-alias>
      <intent-filter> ... </intent-filter>
      <meta-data />
    </activity-alias>
    <service> _
      <intent-filter> ... </intent-filter>
      <meta-data />
    </service>
    <receiver> _
      <intent-filter> ... </intent-filter>
      <meta-data />
    </receiver>
    <provider> _
      <grant-uri-permission />
      <path-permission />
      <meta-data />
    </provider>
    <uses-library />
  </application>
</manifest>
```



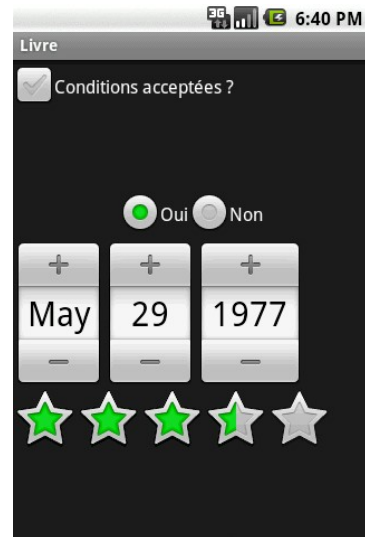
Structure de Manifest

- **<uses-permission>**
 - Les permissions qui seront déclarées ici seront un prérequis pour l'application. À l'installation, l'utilisateur se verra demander l'autorisation d'utiliser l'ensemble des fonctions liées à ces permissions comme la connexion réseau, la localisation de l'appareil, les droits d'écriture sur la carte mémoire...
- **<application>**
 - **Un manifeste contient un seul et unique nœud application qui en revanche contient des nœuds** concernant la définition d'activités, de services...
- **<activity>**
 - Déclare une activité présentée à l'utilisateur. **Si vous oubliez ces lignes de configuration, vos éléments ne pourront pas être utilisés.**
- **<service>**
 - Déclare un composant de l'application en tant que service.
- **<receiver>**
 - **Déclare un récepteur d'objets Intent.** Cet élément permet à l'application de recevoir ces objets alors qu'ils sont diffusés par d'autres applications ou par le système.
- **<provider>**
 - **Déclare un fournisseur de contenu qui permettra d'accéder aux données** gérées par l'application.

Cf. MAJ Android 6 !



IHM





Layout

- Une application utilise le layout créé soit en XML soit en Java :

```
//setContentView(R.layout.main);  
LinearLayout layout = new LinearLayout(this);  
TextView text = new TextView(this);  
text.setText(R.string.hello);  
layout.addView(text);  
setContentView(layout);
```

- mais XML + facile à gérer, à réutiliser, et permet le multilingue



Layout

- Les vues (c'est-à-dire tous les composants graphiques) héritent de View,
- Les vues peuvent être regroupées dans des ViewGroup.
 - De fait, les layouts héritent aussi de Viewgroup
- Les layout sont définis en XML (en général) dans res/layout.
 - Noms uniquement avec des minuscules et des lettres !
- ViewGroup (quelques uns...)
 - **LinearLayout**
 - les éléments sont alignés de gauche à droite ou de haut en bas (propriété orientation);
 - **RelativeLayout**
 - les enfants sont positionnés les uns par rapport aux autres, le premier enfant servant de référence aux autres ; tous les éléments doivent avoir un id.
 - **FrameLayout**
 - le plus basique des gabarits. Chaque enfant est positionné dans le coin en haut à gauche de l'écran et affiché par-dessus les enfants précédents, les cachant en partie ou complètement. Ce gabarit est principalement utilisé pour l'affichage d'un élément (par exemple, un cadre dans lequel on veut charger des images) ;
 - **TableLayout**
 - permet de positionner en lignes et colonnes à l'instar d'un tableau.
 - **ConstraintLayout** (très fortement recommandé maintenant)
- On peut aussi imbriquer des ViewGroup, faire des include, etc.
- Les layouts possèdent des attributs (height/width...) mais aussi **fill_parent** (remplacé par **match_parent** depuis API 8=Android 2.3) et **wrap_content**.
 - Fill/Match = remplit toute la place disponible,
 - Wrap = ne prend que ce qui est nécessaire en hauteur/largeur.



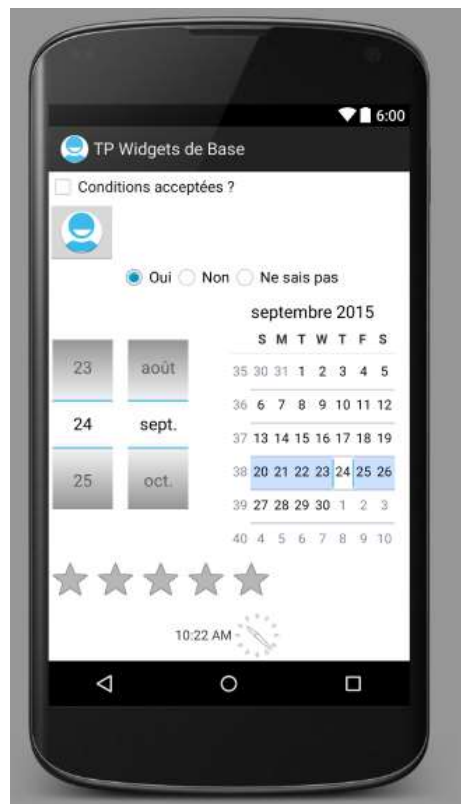
Unités de mesure

- Unités prises en charge par Android :
 - **pixel** (px) : correspond à un pixel de l'écran ;
 - **pouce** (in) : unité de longueur, correspondant à 2,54 cm. Basé sur la taille physique de l'écran ;
 - **millimètre** (mm) : basé sur la taille physique de l'écran ;
 - **point** (pt) : 1/72 d'un pouce ;
 - **pixel à densité indépendante (dp ou dip)** : une unité relative se basant sur une taille physique de l'écran de 160 dpi.
 - Avec cette unité, 1 dp est égal à 1 pixel sur un écran de 160 pixels.
 - Si la taille de l'écran est différente de 160 pixels, les vues s'adapteront selon le ratio entre la taille en pixels de l'écran de l'utilisateur et la référence des 160 pixels ;
 - **pixel à taille indépendante (sp)** : fonctionne de la même manière que les pixels à densité indépendante à l'exception qu'ils sont aussi fonction de la taille de polices spécifiée par l'utilisateur. **Il est recommandé d'utiliser cette unité lorsque vous spécifiez les tailles des polices.**
 - **Ces deux dernières unités sont à privilégier car elles permettent de s'adapter plus aisément à différentes tailles d'écran et rendent ainsi vos applications plus portables.** Notez que ces unités sont basées sur la taille physique de l'écran : l'écran ne pouvant afficher une longueur plus petite que le pixel, ces unités seront toujours rapportées aux pixels lors de l'affichage (1 cm peut ne pas faire 1 cm sur l'écran selon la définition de ce dernier).
- Si vous avez besoin d'intégrer une image avec une taille précise, préférez les valeurs en dip à celles en px.



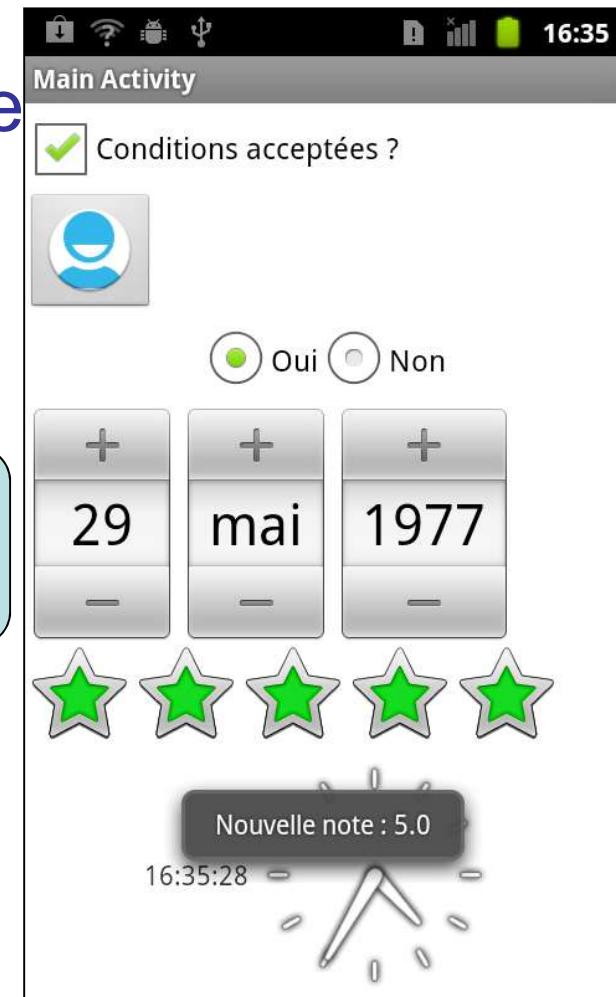
Exemple de widgets

- Cf. projet TP_Widgets_de_base
 - Nous verrons le code + tard



Affichage sur Galaxy
Tab 1

Affichage dans
Android Studio



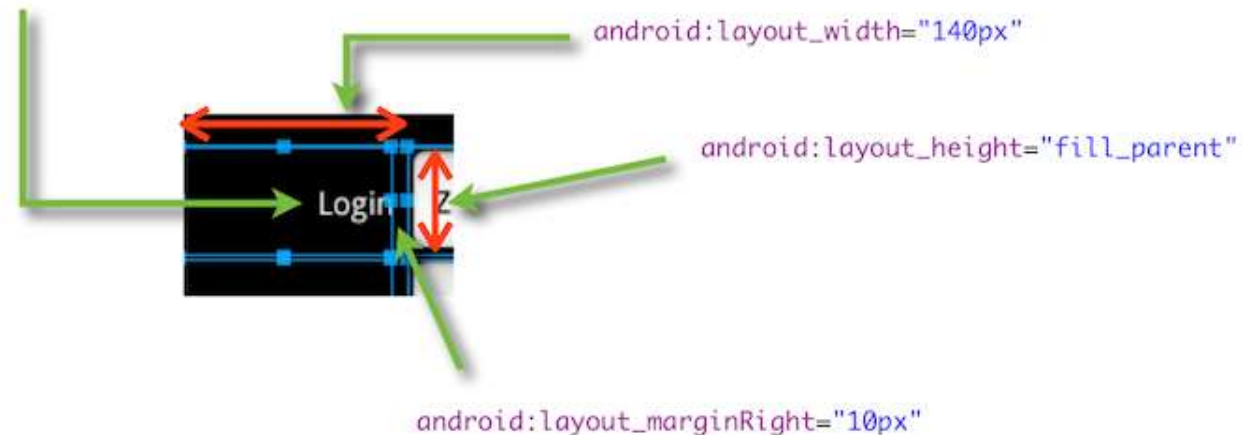


Quelques exemples de widgets

```
<TextView propriétés />
```

= label

```
android:text="Login"  
centré et justifié à droite = android:gravity="center_vertical|right"  
texte en blanc = android:textColor="#FFFFFF"
```



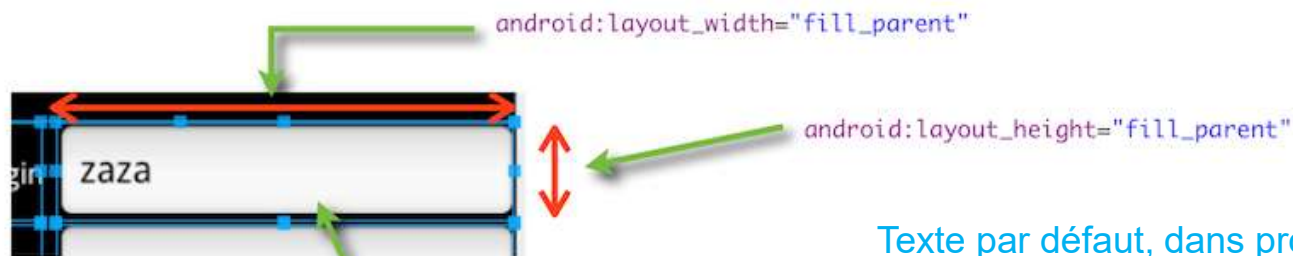
```
final TextView titre = (TextView)findViewById(R.id.id_vue);  
titre.setOnClickListener(new OnClickListener() {  
    public void onClick(View v) {  
        ...  
    }  
});
```



Quelques exemples de widgets

```
<EditText propriétés />
```

= champ de saisie



Texte par défaut, dans propriété « hint ».

Pour accéder à la valeur à partir du code,
il faut un identifiant = `android:id="@+id/loginField"`
Valeur par défaut : `android:text="zaza"`

// Récupérer

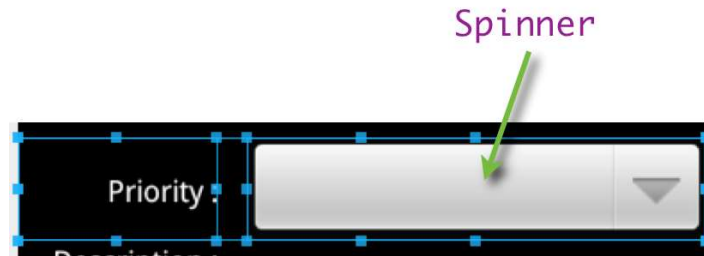
```
String texte=editText.getText()+"";  
// comme le texte des vues est de type CharSequence  
// il faut le convertir en String
```

// Fixer

```
editText.setText("bonjour");
```



Quelques exemples de widgets



Définir la liste de choix possibles en créant un tableau de String dans strings.xml ou dans arrays.xml :

```
<string-array name="mesChoix">
    <item>Choix 1</item>
    <item>Choix 2</item>
    <item>Choix 3</item>
</string-array>
```

et faire référence à ce tableau via la propriété **android:entries**

Ex : `android:entries="@array/mesChoix"`

Vous pouvez aussi fixer le prompt du spinner via sa propriété **android:prompt**.

Pour accéder en Java à la position correspondant au choix de l'utilisateur > **getSelection()** qui renvoie un entier.

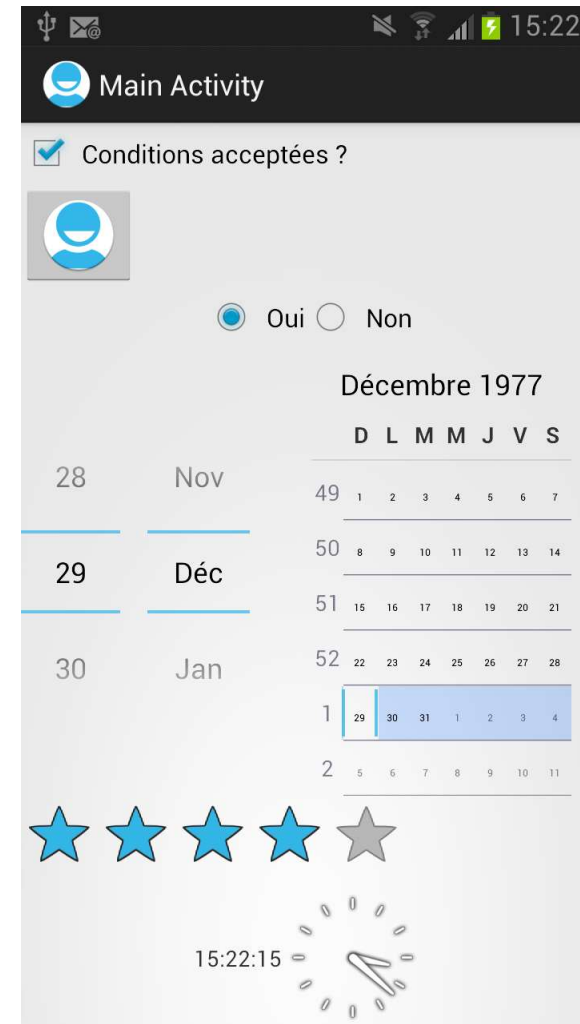
Pour positionner la liste sur un choix/item particulier > **setSelection (position-entier)**.



Attention aux nouveaux widgets !



Certains widgets ne sont pas compatibles (par ex. le TextClock dans « other »), ou ne fonctionnent plus de la même façon (par exemple le DatePicker)





Layout

- Faites des essais par vous-mêmes maintenant
 - Faites des mises en page pour jouer avec les widgets
 - Faites des mises en page que vous aurez faites sur papier avant
 - Dans tous les cas, essayez ensuite de modifier ces mises en page...



Clic sur un bouton

- onClick dans XML
- Listener dans Java



Clic sur un bouton (java) – solution 1

```
Button b1;  
Button b2;  
public void onCreate(Bundle savedInstanceState) {  
    ...  
    b1 = (Button) findViewById(R.id.b1);  
    b2 = (Button) findViewById(R.id.b2);  
    b1.setOnClickListener(myhandler1);  
    b2.setOnClickListener(myhandler2);  
    ...  
}  
  
View.OnClickListener myhandler1 = new View.OnClickListener() {  
    public void onClick(View v) {  
        // bouton 1  
    }  
};  
  
View.OnClickListener myhandler2 = new View.OnClickListener() {  
    public void onClick(View v) {  
        // bouton 2  
    }  
};
```



Clic sur un bouton (java) – solution 2

```
View.OnClickListener gestionnaireCentralise= new View.OnClickListener() {  
  
    public void onClick(View v) {  
  
        switch(v.getId()) {  
  
            case R.id.b1:  
                // bouton 1  
                break;  
  
            case R.id.b2:  
                // bouton 2  
                break;  
        }  
    }  
}
```



Clic sur un bouton (java) – solution 3 (la plus classique)

```
Button btn1 = (Button) findViewById(R.id.btn1);

btn1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // code du bouton
    }
});
```



Clic sur un bouton (java + annotations) – solution 4 (la prochaine ?)

```
@Click(R.id.myButton)
void myButtonWasClicked() {
    [...]
}
```

```
@Click({R.id.myButton, R.id.myOtherButton})
void handlesTwoButtons() {
    [...]
}
```

- Cf. <http://androidannotations.org/> et aussi <http://jakewharton.github.io/butterknife/>
Gestion du clic, mais aussi des appels REST, etc.
cf. Jetpack...



Clic sur un bouton (XML)

```
<Button  
    android:text="Button"  
    ...  
    android:id="@+id/button"  
    android:onClick="onMyClick"/>
```

```
public void onMyClick(View v) {  
    switch(v.getId()) {  
        case R.id.b1:  
            ...  
            break;  
        case R.id.b2:  
            ...  
            break;  
    }  
}
```



Clic sur un bouton (Kotlin)

// le layout **activity_main** contient un bouton avec id=monBouton

```
import kotlinx.android.synthetic.main.activity_main.*
```

```
....
```

```
monBouton.setOnClickListener { view ->
```

```
    ...  
}
```



Exemple de widgets

- Cf. projet TP_Widgets_de_base
→ regardez les clics sur ces widgets

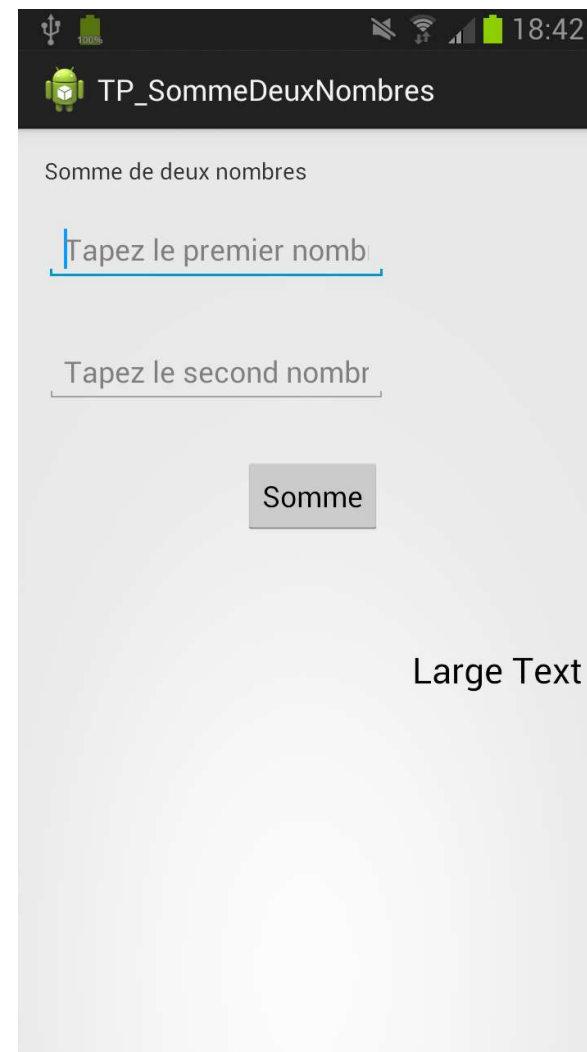
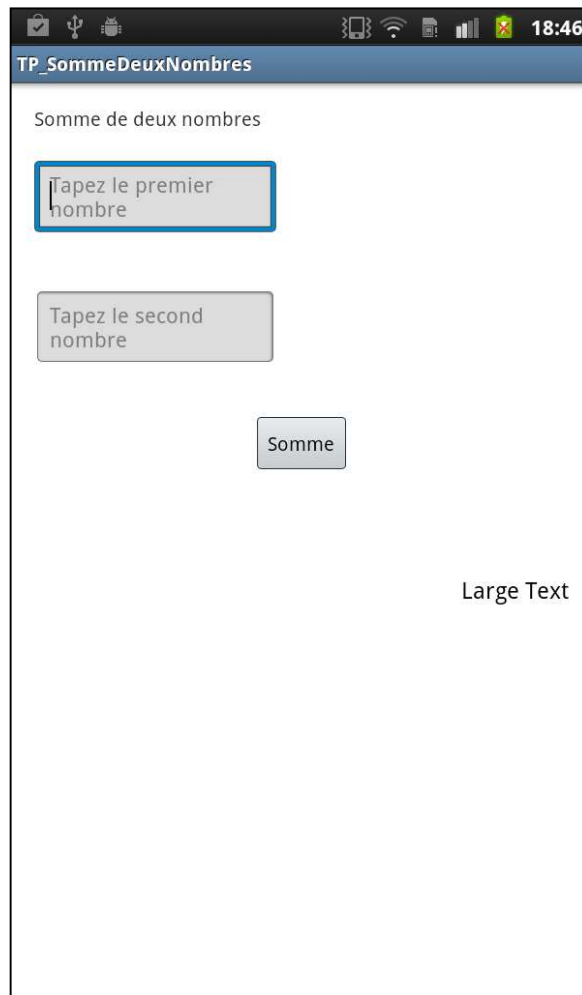
Les fichiers sources sont accessibles à

https://gitlab.com/m2eservices/exemple_widgets_de_base.git



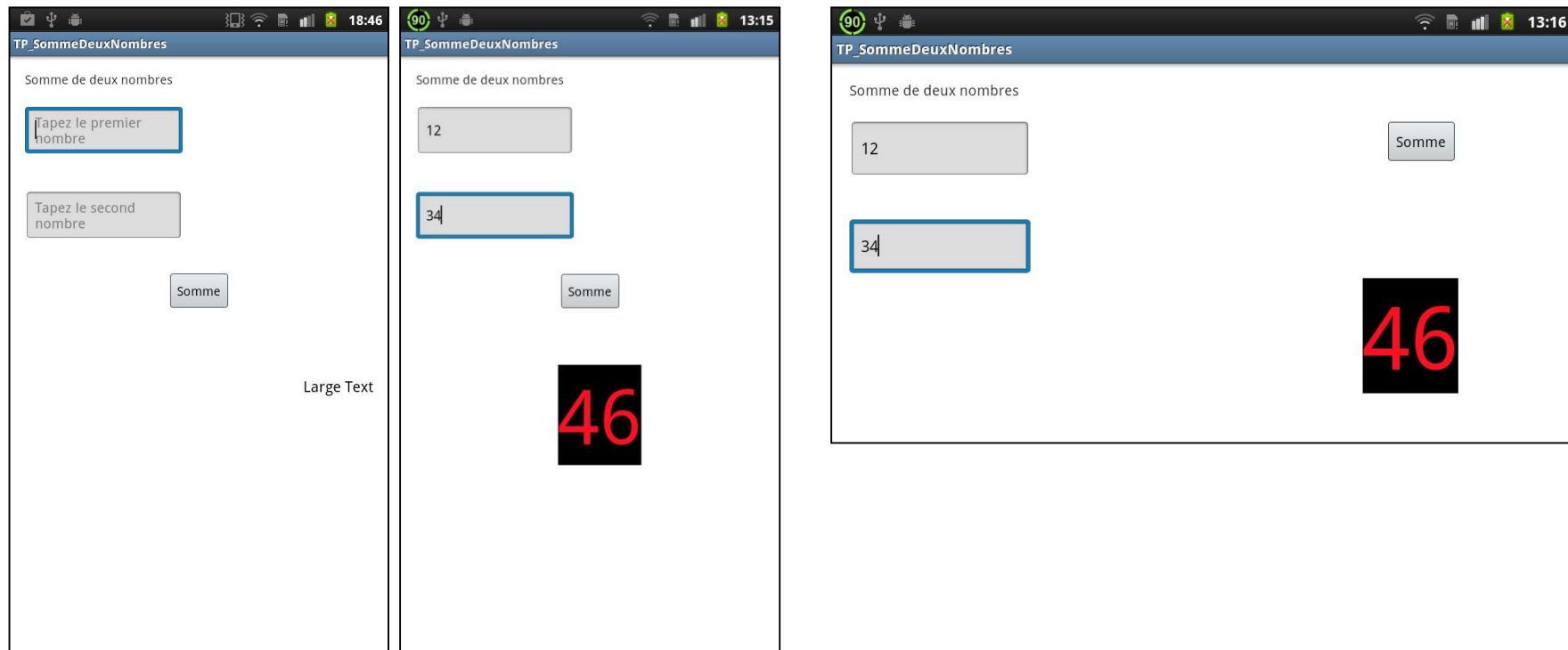


TP : somme de deux nombres





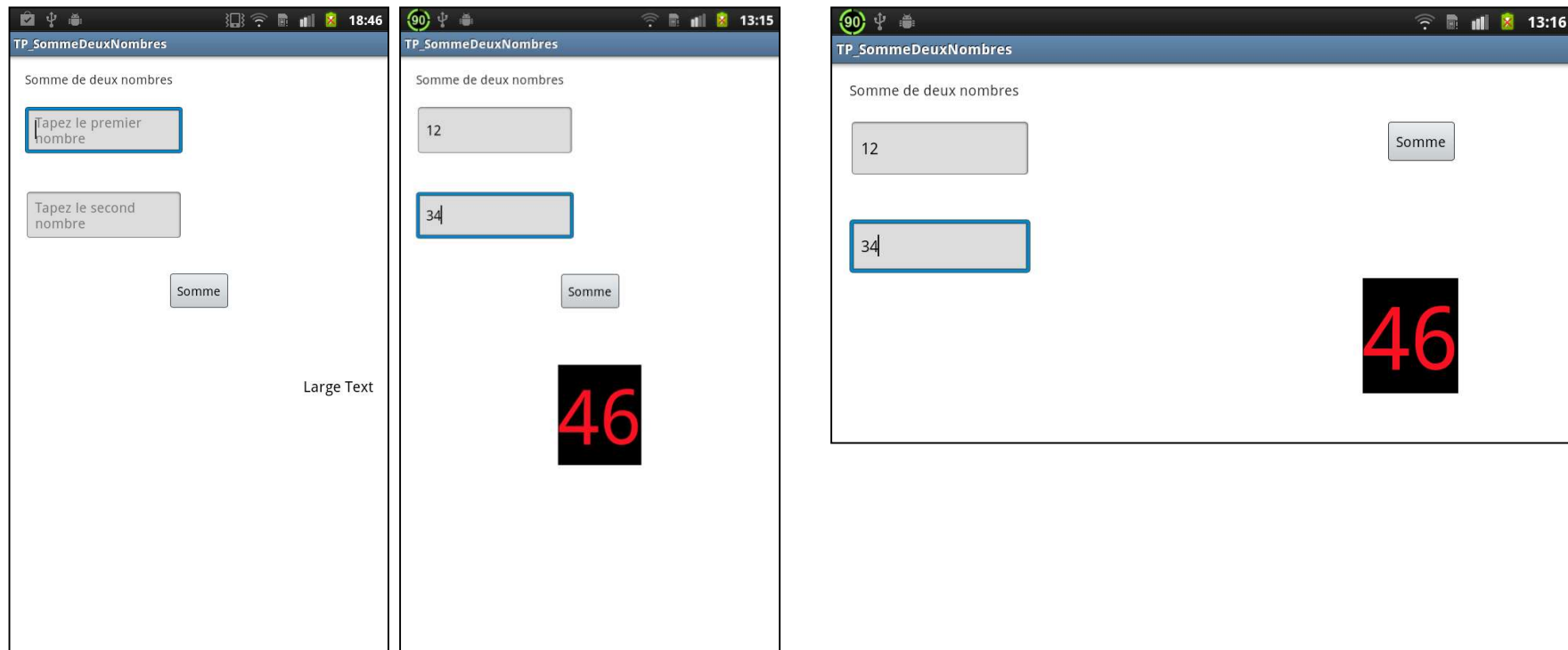
TP : somme de deux nombres



Puis utilisez les layouts
portrait/paysage, les strings, les
couleurs, les styles...



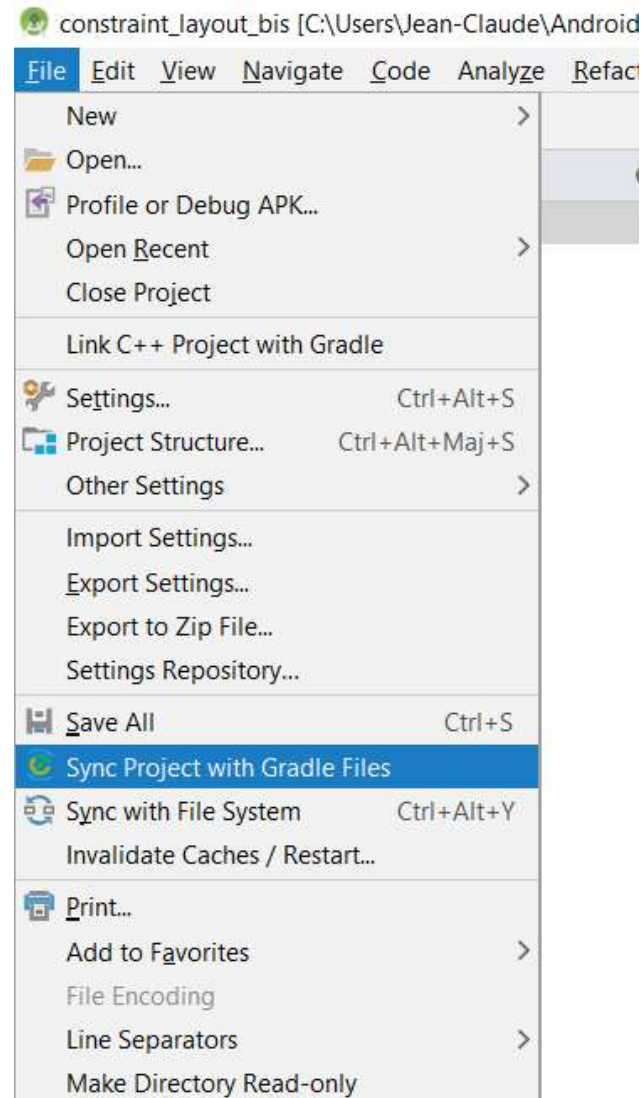
TP : somme de deux nombres



https://gitlab.com/m2eservices/tp_somme_de_deux_nombres.git



Pb de version de AS ?





Styles (layout)

<TextView

android:id="@+id/lbl_resultat"

style="@style/resultat"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:layout_alignRight="@+id/btn_Somme"

android:layout_below="@+id/txt_nb2"

android:text="Large Text"

android:textSize="@dimen/largeText" />



dimens.xml

```
<!-- Dimensions perso -->
```

```
<dimen name="largeText">80sp</dimen>
```

```
<dimen name="mediumText">40sp</dimen>
```

```
<dimen name="smallText">12sp</dimen>
```



styles.xml

```
<!-- Styles persos -->
```

```
<style name="textForTextField">
```

```
<item name="android:textSize">@dimen/largeText</item>
```

```
</style>
```

```
<style name="resultat" parent="@style/textForTextField">
```

```
<item name="android:background">#000000</item>
```

```
<item
```

```
name="android:textAppearance">?android:attr/textAppearanceLarge</item>
```

```
<item name="android:textColor">#FA1122</item>
```

```
</style>
```



Compatibilité !

- <http://developer.android.com/training/material/compatibility.html>
- **Define Alternative Styles**
 - You can configure your app to use the material theme on devices that support it and revert to an older theme on devices running earlier versions of Android:
 - Define a theme that inherits from an older theme (like Holo) in res/values/styles.xml.
 - Define a theme with the same name that inherits from the material theme in res/values-v21/styles.xml.
 - Set this theme as your app's theme in the manifest file.
 - **Note:** If your app uses the material theme but does not provide an alternative theme in this manner, your app will not run on versions of Android earlier than 5.0.
- **Provide Alternative Layouts**
 - If the layouts that you design according to the material design guidelines do not use any of the new XML attributes introduced in Android 5.0 (API level 21), they will work on previous versions of Android. Otherwise, you can provide alternative layouts. You can also provide alternative layouts to customize how your app looks on earlier versions of Android.
 - Create your layout files for Android 5.0 (API level 21) inside res/layout-v21/ and your alternative layout files for earlier versions of Android inside res/layout/. For example, res/layout/my_activity.xml is an alternative layout for res/layout-v21/my_activity.xml.
 - To avoid duplication of code, define your styles inside res/values/, modify the styles in res/values-v21/ for the new APIs, and use style inheritance, defining base styles in res/values/ and inheriting from those in res/values-v21/.
- **Use the Support Library**
 - The [v7 Support Libraries](#) r21 and above includes the following material design features:
 - [Material design styles](#) for some system widgets when you apply one of the Theme.AppCompat themes.
 - [Color palette theme attributes](#) in the Theme.AppCompat themes.
 - The [RecyclerView](#) widget to [display data collections](#).
 - The [CardView](#) widget to [create cards](#).
 - The [Palette](#) class to [extract prominent colors from images](#).



Compatibilité !

The screenshot shows a web browser window displaying the Android Developer website. The URL in the address bar is <https://developer.android.com/training/material/compatibility.html>. The page title is "Maintaining Compatibility". The left sidebar shows a navigation menu with categories like "Training", "Design", "Develop", and "Distribute". The main content area has a breadcrumb trail: "Develop > Training > Best Practices for User Interface > Creating Apps with Material Design". The article text explains that some material design features are only available on Android 5.0 (API level 21) and above, and provides instructions on how to maintain compatibility with older versions of Android. It includes a section titled "Define Alternative Styles" with three steps: 1. Define a theme that inherits from an older theme (like Holo) in `res/values/styles.xml`. 2. Define a theme with the same name that inherits from the material theme in `res/values-v21/styles.xml`. 3. Set this theme as your app's theme in the manifest file. A note states: "If your app uses the material theme but does not provide an alternative theme in this manner, your app will not run on versions of Android earlier than 5.0." There is also a section titled "Provide Alternative Layouts" which mentions that if layouts do not use new XML attributes introduced in Android 5.0, they will work on previous versions, but alternative layouts can be provided for customization on older versions.

Develop > Training > Best Practices for User Interface > Creating Apps with Material Design

Maintaining Compatibility

Some material design features like the material theme and custom activity transitions are only available on Android 5.0 (API level 21) and above. However, you can design your apps to make use of these features when running on devices that support material design and still be compatible with devices running previous releases of Android.

Define Alternative Styles

You can configure your app to use the material theme on devices that support it and revert to an older theme on devices running earlier versions of Android:

1. Define a theme that inherits from an older theme (like Holo) in `res/values/styles.xml`.
2. Define a theme with the same name that inherits from the material theme in `res/values-v21/styles.xml`.
3. Set this theme as your app's theme in the manifest file.

Note: If your app uses the material theme but does not provide an alternative theme in this manner, your app will not run on versions of Android earlier than 5.0.

Provide Alternative Layouts

If the layouts that you design according to the material design guidelines do not use any of the new XML attributes introduced in Android 5.0 (API level 21), they will work on previous versions of Android. Otherwise, you can provide alternative layouts. You can also provide alternative layouts to customize how your app looks on earlier versions of Android.



Quelques compléments sur Android Studio et Google Play...



Nouveau service Google (novembre 2015)

Toutes les applications

Services de jeux

Rapports

Paramètres

Alertes

Annonces

Barakafrit

com.andrilex.barakafrit.activities [Afficher sur le Play Store](#)

PUBLIÉE 17 mars 2015 Annuler la publication de l'application

Statistiques

Acquisition d'utilisateurs

Notes et avis

Plantages et ANR

Conseils d'optimisation 1

Cloud Test Lab

Fichiers APK

Fiche Google Play Store


Catégorie de contenu

Tarifs et disponibilité

Produits intégrés à l'application

Services et API

CLOUD TEST LAB



NOUVEAUTÉ : CLOUD TEST LAB

Le service Cloud Test Lab teste votre application sur une vaste sélection des téléphones et des tablettes Android physiques populaires.

Lorsque vous publiez des fichiers APK sur les versions alpha et bêta de Play, ils sont envoyés automatiquement à nos sites de test sur des appareils sans aucune modification requise de votre part.

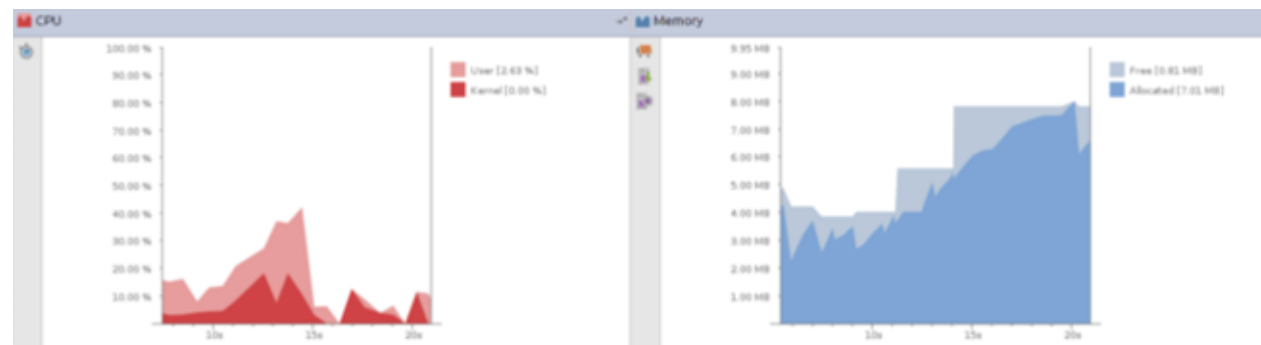
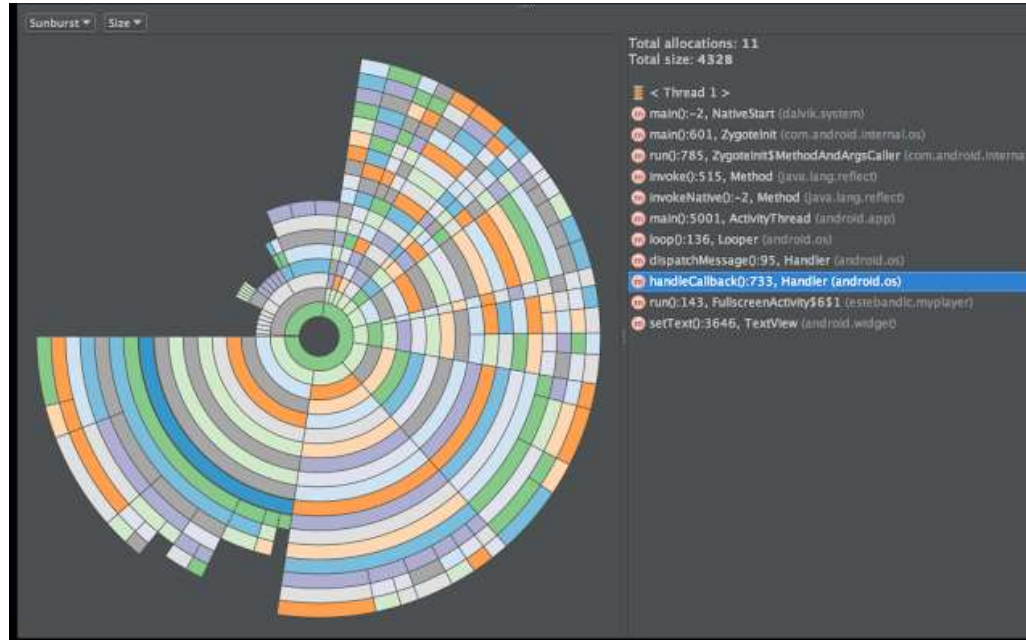
Le service Cloud Test Lab installe votre application, puis crée des interactions similaires à celles associées à un utilisateur. Il appuie sur des éléments et les fait glisser sur l'écran pour tester la fiabilité de l'application sur différents matériels. Vous pouvez consulter les rapports relatifs au test, y compris des captures d'écran et des traces de débogage complètes pour les problèmes détectés, dans la console développeur de Google Play.

Inscrivez-vous ci-dessous pour bénéficier d'un accès anticipé au service Cloud Test Lab !

S'inscrire

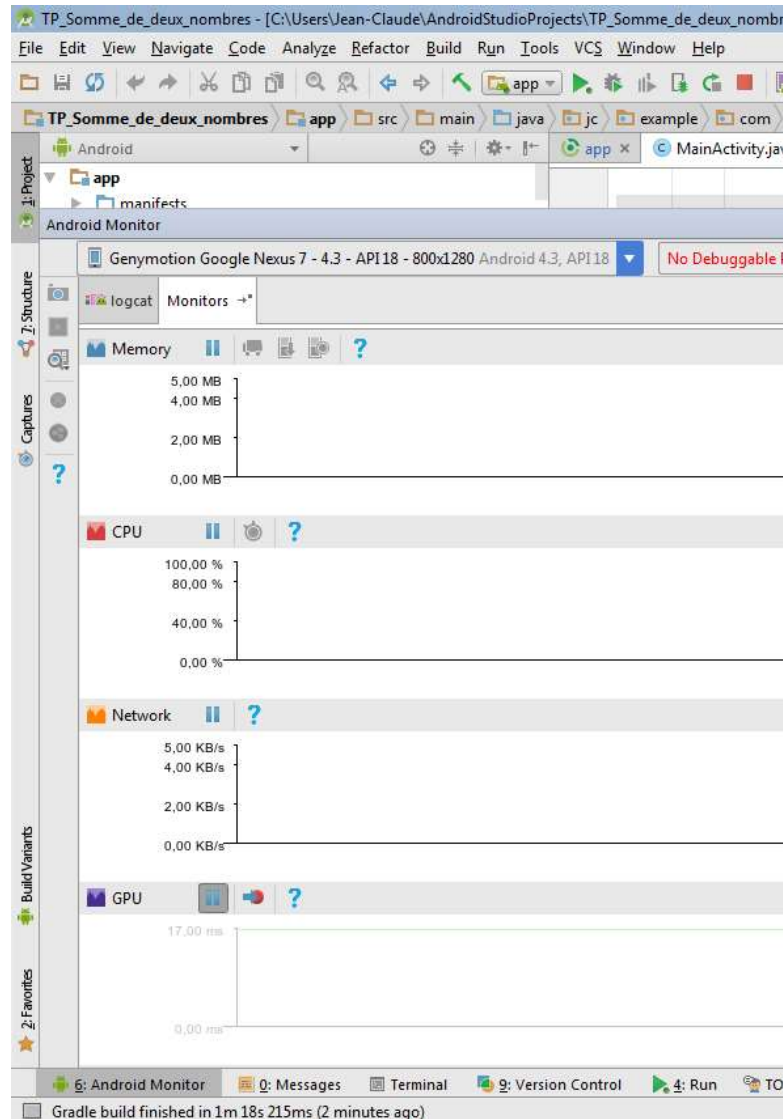


Android Studio : mémoire/CPU



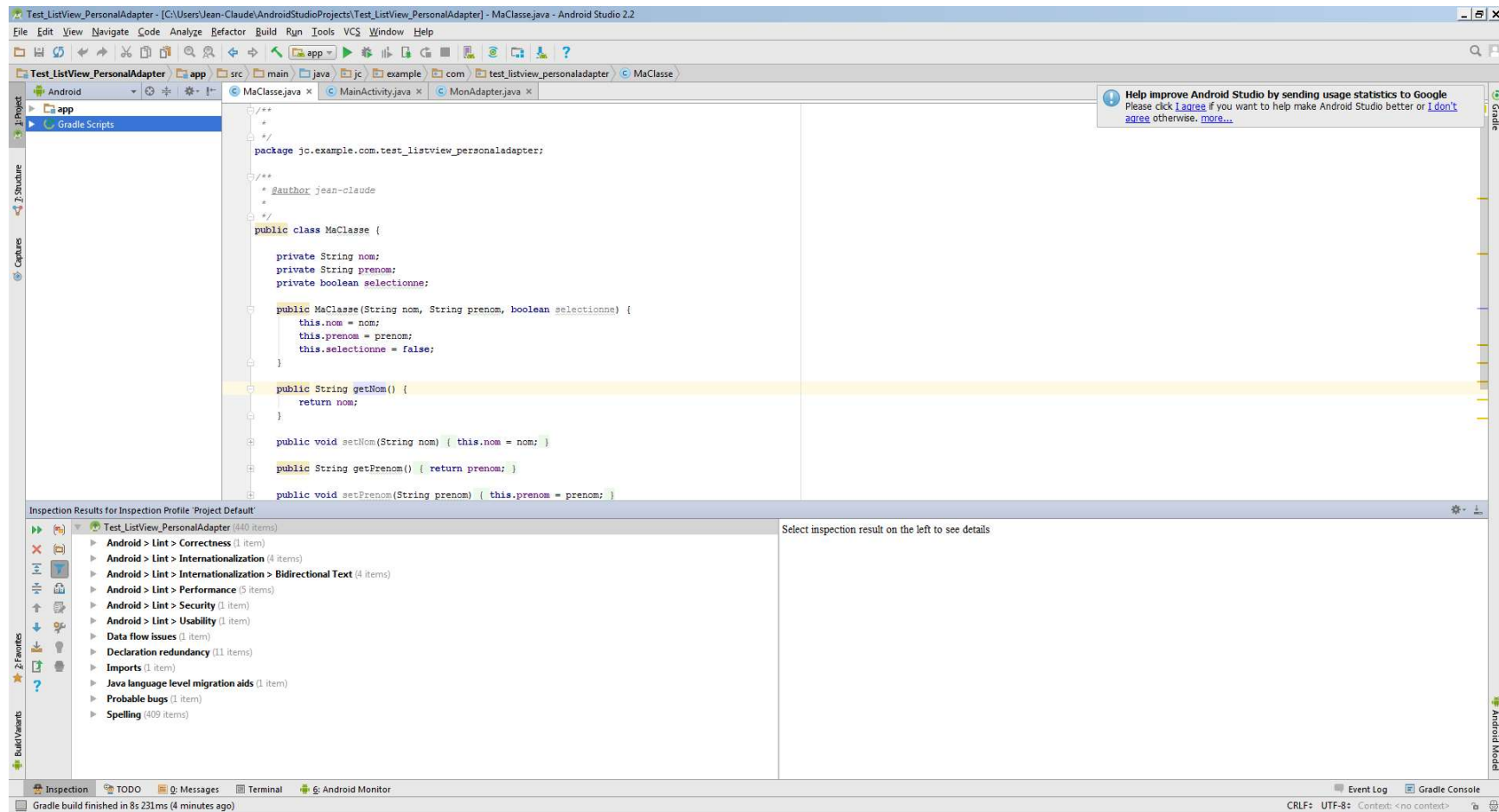


Android Studio : mémoire/CPU





Android Studio : inspection de code...





Android Studio : VCS

C:\Users\Jean-Claude\AndroidStudioProjects\MyFirstApplication\app\src\main\java\fr\univ_lille\ieea\tarby\myfirstapplication\MyActivity.java

07/09/2015 17:15 - MyActivity.java (Read-only)

package fr.univ_lille.ieea.tarby.myfirstapplication;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

public class MyActivity extends AppCompatActivity {

 @Override

 protected void onCreate(Bundle savedInstanceState) {

 super.onCreate(savedInstanceState);

 setContentView(R.layout.activity_my);

 }

 @Override

 public boolean onCreateOptionsMenu(Menu menu) {

 // Inflate the menu; this adds items to the action bar

 getMenuInflater().inflate(R.menu.menu_my, menu);

 return true;

 }

 @Override

 public boolean onOptionsItemSelected(MenuItem item) {

 // Handle action bar item clicks here. The action bar v

 // automatically handle clicks on the Home/Up button, s

 // as you specify a parent activity in AndroidManifest.

 int id = item.getItemId();

 //noinspection SimplifiableIfStatement

 if (id == R.id.action_settings) {

 return true;

 }

 }

}

Current

1 package fr.univ_lille.ieea.tarby.myfirstapplication;

2

3 import android.support.v7.app.AppCompatActivity;

4 import android.os.Bundle;

5 X import android.view.KeyEvent;

6 import android.view.Menu;

7 import android.view.MenuItem;

8

9 public class MyActivity extends AppCompatActivity {

10

11 @Override

12 protected void onCreate(Bundle savedInstanceState) {

13 super.onCreate(savedInstanceState);

14 setContentView(R.layout.activity_my);

15 }

16

17 @Override

18 public boolean onKeyLongPress(int keyCode, KeyEvent event)

19 return super.onKeyLongPress(keyCode, event);

20 }

21

22 @Override

23 public boolean onOptionsItemSelected(MenuItem item) {

24 // Handle action bar item clicks here. The action bar v

25 // automatically handle clicks on the Home/Up button, :

26 // as you specify a parent activity in AndroidManifest.

27 int id = item.getItemId();

28

29 //noinspection SimplifiableIfStatement

30 if (id == R.id.action_settings) {

31 return true;

32 }

33 }

33

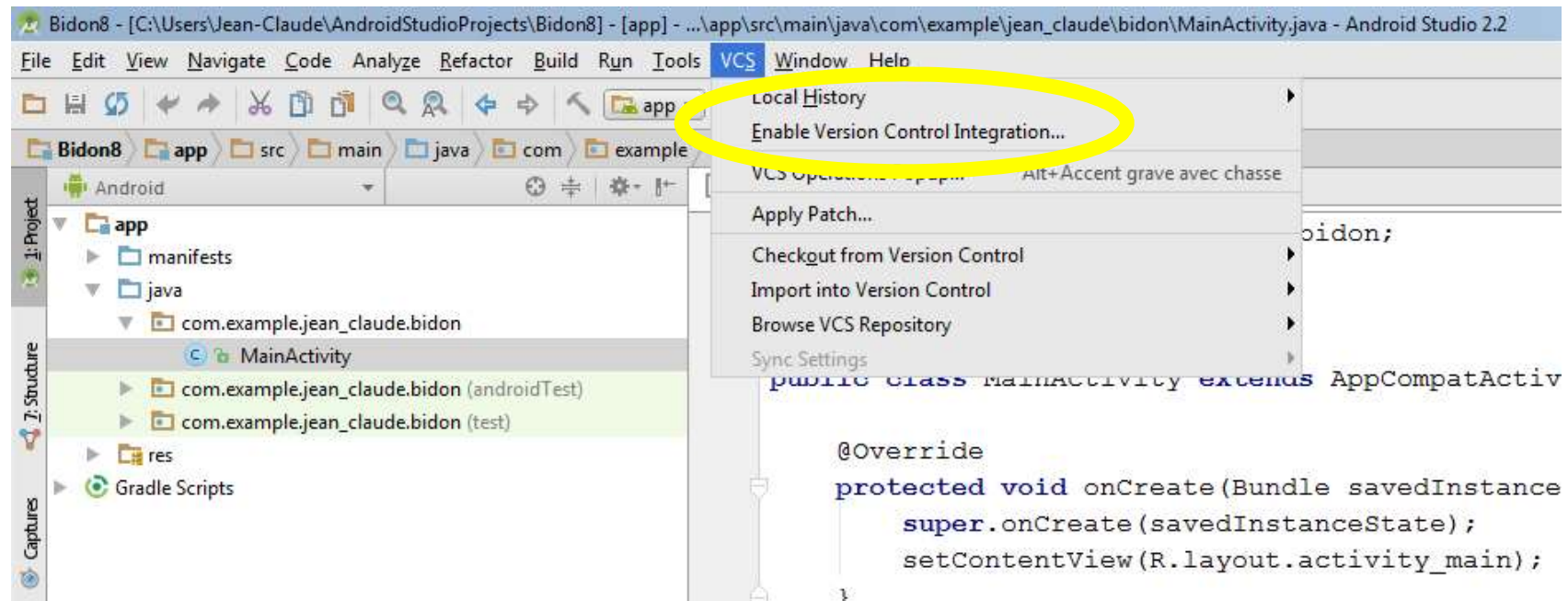
3 differences

Deleted Changed Inserted

VCS

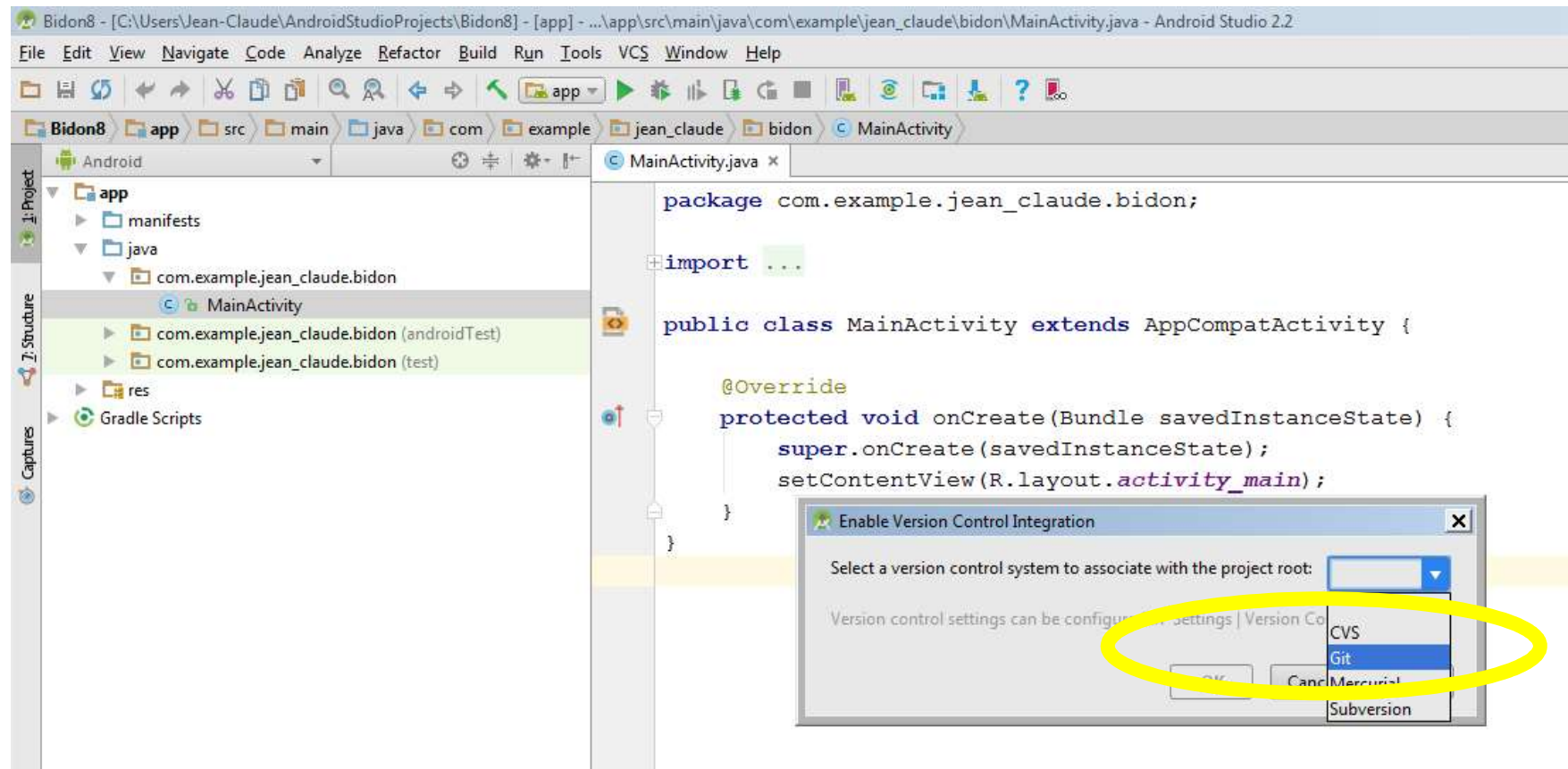


Android Studio : Git...





Android Studio : Git...



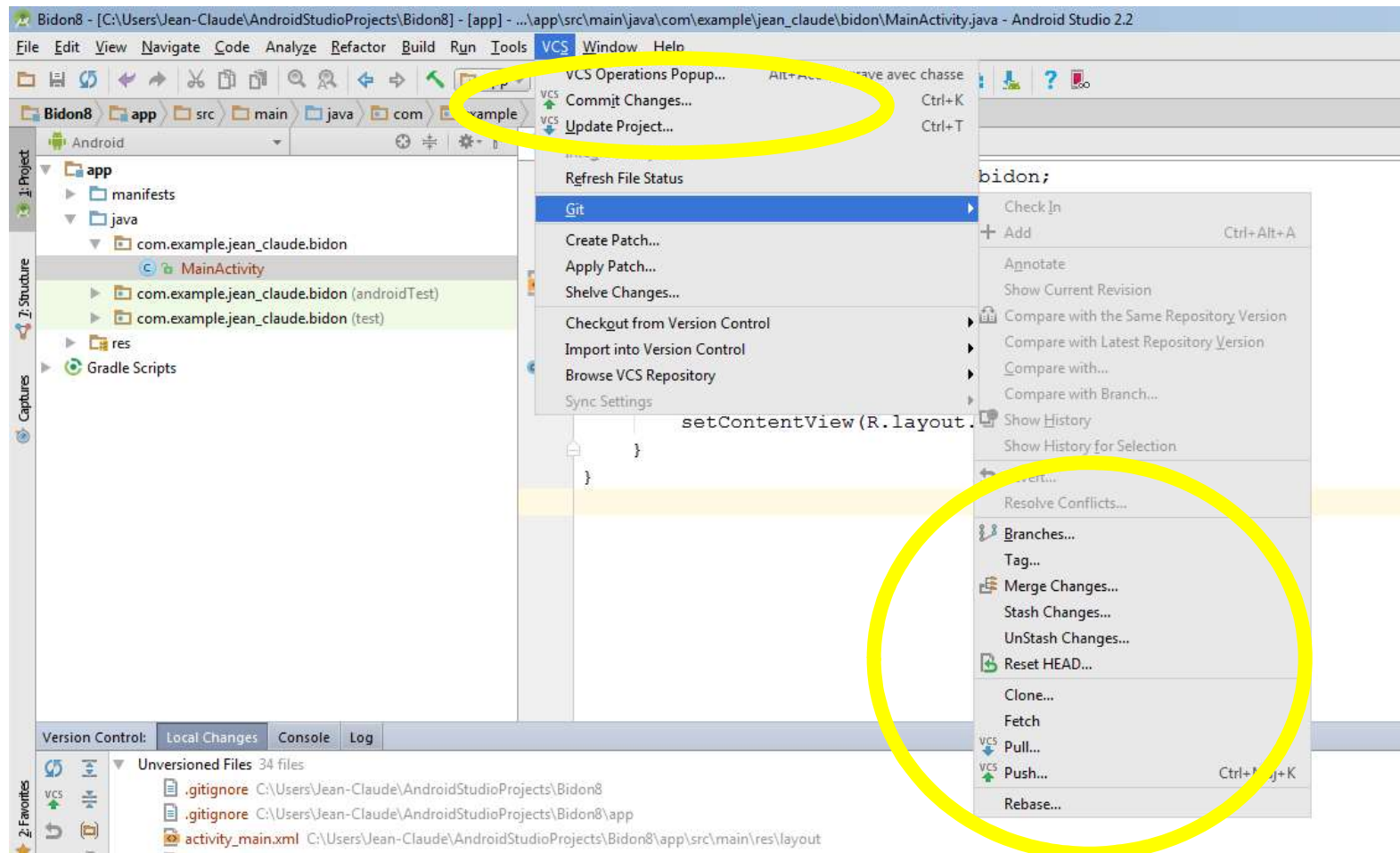


Android Studio : Git...



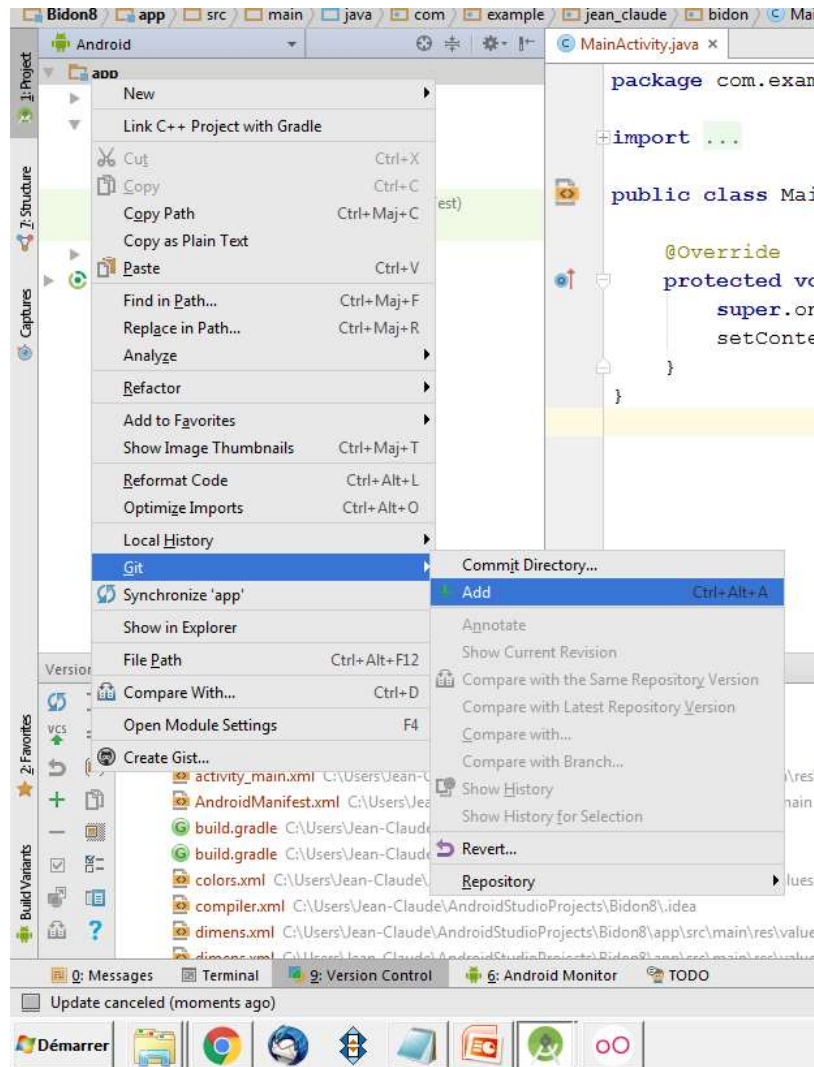


Android Studio : Git...





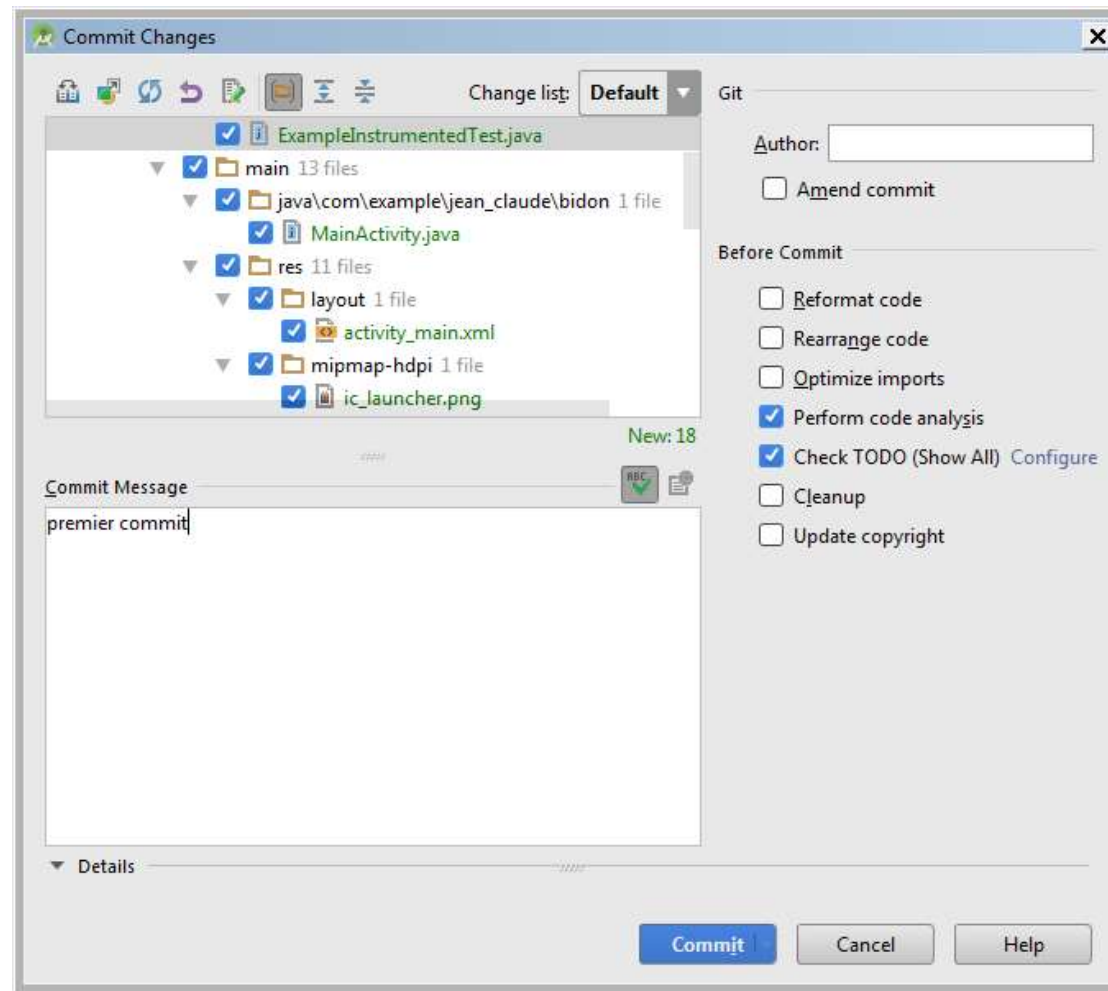
Android Studio : Git...





Android Studio : Git...

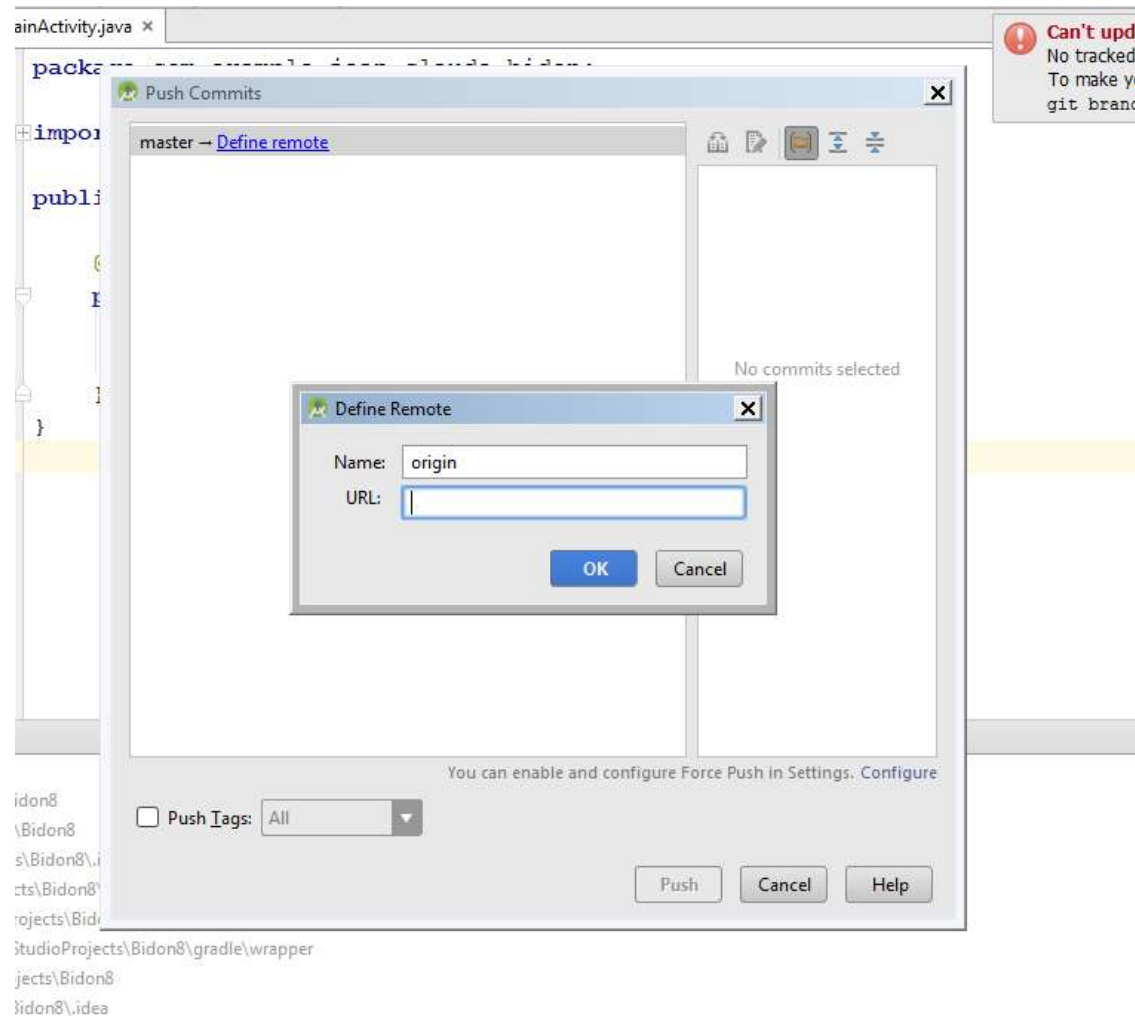
Commit...
Ctrl K





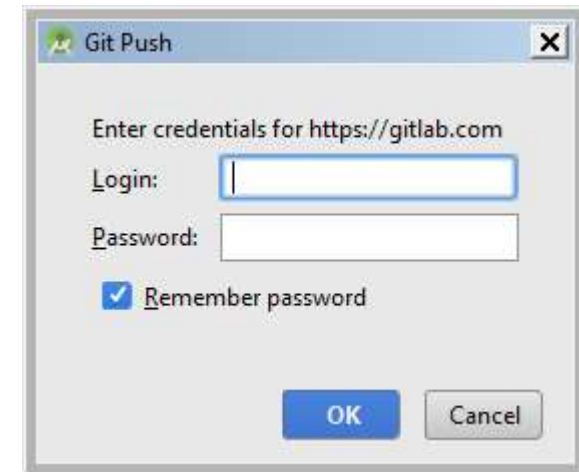
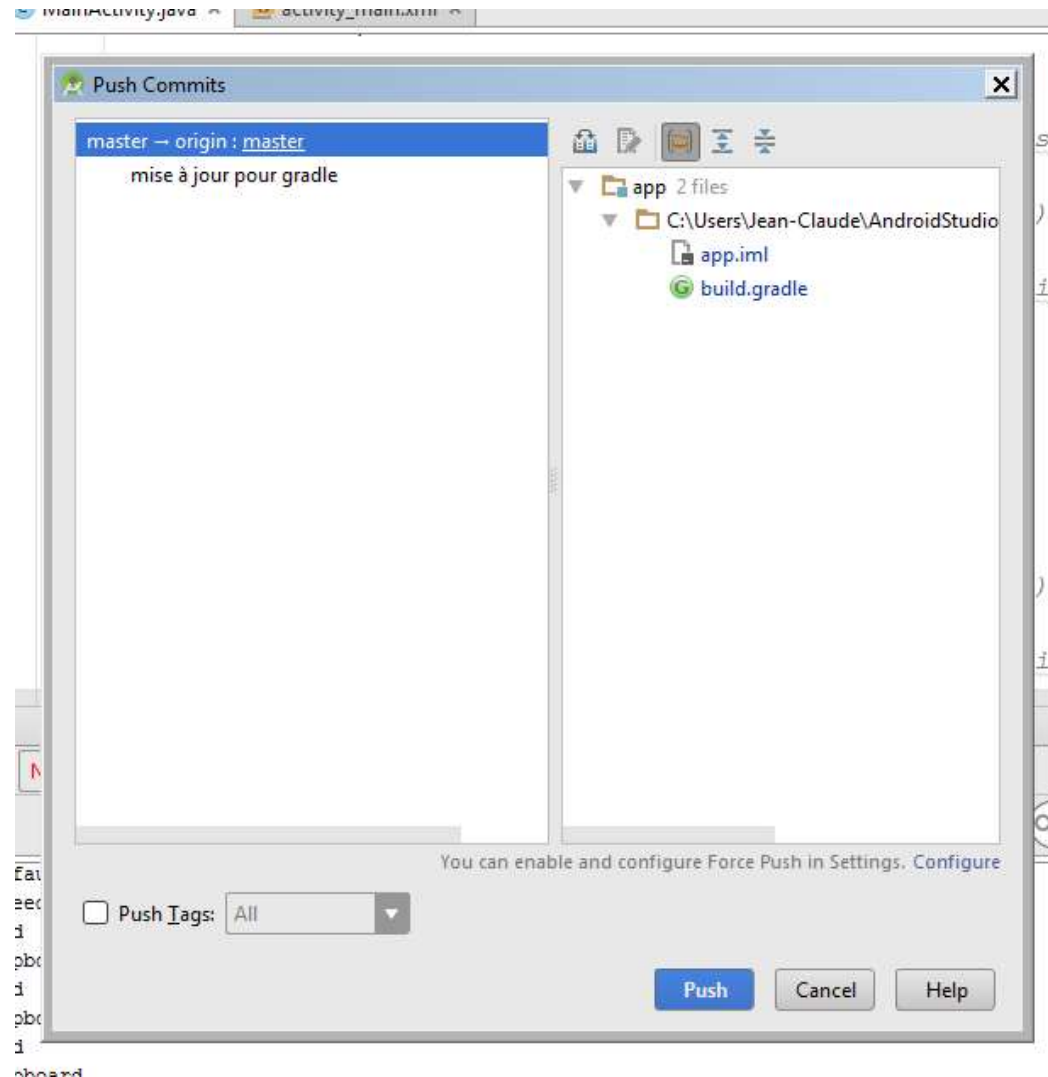
Android Studio : Git...

Push...
Ctrl Maj K





Android Studio : Git...





Android Studio : Git...

