ANDROID & ECLIPSE

Tutorial pour une application simple

1. Introduction

Android est un système d'exploitation pour téléphone portable de nouvelle génération développé par Google. Celui-ci met à disposition un kit de développement (SDK) basé sur le langage Java.
Ce tutoriel explique comment installer ce SDK et présente le développement d'une application simple.

2. Installation

2.1 Installation du SDK Android

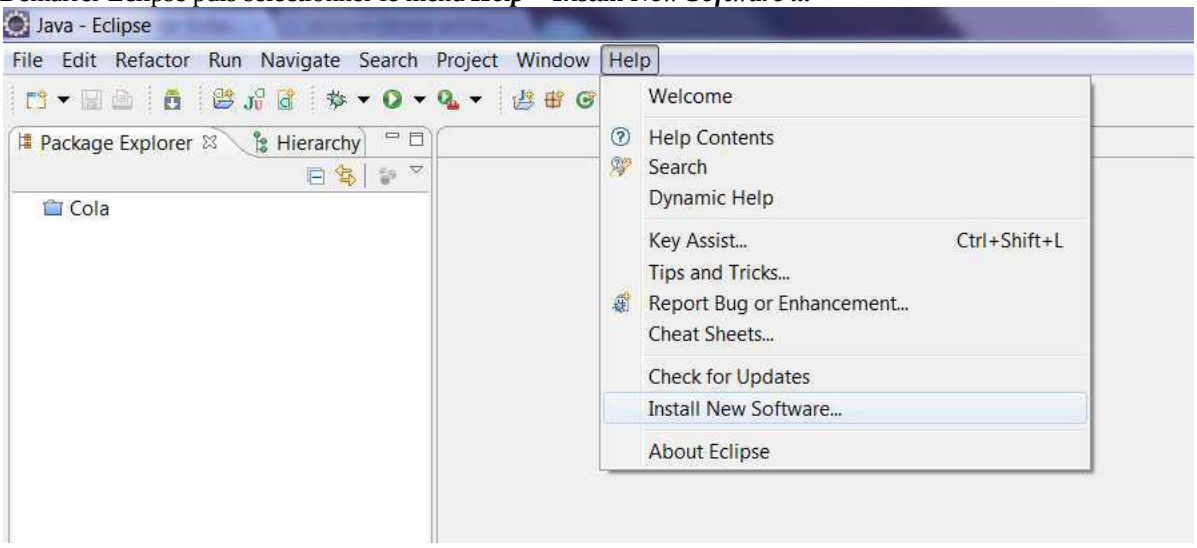
Basé sur le langage Java, le SDK Android nécessite d'avoir un JDK (Java Development Kit) et un JRE (Java Runtime Environment) installé sur sa machine pour pouvoir être utilisé ***(http://java.sun.com/javase/downloads/index.jsp)***.
Le SDK est disponible en téléchargement pour les plateformes Linux, Mac et Windows à l'adresse suivante :
http://developer.android.com/sdk/index.html
Décompresser l'archive zip ; on obtient alors un répertoire contenant le SDK nommé selon le format suivant : *android-sdk-<machine-platform>*
Noter le chemin complet vers ce répertoire (référence par la suite : dénomination *<repertoire_sdk>*).
Afin de faciliter l'utilisation des outils du SDK, il faut ajouter le sous-répertoire *tools* du répertoire *<repertoire_sdk>* dans la variable d'environnement *PATH* de votre système.

- **Linux :** éditer le fichier ~/.bash_profile ou ~/.bashrc et chercher la ligne définissant le PATH afin d'y ajouter le chemin *<repertoire_sdk>/tools* . Si cette ligne n'existe pas, ajouter celle-ci : *export PATH=${PATH}:<repertoire_sdk>/tools*
- **Mac :** éditer le fichier .bash_profile présent dans votre répertoire personnel (à créer s'il n'existe pas) puis procéder comme pour Linux
- **Windows : f**aire un clic droit sur le "Poste de travail" et sélectionner "Propriétés". Dans l'onglet "Avancé" cliquer sur le bouton "Variables d'environnement". Une boite de dialogue apparaît, double-cliquez sur l'entrée "Path" présente dans la partie "Variables Systèmes". Ajouter le chemin *<repertoire_sdk>\tools*.

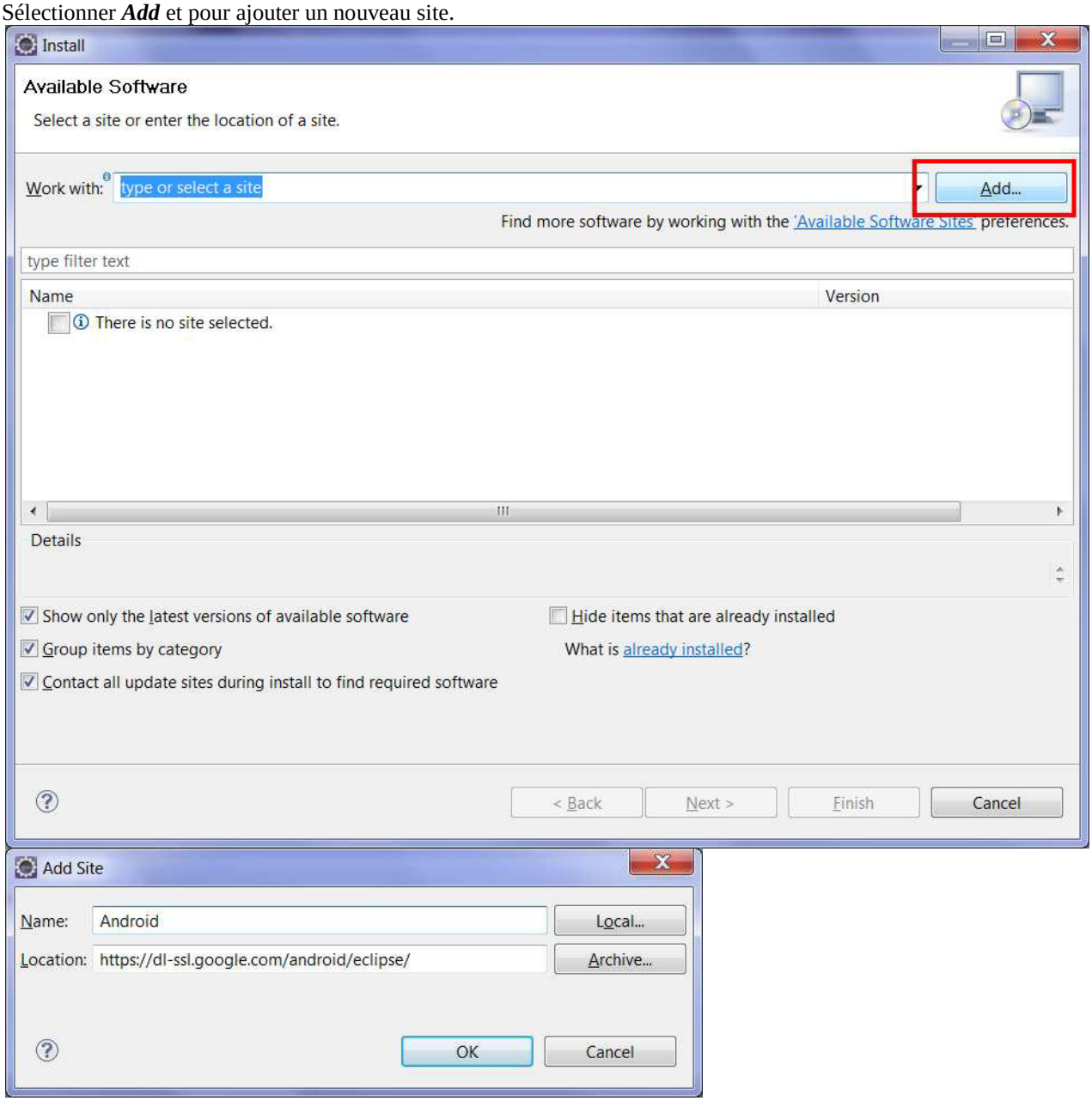
2.2 Plugin Eclipse

Google fournit un plugin pour l'IDE Eclipse (Eclipse pour Java recommandé) nommé *Android Development Tools* (ADT). Voici la marche à suivre pour installer ce plugin :

- Installer "Eclipse pour Java" le cas échéant : **http://www.eclipse.org/downloads/** (télécharger une version comprenant Java).
- Démarrer Eclipse puis sélectionner le menu ***Help > Install New Software ...***

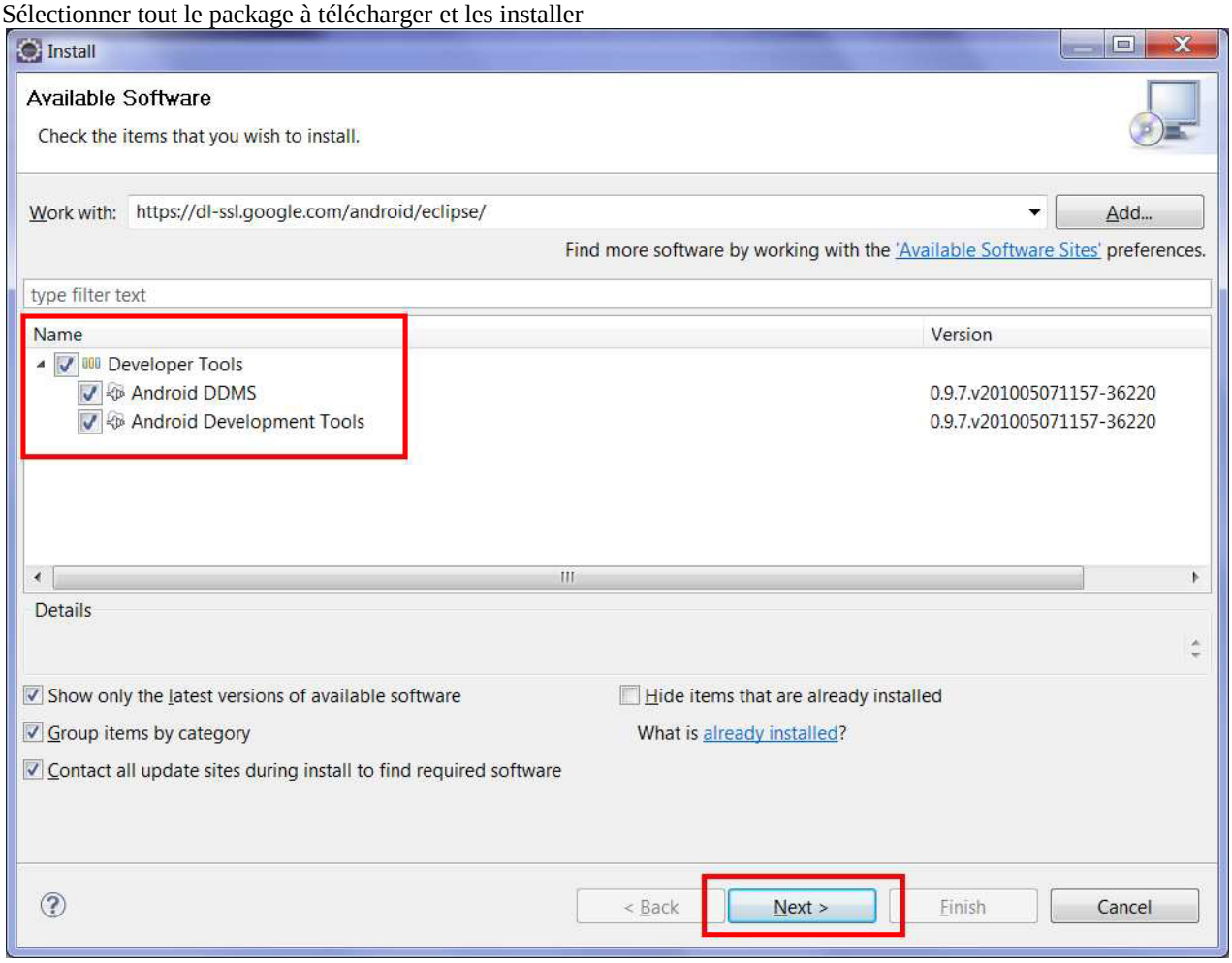


- Sélectionner ***Add*** et pour ajouter un nouveau site.

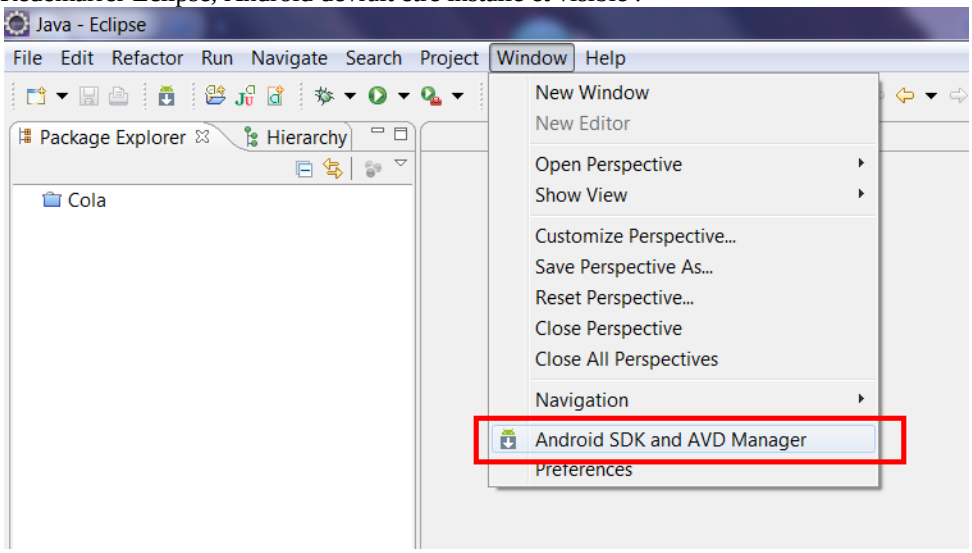


- Dans la boite de dialogue qui apparaît, indiquer un nom (par exemple *Android Plugin*) et l'URL ***https://dlssl.google.com/android/eclipse/*** . Appuyer sur le bouton ***OK***.

- Sélectionner tout le package à télécharger et les installer

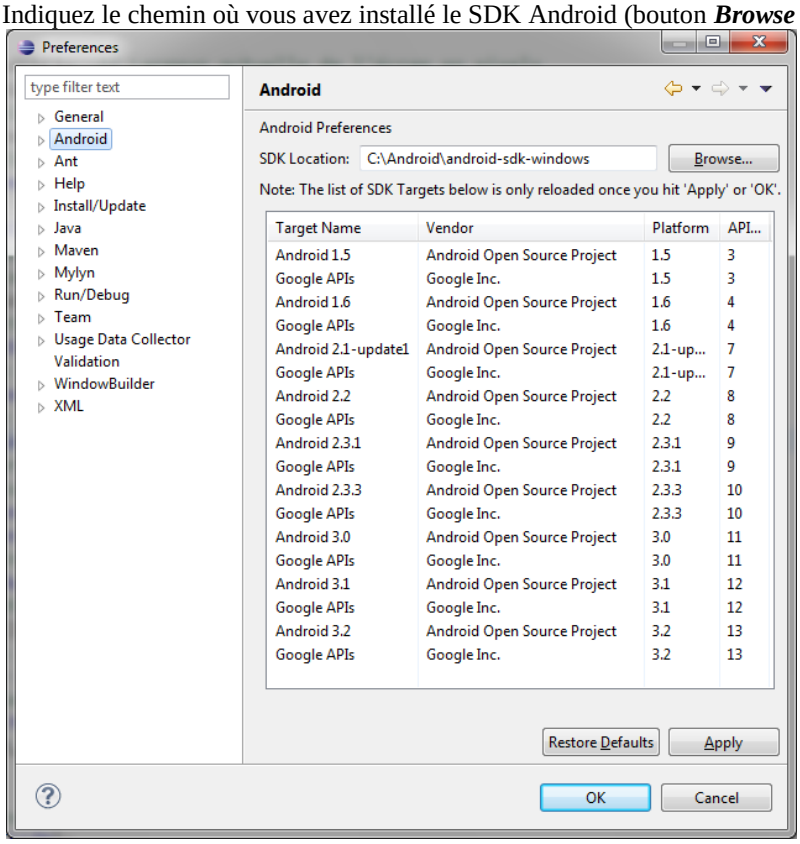


- Redémarrer Eclipse, Androïd devrait être installé et visible :



- Une fois Eclipse redémarré, sélectionner le menu ***Window*** > ***Preferences...*** (ou ***Eclipse*** > ***Preferences*** si vous êtes sous Mac OS X).
- Sélectionner ***Android*** dans le panel de gauche.

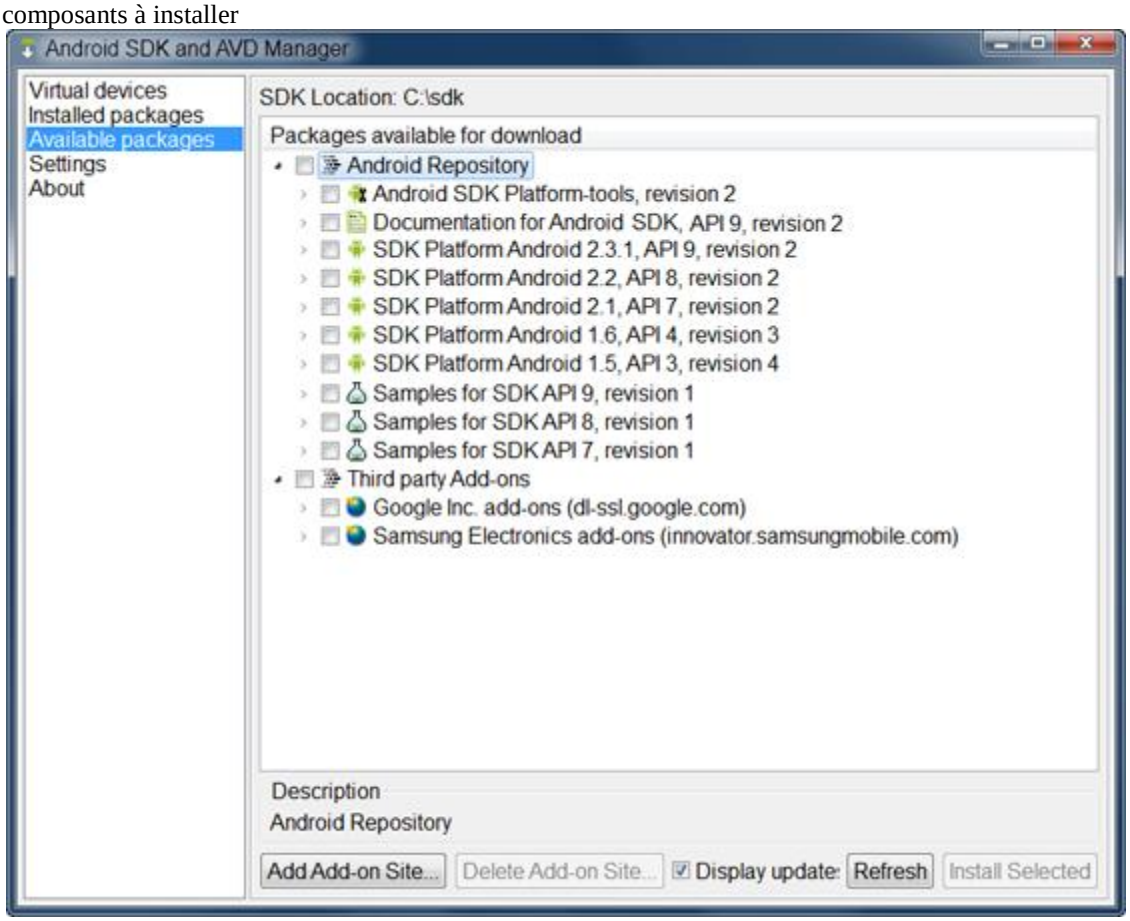
- Indiquez le chemin où vous avez installé le SDK Android (bouton ***Browse*** pour parcourir le système de fichier).



2.3 Ajouter des plateformes et d'autres composants

La dernière étape pour configurer le SDK consiste à utiliser l'outil "Android SDK and AVD Manager". Il installe les plateformes Android disponibles, des "add-ons", des exemples, de la documentation et d'autres composants.
Pour cela il faut une liaison Internet.

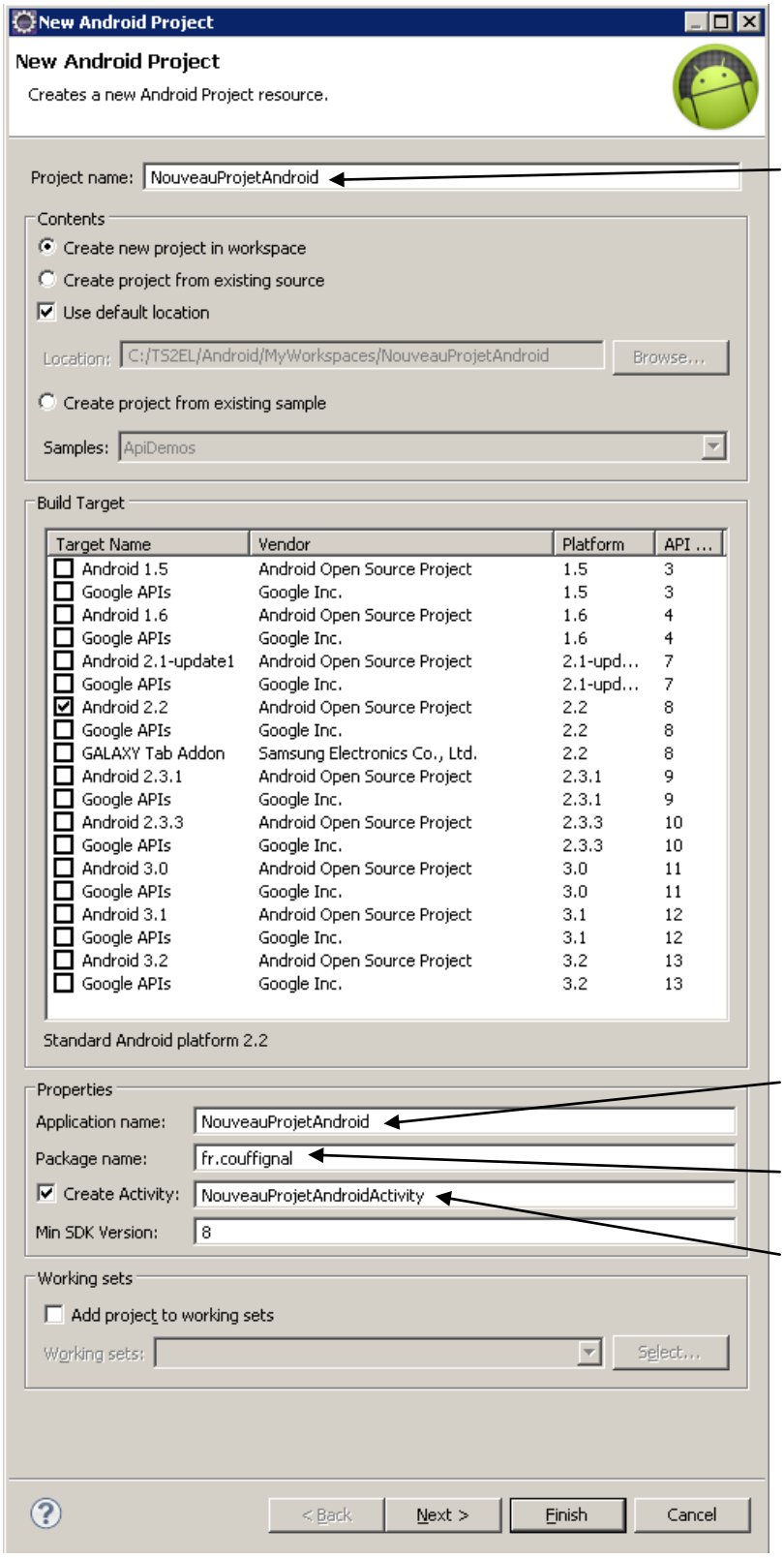
- Sélectionner le menu ***Window > Android SDK and AVD Manager***
- Pour télécharger les nouveaux composants, ouvrir le dossier "Available packages" et sélectionner (cocher) les composants à installer



Il faut au minimum choisir "Android SDK Platform-tools", la plateforme Android correspondant à la tablette utilisée et le driver USB (pour Windows) dans les "Add-ons" Google.

3. Créer une nouvelle application

- Lancer la version d'Eclipse comprenant le SDK Android
- "File/New/Android Projet" ou



Nom du projet. Un dossier à ce nom est créé dans le "workspace"

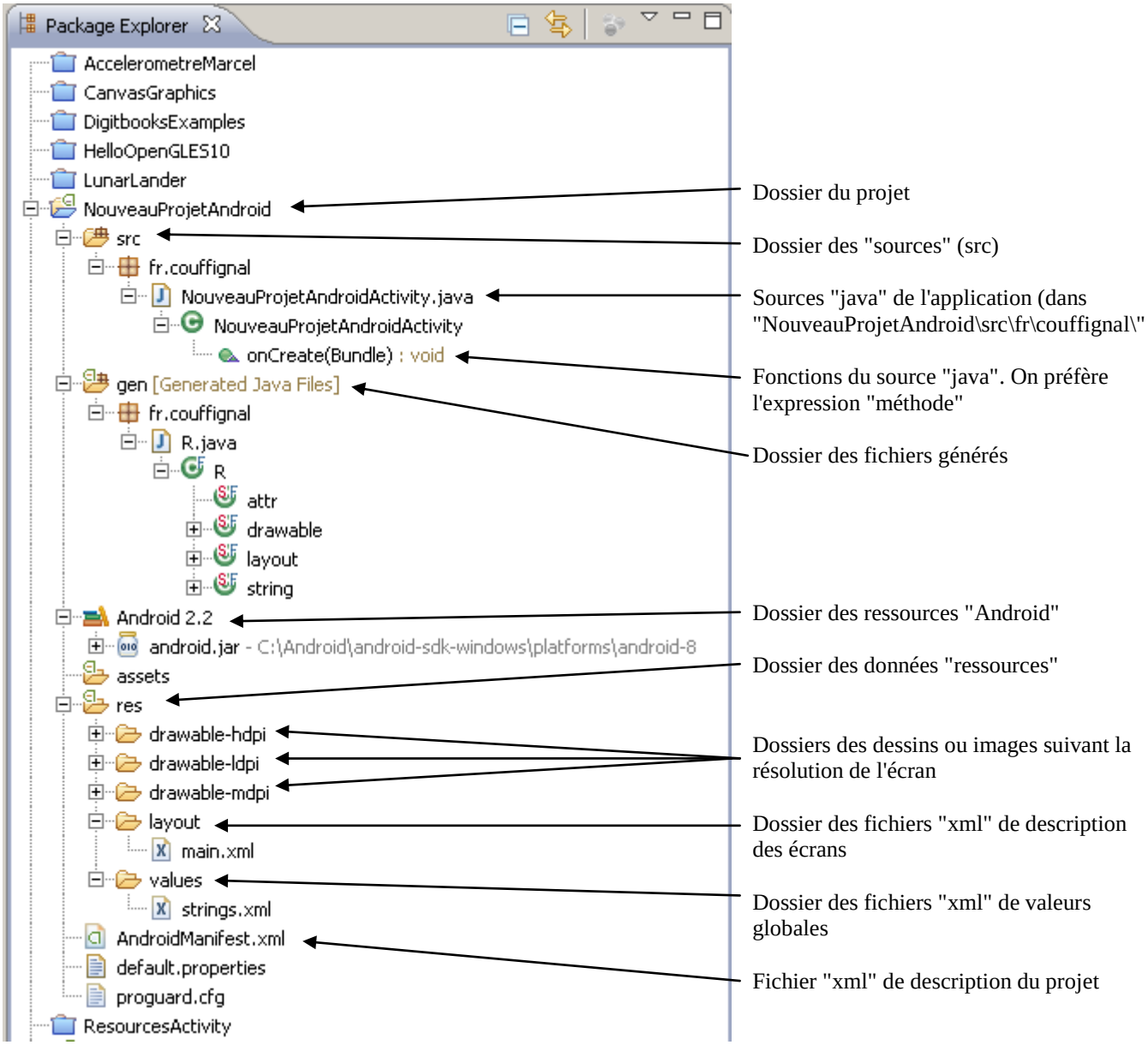
Version d'Android de la tablette utilisée.

Recopie du nom du projet. Peut être modifié

"fr" et "couffignal" sont 2 dossiers imbriqués dans le dossier du projet dans lesquels seront placés les fichiers du projet.

Nom de la première activité de l'application

Le projet est créé avec "Finish" et on obtient la structure suivante :



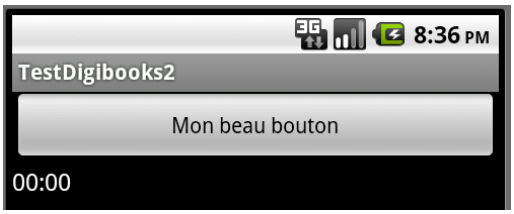
4. Une application simple

L'application décrite est sans intérêt pratique ! L'objectif est de mettre en œuvre rapidement les principes fondamentaux de développement d'une application Android :

- Création des écrans, avec leurs "widgets" (boutons, zones de textes, etc.)
- Création des objets "activité" de l'application (au moins une) comportant les fonctions (méthodes) "évènement"
- Création des objets "ressources" (méthodes des tâches de fond)
- Gestion des communications entre les objets.

4.1 Description de l'application

Ecran de démarrage



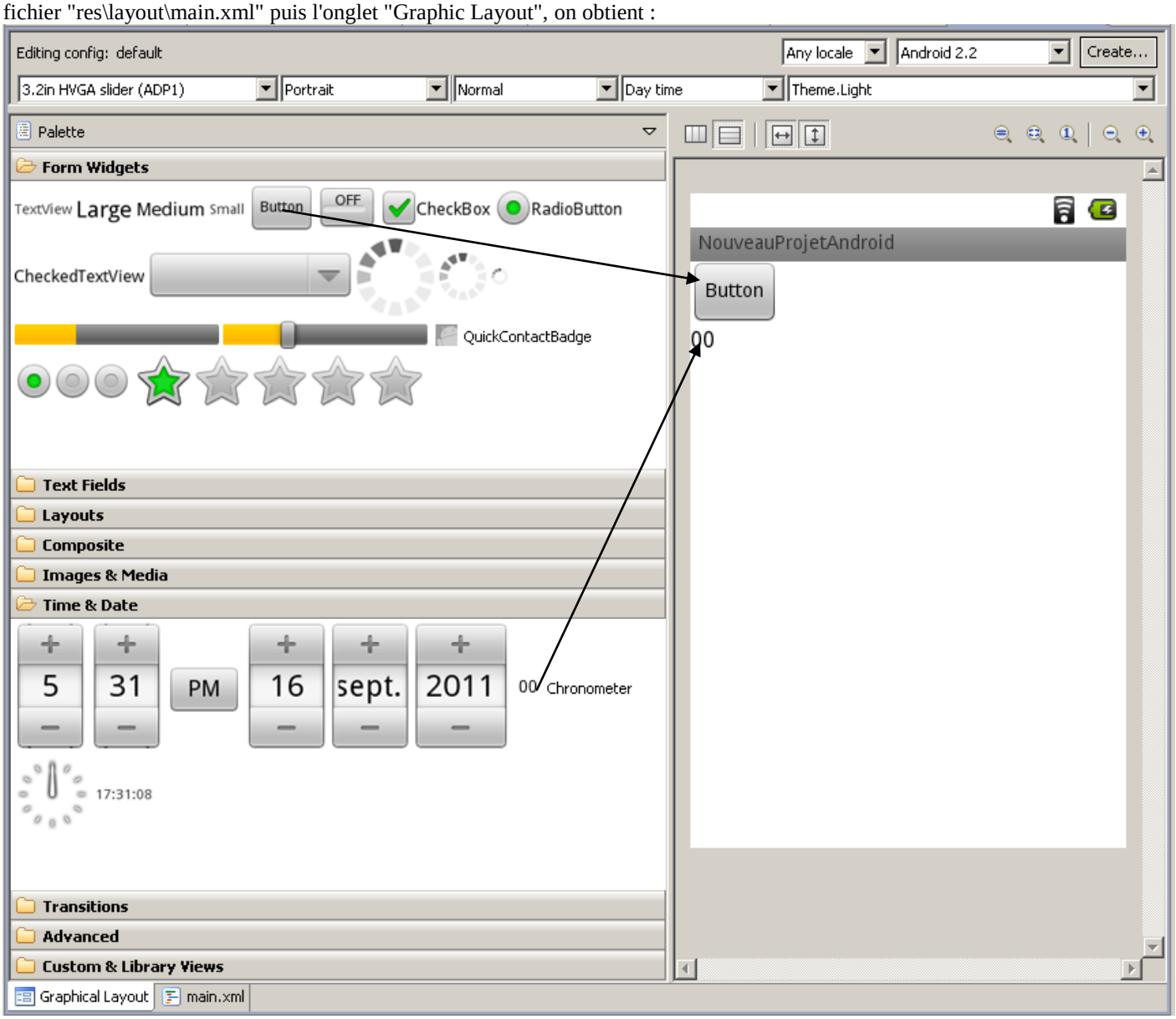
Ecran d'édition

Fonctionnement attendu :

- Démarrage de l'application : "activité 1" : afficher l'écran de démarrage comportant un bouton "Mon beau bouton" et un chronomètre numérique
- Action sur le bouton :
  - Démarrage de la méthode "onStart" de la classe "MonService" :
    - Affichage de la notification "Le service est démarré"
    - Déclenchement d'un délai de 5s
    - Après ce délai de 5s : affichage de la notification "Le service est arrêté"
  - Préparation d'un dialogue entre les activités 1 et 2
  - Démarrage de l'activité 2 : afficher l'écran de dialogue comportant :
    - Un message "Coucou ! Ma deuxième activité
    - Un message "Quel est votre nom ?"
    - Une fenêtre d'édition du nom
    - Un bouton "Valider"
- Action sur le bouton "Valider" :
  - fermer le 2° écran (et la 2° activité) et transmission de la valeur éditée vers la 1° activité
  - afficher temporairement la valeur éditée sur l'écran de démarrage

4.2 Création du premier écran

Dans une application Android, la composition des écrans est réalisée dans des fichiers "xml" séparés du code java. Cette méthode permet d'adapter rapidement l'interface utilisateur suivant la résolution et les dimensions de l'écran de la cible.
Toutefois, le plugin Android permet de construire l'essentiel de son interface utilisation de manière graphique. En ouvrant le fichier "res\layout\main.xml" puis l'onglet "Graphic Layout", on obtient :



Pour construire l'écran, il suffit de "glisser" et "déposer" le widget choisi. Le fichier "main.xml" est automatiquement mis à jour.
Toutefois, pour fignoler la présentation, il est nécessaire d'ajouter "à la main" quelques attributs complémentaires.
Le langage "xml" est analogue au langage "HTML" : comme lui, il utilise des "tags"

Contenu initial du fichier "main.xml" :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button android:text="Button"
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
    </Button>
    <Chronometer android:text="Chronometer"
                 android:id="@+id/chronometer1"
                 android:layout_width="wrap_content"
                 android:layout_height="wrap_content">
    </Chronometer>
</LinearLayout>
```

Tag "LinearLayout" : défini un écran dont les éléments sont placés les uns sous les autres

Tag du bouton

Texte dans le bouton

Identificateur du bouton : "button1"

Dimensions adaptées au contenu

Tag du chronomètre

Contenu modifié du fichier "main.xml" :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button android:text="Mon beau bouton"
            android:id="@+id/monBoutonId"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">
    </Button>
    <Chronometer android:text="Chronometer"
                 android:id="@+id/monChronoId"
                 android:layout_height="wrap_content"
                 android:layout_width="fill_parent"
                 android:gravity="center_horizontal">
    </Chronometer>
</LinearLayout>
```

Texte du bouton modifié

Identificateur modifié

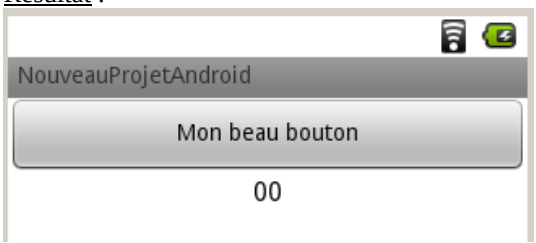
Largeur = largeur de l'écran

Identificateur modifié

Largeur = largeur de l'écran

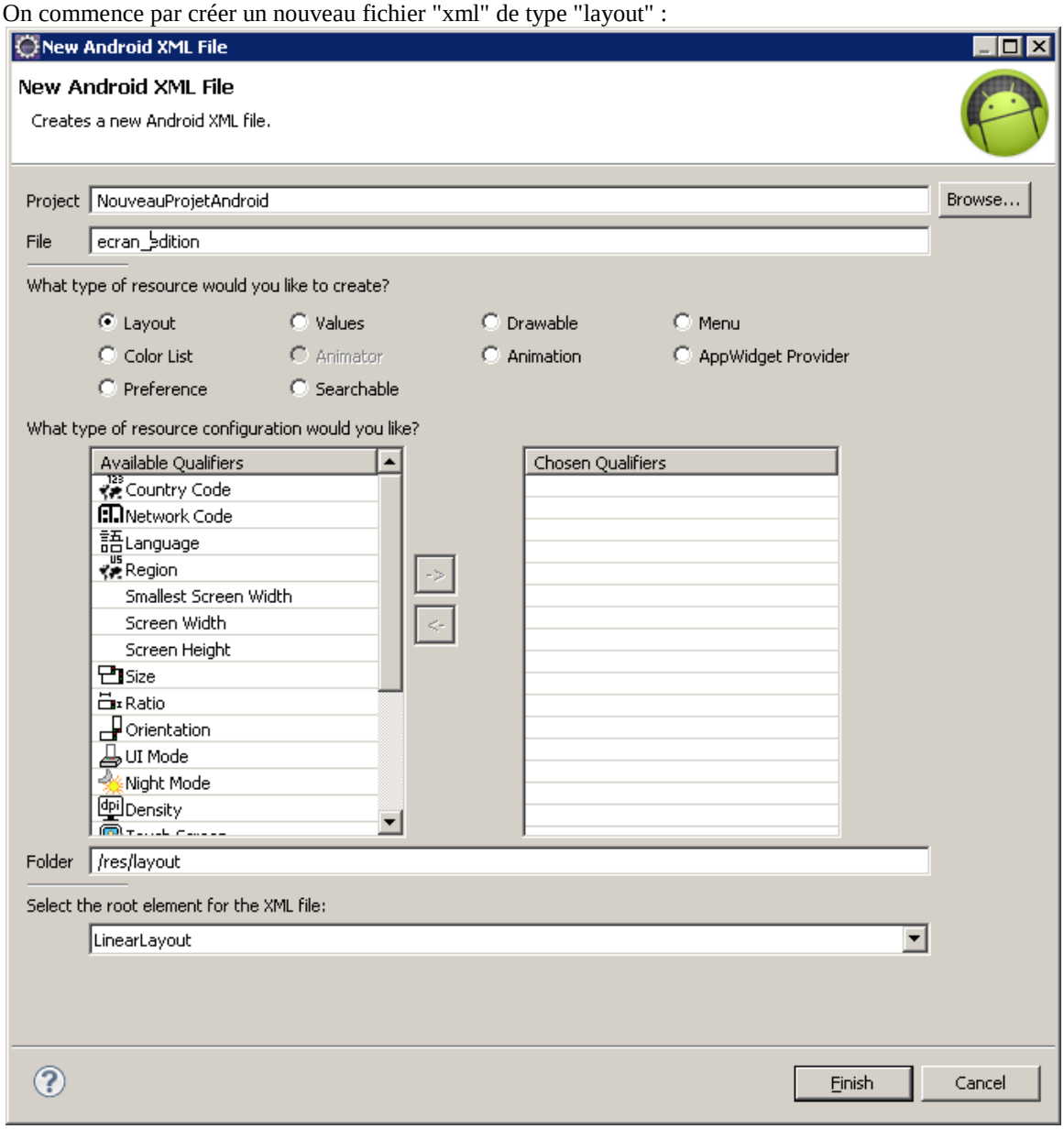
Centré

Résultat :



4.3 Création du deuxième écran

On commence par créer un nouveau fichier "xml" de type "layout" :



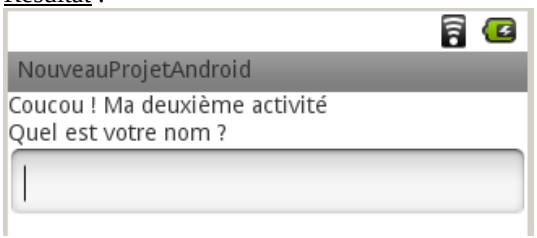
Contenu initial du fichier "ecran_edition.xml" :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <TextView android:layout_height="wrap_content"
              android:id="@+id/textView1"
              android:text="TextView"
              android:layout_width="match_parent">
    </TextView>
    <TextView android:layout_height="wrap_content"
              android:id="@+id/textView2"
              android:text="TextView"
              android:layout_width="match_parent">
    </TextView>
    <EditText android:layout_height="wrap_content"
              android:id="@+id/editText1"
              android:layout_width="match_parent">
        <requestFocus></requestFocus>
    </EditText>
</LinearLayout>
```

Contenu modifié du fichier "ecran_edition.xml" :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <TextView android:layout_height="wrap_content"
              android:id="@+id/textView1"
              android:text="Coucou ! Ma deuxième activité"
              android:layout_width="match_parent">
    </TextView>
    <TextView android:layout_height="wrap_content"
              android:id="@+id/textView2"
              android:text="Quel est votre nom ?"
              android:layout_width="match_parent">
    </TextView>
    <EditText android:layout_height="wrap_content"
              android:id="@+id/editTextNomId"
              android:layout_width="match_parent">
        <requestFocus></requestFocus>
    </EditText>
</LinearLayout>
```

Résultat :



4.4 Code "java" de l'application

Les 2 écrans de l'application sont créés, mais l'application ne fait rien ! On peut le constater en lançant l'application (sur l'émulateur par exemple) avec la commande "Run as Android project".
L'activité 1 de l'application est créé automatiquement à la création du projet. La classe correspondante est nommée "NouveauProjetAndroid**Activity**", "NouveauProjetAndroid" étant le nom du projet.

Contenu initial du fichier "NouveauProjetAndroidActivity.java" :

```
package fr.couffignal;

import android.app.Activity;
import android.os.Bundle;

public class NouveauProjetAndroidActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Imports nécessaires à l'application

Classe "NouveauProjetAndroidActivity" héritée du type "Activity"

Cette directive indique que le constructeur "onCreate" de la classe mère est "overridé" par la méthode de même nom qui suit.

La méthode "onCreate" est exécutée au lancement de l'application

Appel du constructeur de la classe mère

Affichage de l'écran décrit par "main.xml".

Le paramètre de type "Bundle" de la méthode "onCreate" permet de sauvegarder dans "savedInstanceState" l'état de l'application à chaque changement d'écran ou au retour sur le bureau.

4.4.1 Utilisation des "widgets" dans le code source.

Les widgets sont créés avec les fichiers "xml" dans le dossier "layout". Pour les utiliser dans le code java, il faut y créer des instances de ces widgets.

```
package fr.couffignal;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.Chronometer;

public class NouveauProjetAndroidActivity extends Activity
{
 // Déclaration des instances des widgets de l'écran "main"
 Button monBouton;
 Chronometer monChrono;

 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState)
 {
  super.onCreate(savedInstanceState);
  // Affichage de l'écran de démarrage
  setContentView(R.layout.main);
  // Création des instances du bouton et du chronomètre de l'écran "main"
  monBouton=(Button)findViewById(R.id.monBoutonId);
  monChrono=(Chronometer)findViewById(R.id.monChronoId);
 }
}
```

Imports nécessaires

A ajouter

Les instances des widgets doivent être créées après l'affichage de l'écran

Note : la fonction "***findViewById***" permet d'affecter l'instance d'un objet à partir de son identificateur défini dans le fichier "main.xml".

4.4.2 Détecter un clic sur le bouton de l'écran "main"

Pour détecter une action sur le bouton, il faut créer un écouteur : un "**Listener**". Celui-ci "écoute" alors le bouton dès que la fenêtre est visible pour lancer la méthode associée.

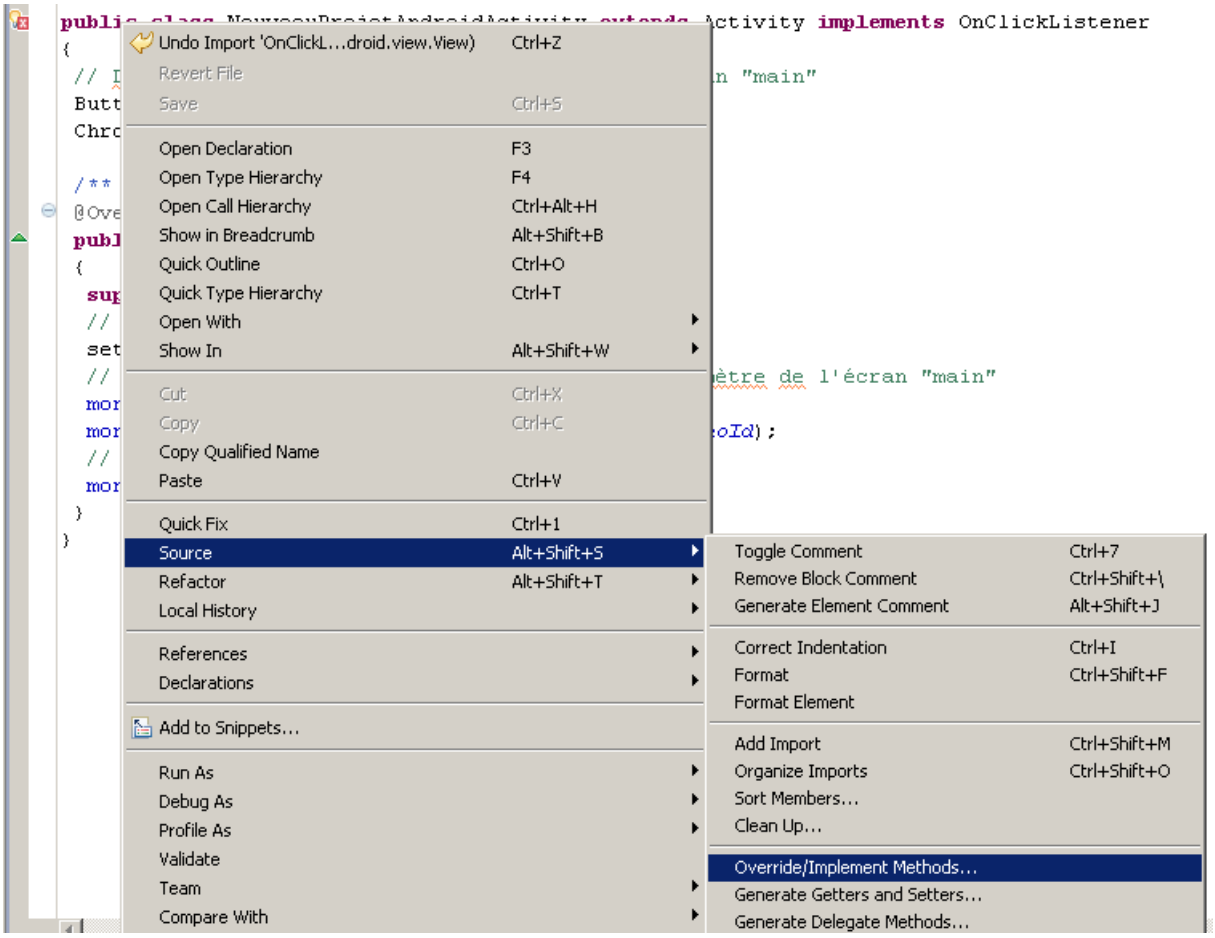
```
package fr.couffignal;

import android.app.Activity;
import android.os.Bundle;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Chronometer;

public class NouveauProjetAndroidActivity extends Activity implements OnClickListener
{
 // Déclaration des instances des widgets de l'écran "main"
 Button monBouton;
 Chronometer monChrono;

 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState)
 {
  super.onCreate(savedInstanceState);
  // Affichage de l'écran de démarrage
  setContentView(R.layout.main);
  // Création des instances du bouton et du chronomètre de l'écran "main"
  monBouton=(Button)findViewById(R.id.monBoutonId);
  monChrono=(Chronometer)findViewById(R.id.monChronoId);
  // "Ecouteur" du bouton
  monBouton.setOnClickListener(this);
 }
}
```

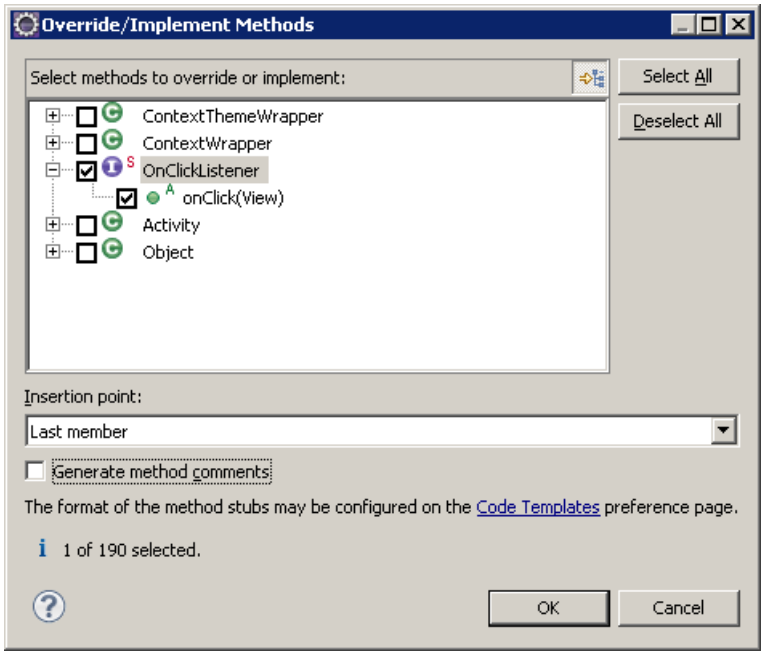
Import nécessaire

L'interface "OnClickListener" doit être implantée à la classe "NouveauProjet…"

Activation de l'écouteur. "this" fait référence à l'objet courant : l'écouteur "OnClickListener" est implanté dans l'objet "NouveauProjet…", donc dans l'activité 1

On peut noter la notification d'une "erreur" ici. En fait, il s'agit d'une notification pour implémenter la méthode "onClick". Cela est réalisé automatiquement avec un clic droit sur la ligne et en sélectionnant "source", puis "Implements Methods" :





Note : veiller à choisir "Last member" pour placer la méthode à la fin du source.

Avec un clic sur OK on obtient :

```
package fr.couffignal;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Chronometer;

public class NouveauProjetAndroidActivity extends Activity implements OnClickListener
{

// Déclaration des instances des widgets de l'écran "main"
 Button monBouton;
 Chronometer monChrono;

 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState)
 {
  super.onCreate(savedInstanceState);
  // Affichage de l'écran de démarrage
  setContentView(R.layout.main);
  // Création des instances du bouton et du chronomètre de l'écran "main"
  monBouton=(Button)findViewById(R.id.monBoutonId);
  monChrono=(Chronometer)findViewById(R.id.monChronoId);
  // "Ecouteur" du bouton
  monBouton.setOnClickListener(this);
 }

 @Override
 public void onClick(View v)
 {
    // TODO Auto-generated method stub

 }
}
```

Méthode "onClick" implémentée automatiquement

Résultat : l'écouteur "onClickListener" de cette classe appellera la méthode "onClick" à l'appui sur le bouton.

La méthode "onClick" ne comporte aucun traitement pour l'instant. Le traitement que l'on va ajouter consiste à déclencher le chronomètre :

```
 @Override
 public void onClick(View v)
 {
  // TODO Auto-generated method stub
  // Démarrage du chronomètre
  monChrono.start();
 }
}
```

Méthode de démarrage du chronomètre. En fait, le chronomètre démarre avec l'application; cette méthode ne fait que valider le rafraichissement sur l'écran.

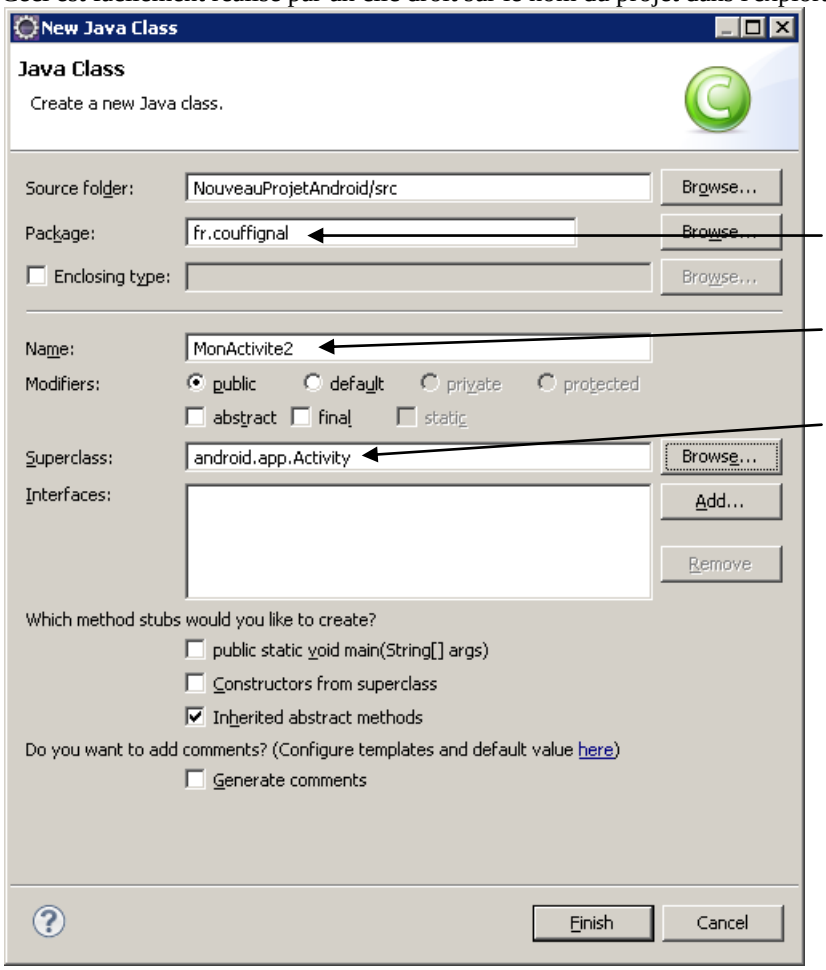
On peut tester tout cela sur l'émulateur ou une tablette Android réelle (clic droit sur le projet dans l'explorateur, puis "Run As Android Application")

4.4.3 Ouverture de l'écran d'édition

On aborde ici la notion d'***Intent***.
Les ***Intent*** permettent d'envoyer et recevoir des messages (éventuellement avec des données) pour déclencher une action, dans un composant d'une même application (exemple : une ***Activity***) voir même dans une autre application. Dans notre cas, nous allons utiliser l'***Intent*** pour démarrer une autre ***Activity***, puis échanger des messages entre les 2 activités.
Les ***Intent*** permettent aussi la communication inter-applications.

4.4.3.1 Création de la classe de la 2° activité : "MonActivite2"

Ceci est facilement réalisé par un clic droit sur le nom du projet dans l'explorateur, puis "New/Class" :

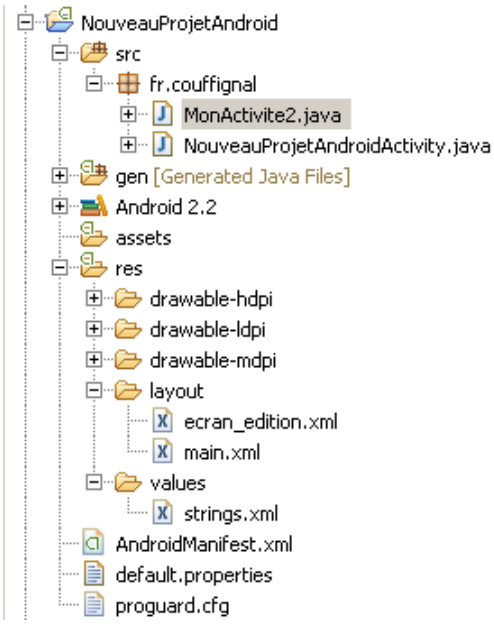


Attention : même package.

Nom de la nouvelle classe

Classe héritée de "Activity" (on peut utiliser le bouton "Browse" et taper "Ac..".

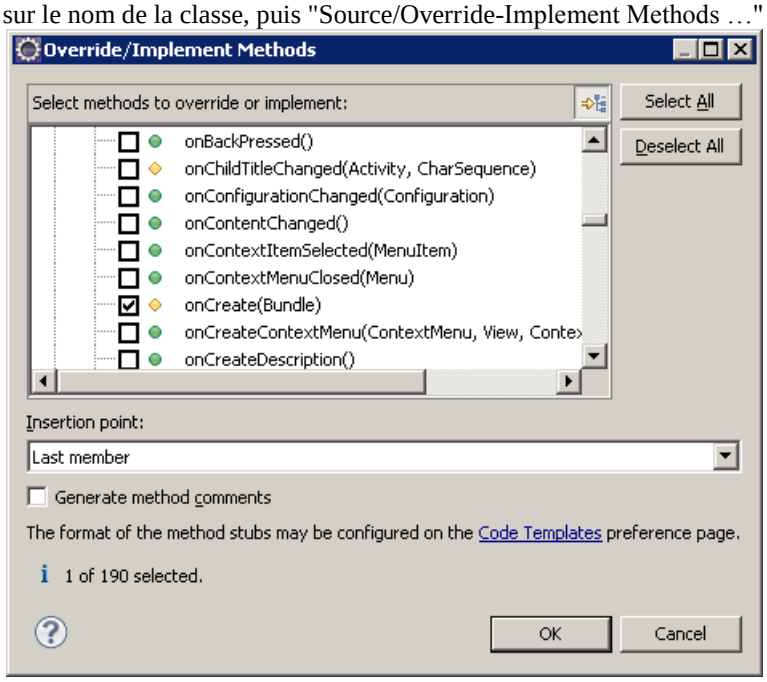
On obtient alors la structure de projet suivante :



et une classe "MonActivite2" au code très réduit (!) :

```
package fr.couffignal;

import android.app.Activity;

public class MonActivite2 extends Activity
{

}
```

Comme au §2.4, il faut "overrider" la méthode "onCreate". Le code correspondant peut être facilement implanté avec un clic droit sur le nom de la classe, puis "Source/Override-Implement Methods …" :



On obtient :

```
package fr.couffignal;

import android.app.Activity;
import android.os.Bundle;

public class MonActivite2 extends Activity
{

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
    }

}
```

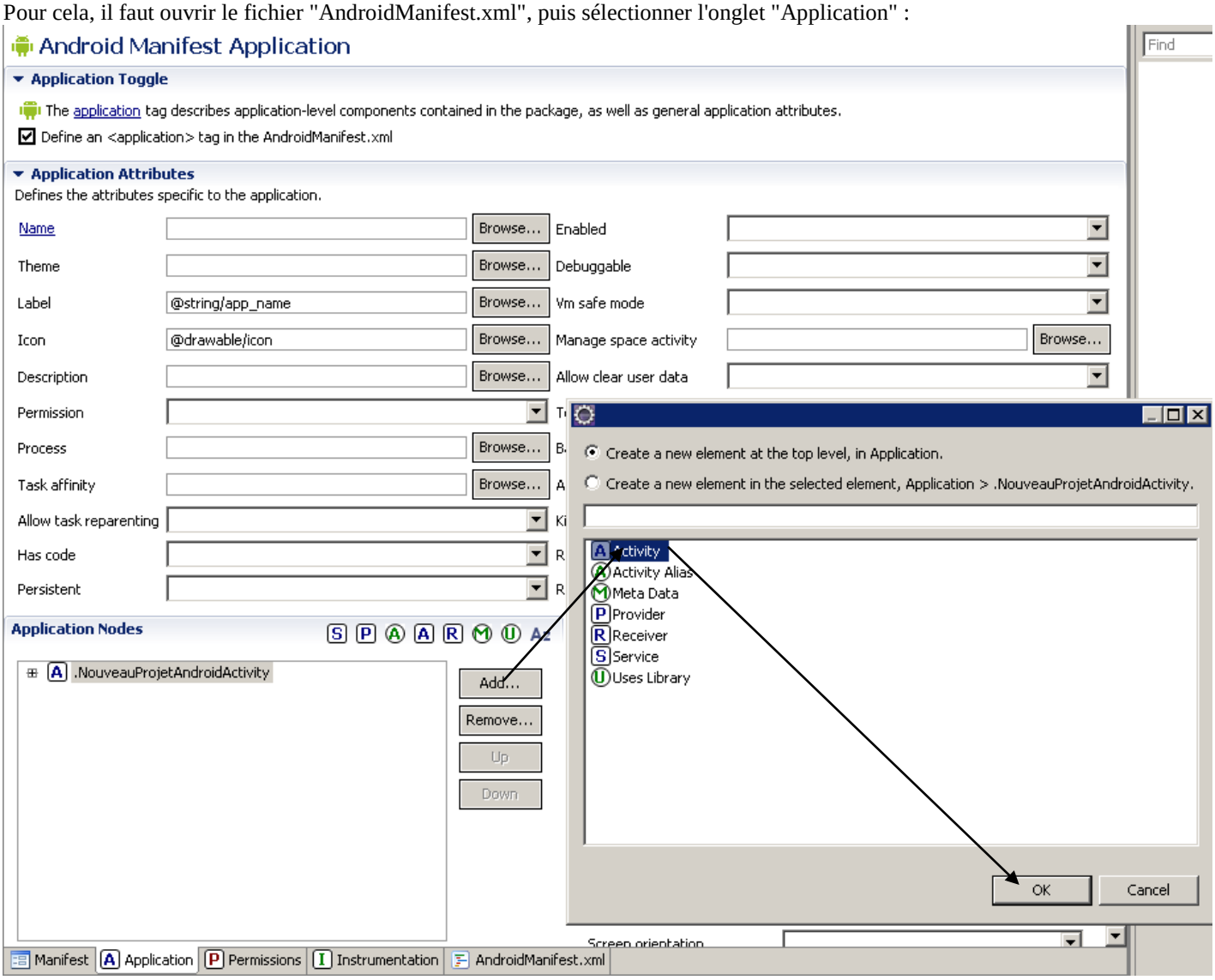
Pour afficher le 2° écran, on ajoute la méthode "setContentView" avec le bon argument dans "onCreate" :

```
package fr.couffignal;

import android.app.Activity;
import android.os.Bundle;

public class MonActivite2 extends Activity
{
 @Override
 protected void onCreate(Bundle savedInstanceState)
 {
  // TODO Auto-generated method stub
  super.onCreate(savedInstanceState);
  // Affichage de l'écran d'édition
  setContentView(R.layout.ecran_edition);
 }
}
```

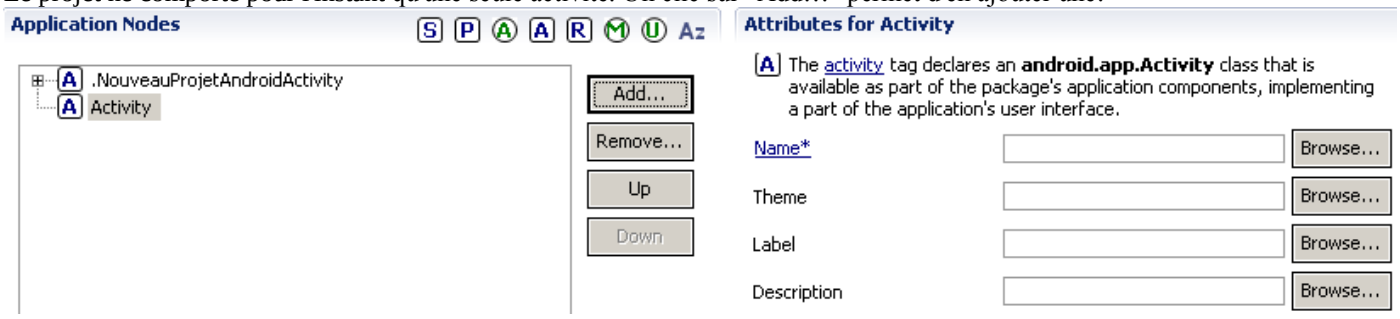
Mais pour l'instant, aucun évènement ne lance la 2° activité et l'écran "ecran_edition" reste invisible.

4.4.3.2 Ajout de la nouvelle activité dans "AndroidManifest.xml"

Pour cela, il faut ouvrir le fichier "AndroidManifest.xml", puis sélectionner l'onglet "Application" :



Le projet ne comporte pour l'instant qu'une seule activité. Un clic sur "Add…" permet d'en ajouter une.



Un clic sur "Browse" permet de choisir la 2° activité

4.4.3.3 Lancer la 2° activité à partir de la 1°

Pour lancer une activité, il faut créer et instancier un objet de type ***Intent*** qui se chargera de faire le lancement de l'activité à travers la plateforme Android.

```
@Override
public void onClick(View v)
{
 // TODO Auto-generated method stub
 // Démarrage du chronomètre
 monChrono.start();
 // Instanciation de l'objet "monIntent" utilisé pour communiquer avec la 2° activité
 Intent monIntent=new Intent(this,MonActivite2.class);
}
}
```

La fonction "Intent" nécessite deux arguments : le contexte (notre activité, donc ici "this") et l'activité de destination.

La méthode startActivity qui prend en paramètre un ***Intent***, nous servira à lancer l'activité demandée :

```
@Override
public void onClick(View v)
{
 // TODO Auto-generated method stub
 // Démarrage du chronomètre
 monChrono.start();
 // Instanciation de l'objet "monIntent" utilisé pour communiquer avec la 2° activité
 Intent monIntent=new Intent(this,MonActivite2.class);
 // Lancer l'activité "MonActivite2"
 startActivity(monIntent);
}
}
```

On peut constater le bon fonctionnement de l'application avec l'émulateur ou sur une tablette réelle. Des problèmes subsistent :

- le texte édité n'est pas utilisé
- on ne peut pas fermer cette fenêtre d'édition

4.5 Retour d'information de "MonActivite2" vers "NouveauProjetAndroidActivity"

Cette fonctionnalité nécessite un lancement différent de la 2° activité :

```
@Override
public void onClick(View v)
{
 // TODO Auto-generated method stub
 // Démarrage du chronomètre
 monChrono.start();
 // Instanciation de l'objet "monIntent" utilisé pour communiquer avec la 2° activité
 Intent monIntent=new Intent(this,MonActivite2.class);
 // Lancer l'activité "MonActivite2"
 //startActivity(monIntent);
 startActivityForResult(monIntent, 0);
}
}
```

La méthode "startActivityForResult" lance l'activité de "monIntent" (soit "MonActivite2") et prépare un canal de communication pour recevoir en retour un message de cette activité. Le 2° argument de valeur 0 est le code de retour qui sera ensuite utilisé pour filtrer les messages reçus.

La classe "MonActivite2" doit être modifiée pour envoyer le message édité à l'appui du bouton "Valider" :

```
package fr.couffignal;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MonActivite2 extends Activity implements OnClickListener
{
 // Déclaration de l'instance du bouton "Valider" de l'écran d'édition
 Button monBtnValider;
 // Déclaration d'un objet de type "EditText"
 EditText editTextNom;

 @Override
 protected void onCreate(Bundle savedInstanceState)
 {
  // TODO Auto-generated method stub
  super.onCreate(savedInstanceState);
  // Affichage de l'écran d'édition
  setContentView(R.layout.ecran_edition);
  // Instanciation de l'objet "monBtnValider" avec le widget de l'écran
  monBtnValider=(Button)findViewById(R.id.btnValiderId);
  // Instanciation de l'objet "editTextNom" avec le widget de l'écran
  editTextNom = (EditText)findViewById(R.id.editTextNomId);
  // "Ecouteur" du bouton
  monBtnValider.setOnClickListener(this);
 }

 @Override
 public void onClick(View v)
 {// Traitements après un clic sur "Valider"
    // TODO Auto-generated method stub
    // On vérifie que la taille de la chaine de retour est supérieur à 0
    if (editTextNom.getText().length() > 0)
    {
      // Instanciation de l'objet "monIntent" utilisé pour communiquer
      // avec la 1° activité
      Intent monIntent2 = new Intent();
      // On ajoute le nom saisi dans l'intent
      monIntent2.putExtra("Nom", editTextNom.getText().toString());
      // On retourne le résultat avec l'intent
      setResult(RESULT_OK, monIntent2);
      // On ferme cette activité
      finish();
    }
 }
}
```

L'interface "OnClickListener" doit être implantée à la classe "MonActivite2" pour l'écoute du bouton "Valider"

Pour créer une instance du widget "EditText"
Ici

La méthode "onClick" a été implantée comme au §2.4.2

On utilise les méthodes de la classe editTextNom pour obtenir :
la longueur du message
le message

L'objet "monIntent2" gère les échanges avec les autres activités

Affectation du message accompagné de son identificateur dans "monIntent2"

La chaîne "Nom" servira d'identificateur du message à la réception dans l'activité 1

Envoi effectif du message de retour. La constante "RESULT_OK" indique au destinataire la validité du message.

4.6 Afficher le message de retour dans l'activité 1

La méthode de la classe "Activity" permettant ceci se nomme "onActivityResult". Le plus simple est de l'implanter suivant la procédure décrite au §2.4.2. On obtient ceci :

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    // TODO Auto-generated method stub
    super.onActivityResult(requestCode, resultCode, data);
}
```

Il reste à ajouter les méthodes permettant l'affichage du message envoyé par l'activité 2 :

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
 // TODO Auto-generated method stub
 super.onActivityResult(requestCode, resultCode, data);
 // On récupère le message identifié par le paramètre "Nom" de l'intent
 String nom = data.getStringExtra("Nom");
 // On affiche le message
 Toast.makeText(this, "Votre nom est : "+nom,Toast.LENGTH_SHORT).show();
}
```

Méthode permettant d'afficher un message de façon temporaire sur l'écran.

Note : on ne réalise ici aucune vérification de l'expéditeur ou de la validité du message. Cela peut être fait en utilisant les arguments de la méthode onActivityResult :

- ***requestCode*** : il s'agit du code de retour qui a été utilisé comme argument de la méthode "startActivityForResult" (voir plus haut dans le source). Sa valeur devrait être 0 dans cet exemple.
- ***resultCode*** : ses valeurs peuvent être "RESULT_OK" ou "RESULT_CANCELED" suivant ce que l'utilisateur a fait dans l'activité 2. On peut utiliser cette valeur pour éviter d'afficher des messages erronés

On peut constater le bon fonctionnement de l'application avec l'émulateur ou sur une tablette réelle.

5. Ajouter une classe de services

Les services ont pour but de réaliser des tâches de fond sans aucune interaction avec l'utilisateur pour une durée indéfinie. Il existe deux types de services :

- les services locaux (ou LocalService) qui s'exécutent dans le même processus que l'application
- Les services distants (ou RemoteService) qui s'exécutent dans un processus différent de celui de l'application

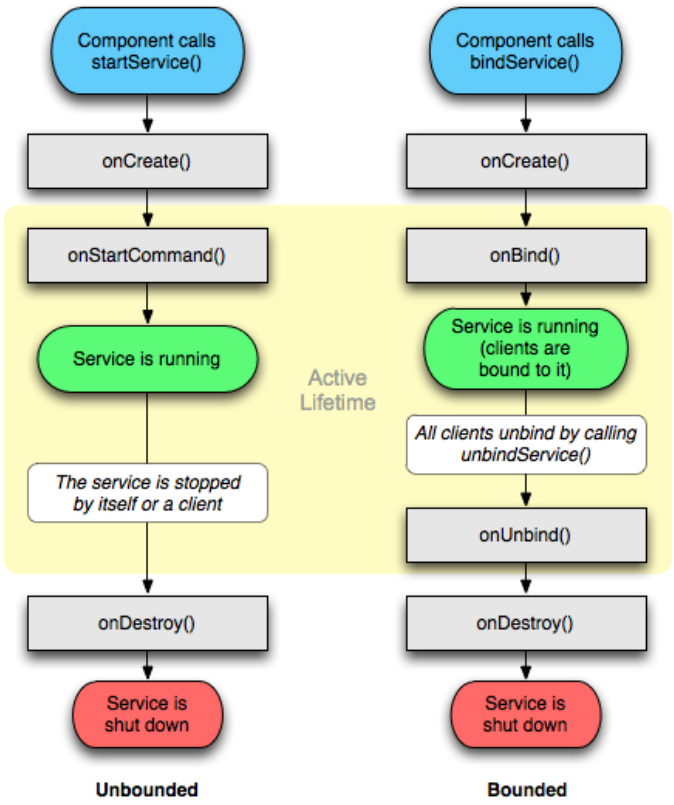
Les services s'exécutent dans le Thread (tâche ou unité d'exécution) principal du processus parent. Ils doivent être déclarés dans le fichier `AndroidManifest.xml` :

```
<service android:name=".subpackagename.ServiceName"/>
```

Ils doivent étendre la classe Service dont vous devrez surcharger les méthodes suivantes en fonction de vos besoins :

```
void onCreate();              // Initialisation des ressources
void onStart(Intent intent); // SDK<2.0 la tâche de fond démarre
void onStartCommand(Intent intent, int flags, int startId); // SDK>2.0 la tâche de fond démarre
void onDestroy();             // libération des ressources

IBinder onBind(Intent intent);   // Connexion client distant
boolean onUnbind(Intent intent); // Déconnexion d'un client
void onRebind(Intent intent)
```

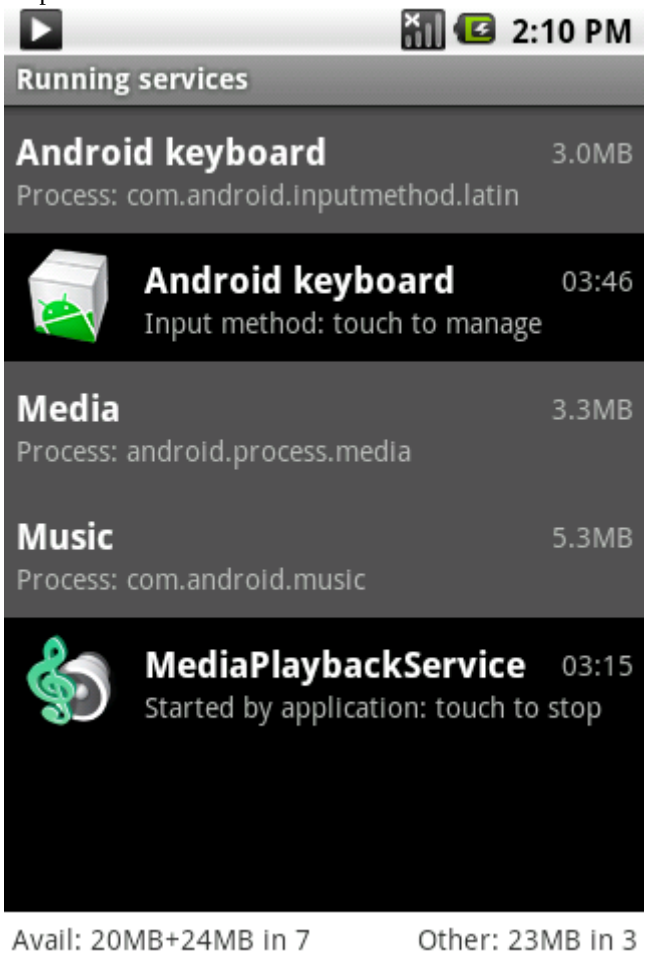
Quant au cycle de vie d'un service, Google l'illustre de la manière suivante :



Pour interagir (démarrer/arrêter...) avec un service, deux possibilités :

- soit on appelle la méthode `startService()` qui invoque la méthode `onCreate()` puis `onStart()`
  service.startService() → onCreate() → onStartCommand() [service running]
  L'appel de la méthode `stopService()` invoque la méthode onDestroy()
- soit on appelle la méthode `bindService()` qui appelle uniquement la méthode `onCreate()`
  activity.bindService() | → onCreate() [service created]

Il est possible de voir la liste des services exécutés en allant dans *Menu* > *Settings* > *Applications* > *Running Services* > du téléphone:

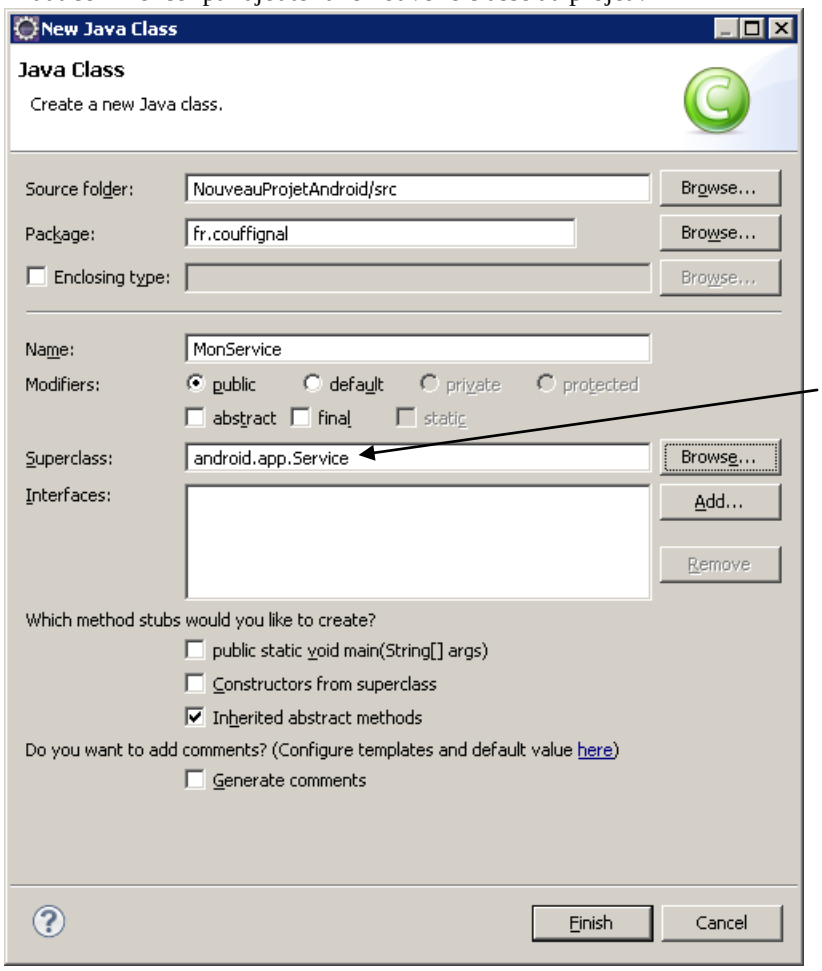


5.1 Exemple de service local

Tâches du service :
Le service initialise un Timer et l'utilise pour appeler une fonction toutes les 10 secondes. Cette fonction affichera un message de contrôle sur l'écran actuel.
La classe "MonService" hérite de Service et doit surcharger les méthodes onCreate(), onStart() et onDestroy().

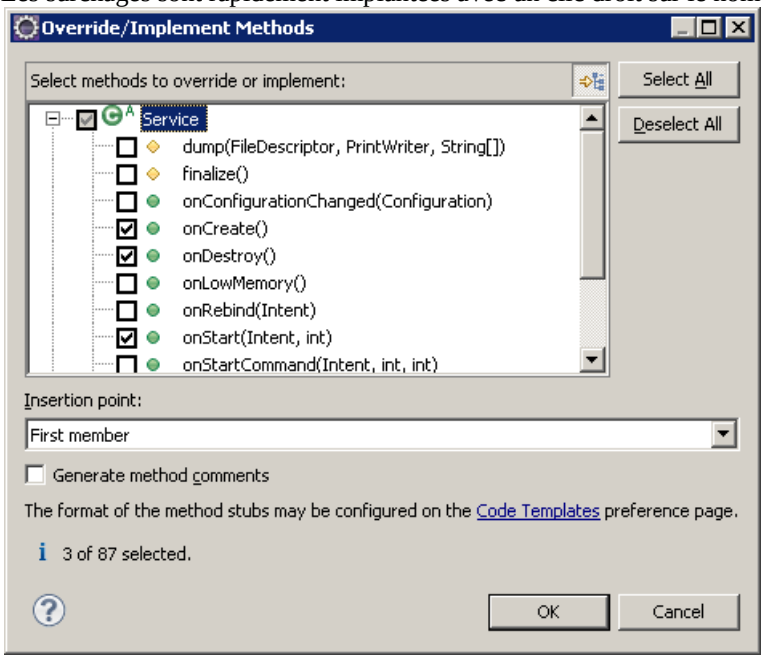
5.1.1 Création de la classe "MonService"

Il faut commencer par ajouter une nouvelle classe au projet :



Classe mère = Service

Les surchages sont rapidement implantées avec un clic droit sur le nom de la classe, puis "Source/Override-Methods …" :



On obtient alors :

```
package fr.couffignal;

import java.util.Timer;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MonService extends Service
{
 @Override
 public void onCreate()
 {
  // TODO Auto-generated method stub
  super.onCreate();
 }

 @Override
 public void onDestroy()
 {
  // TODO Auto-generated method stub
  super.onDestroy();
 }

 @Override
 public void onStart(Intent intent, int startId)
 {
  // TODO Auto-generated method stub
  super.onStart(intent, startId);
 }

 @Override
 public IBinder onBind(Intent intent)
 {
  // TODO Auto-generated method stub
  return null;
 }

}
```

5.1.2 Déclaration et instanciation d'un objet de type "Timer"

```
// Instanciation de l'objet monTimer de type Timer
Timer monTimer = new Timer();
```

Un objet "Timer" est utilisé pour :

- exécuter une seule fois une tâche après un délai choisi
- exécuter une tâche de façon répétitive, avec un délai et une période réglables.

C'est cette 2° possibilité qui sera exploitée ici.

Chaque "Timer" a son propre "thread" exécuté séquentiellement par le système d'exploitation; cela peut occasionner des délais au lancement de la tâche.

5.1.3 Déclaration et instanciation d'un objet de type "Handler"

On utilise la classe "Toast" pour afficher les messages temporaires. La fonction "run" de la classe "TimerTask" (la tâche qui est lancée toutes les 10s et qui nécessite l'affichage d'un message) a son propre thread et les méthodes "Toast" ***plantent*** si elles s'exécutent dans ce thread.
Il faut donc créer un objet "Handler" qui a son propre thread ("MessageQueue") et qui sera chargé des traitements des messages "Toast".

```
public char c = '0';
// Les messages "Toast" de la fonction "run" de la classe "TimerTask"
// nécessitent un "Handler" pour placer leurs traitements dans le
// thread "MessageQueue" (sinon plantage !)
Handler toastHandler = new Handler()
{
 @Override
 public void handleMessage(Message msg)
 {
  Toast.makeText(getApplicationContext(),"Tâche timer "+c, Toast.LENGTH_SHORT).show();
 }
};
```

La méthode surchargée "handleMessage" est invoquée par un appel de la méthode "toastHandler.sendEmptyMessage(0)".

5.1.4 Classe "myTimerTask" héritée de "TimerTask"

Cette classe comporte la méthode surchargée "run" qui sera appelée périodiquement.

```
// Fonction argument de la fonction "monTimer.scheduleAtFixedRate'
private class myTimerTask extends TimerTask
{
 @Override
 public void run()
 {// Traitement exécuté toutes les 10000mS = 10s
  c++;
  // Pour invoquer la méthode surchargée "handleMessage"
  //toastHandler.sendMessage(toastHandler.obtainMessage());
  toastHandler.sendEmptyMessage(0);
 }
}
```

Cette incrémentation permet de modifier le message à chaque lancement de la tâche

L'appel de l'une de ces méthodes entraine l'exécution de la méthode "handleMessage" décrite ci-dessus.

Le traitement est volontairement simplifié pour la démonstration.

5.1.5 Surcharge de la méthode "onCreate"

```
// Attention : le timer est démarré dans "onCreate" car la méthode "onStart"
// est lancée à chaque fois que la tablette est tournée ce qui multiplierait
// les classes "TimerTask"
@Override
public void onCreate()
{
 // TODO Auto-generated method stub
 super.onCreate();
 // Configuration du timer
 monTimer.scheduleAtFixedRate(new myTimerTask(), 0, 10000);
 // On affiche un message indiquant le démarrage du service
 Toast.makeText(this, "Le service est démarré ...",Toast.LENGTH_SHORT).show();
}
```

Délai = 0s
Période = 10s

La dernière ligne affiche un message temporaire pour indiquer que le service est démarré. La méthode "Toast" utilise le thread du service (contexte this), il est donc inutile ici de passer par un "Handler".

On peut tester tout cela sur l'émulateur ou une tablette Android réelle (clic droit sur le projet dans l'explorateur, puis "Run As Android Application")