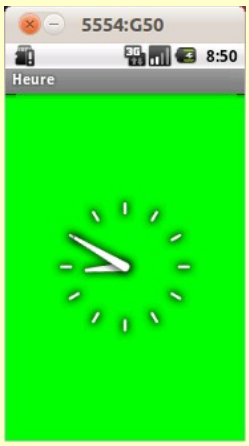
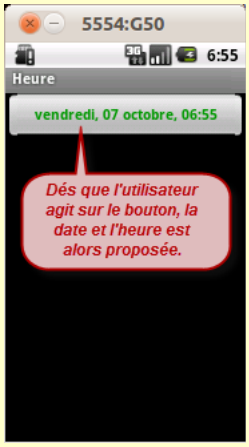
Cette étude va nous permettre d'écrire nos premières lignes de code pour des smartphones sous un environnement Android. Lors de cette étude nous découvrirons les points suivants :

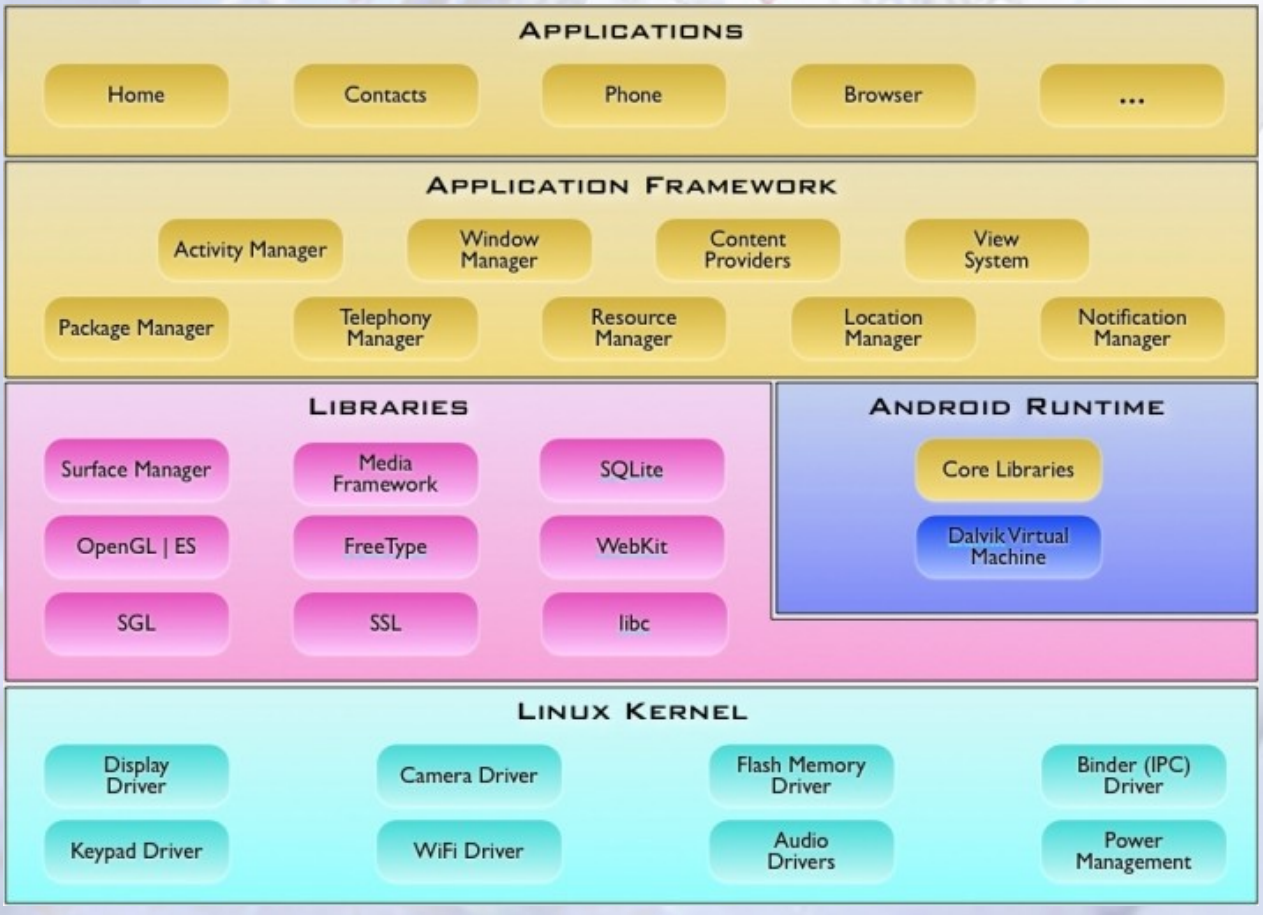
- *L'architecture générale d'un système Android.*
- *Outils de développement, l'émulateur, les AVD (Android Virtual Device).*
- *Structure d'un projet Android et contenu d'un programme Android.*
- *Création d'interface utilisateur.*
- *Les ressources, les widgets de base, les conteneurs.*
- *Voici quelques exemples d'activités développées lors de cette introduction :*





Nota : *Android est une plate-forme ouverte pour le développement de mobiles (smartphones). C'est la première plate-forme pour appareils mobiles qui soit réellement ouverte et complète avec tout le logiciel nécessaire au fonctionnement d'un téléphone mobile mais sans les obstacles propriétaires qui ont entravé l'innovation sur ces derniers.*

ARCHITECTURE GÉNÉRALE D'UN SYSTÈME ANDROID



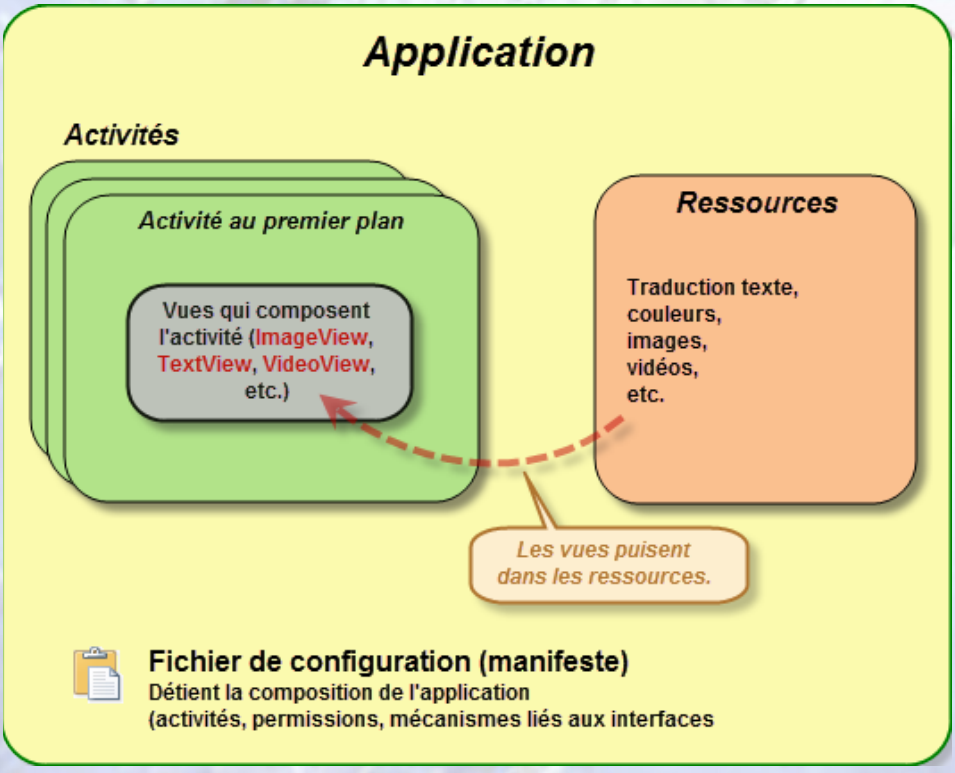
Android est en réalité un ensemble de technologies, qui prennent leur départ depuis le système d'exploitation pour finir dans les applications disponibles pour les utilisateurs. Il est donc constitué :

- ***Un système d'exploitation Linux*** *assurant l'interface de bas-niveau avec le matériel, la gestion de la mémoire et le contrôle des processus, le tout optimisé pour les appareils mobiles.*
- ***Des bibliothèques de base*** *codées en C++ exécutées juste au-dessus du noyau Linux, comme SQLite, Webkit, OpenGL, etc.*
- ***Un moteur d'exécution et d'hébergement des applications d'Android*** *incluant la machine virtuelle Dalvik et les bibliothèques de base fournissant les fonctionnalités spécifiques à Android. Ce moteur est conçu avec un soucis de compacité et d'efficacité pour un usage sur des appareils mobiles.*
- ***Un framework applicatif*** *qui fournit les classes utilisées pour créer des applications Android. Il fournit également une abstraction générique pour l'accès au matériel et gère l'interface utilisateur ainsi que les ressources de l'application.*
- ***Une couche applicative :*** *Toutes les applications qu'elles soient natives ou tierces sont construites sur la couche applicative par le moyen de mêmes bibliothèques d'API. La couche applicative est exécutée par le moteur d'exécution Android et utilise les classes et les services rendus disponibles par le framework applicatif.*

CONSTITUTION D'UNE APPLICATION ANDROID

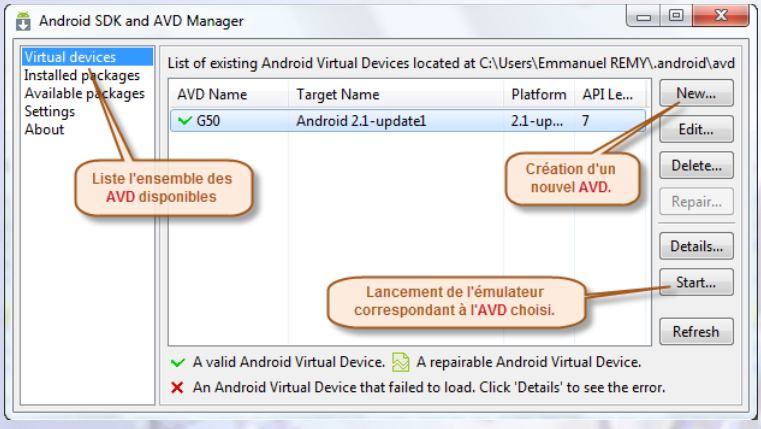
Une application Android est un assemblage de composants liés grâce à un fichier de configuration. Nous allons découvrir chaque pièce de notre puzzle applicatif et voir comment toutes les pièces interagissent entre elles. Les éléments constituant une application Android sont les suivants :

- ***Les activités*** *: une activité peut être assimilé à un écran structuré par un ensemble de vues et de contrôles composant son interface de façon logique ; elle est composée d'une hiérarchie de vues contenant elles-mêmes d'autres vues. Une activité est, par exemple, un formulaire d'ajout de contacts ou encore un plan Google Maps sur lequel vous ajoutez de l'information. Une application comportant plusieurs écrans, possédera donc autant d'activités.*
- ***Les vues*** *(et leur mise en page) : Les vues sont les éléments de l'interface graphique que l'utilisateur voit et sur lesquels il pourra agir. Les vues contiennent des composants, organisés selon diverses mises en page (les uns à la suite des autres, en grille ...).*
- ***Les contrôles*** *: les contrôles (boutons, champs de saisie, case à cocher, etc.) sont eux-même un sous-ensemble des vues. Ils ont besoin d'accéder aux textes et aux images qu'ils affichent (par exemple, un bouton représentant un téléphone aura besoin de l'image du téléphone correspondante). Ces textes et ces images seront puisés dans les fichiers ressources de l'application.*
- ***Les ressources*** *: Contiennent des fichiers statiques fournis avec l'application qui comprennent, des images, des textes et des couleurs prédéfinis, des dispositions de composants, etc.*
- ***Le fichier de configuration appelé également manifeste*** *: A côté de tous ces éléments, se trouve un fichier XML qui sert à la configuration de l'application. Ce fichier est indispensable à chaque application qui décrit : le point d'entrée de votre application (quel code doit être exécuté au démarrage de l'application), quels composants constituent ce programme, les permissions nécessaires à l'exécution du programme (accès à Internet, accès à l'appareil photo, etc.).*

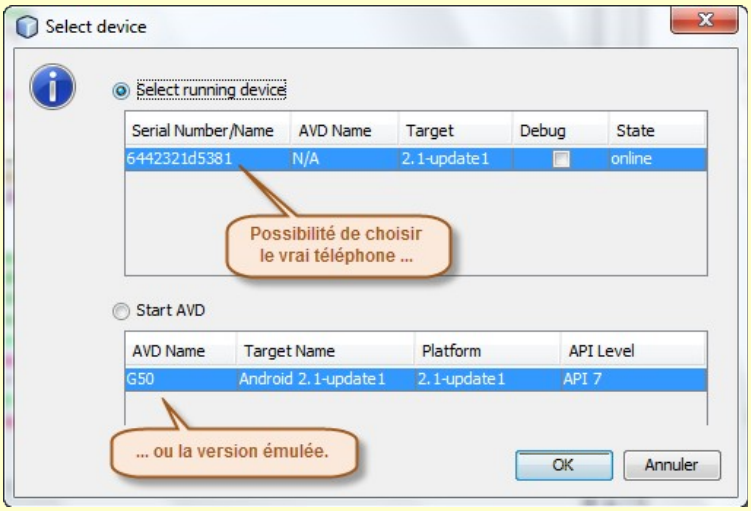


x ANDROID VIRTUAL DEVICE ET EMULATEUR ANDROID

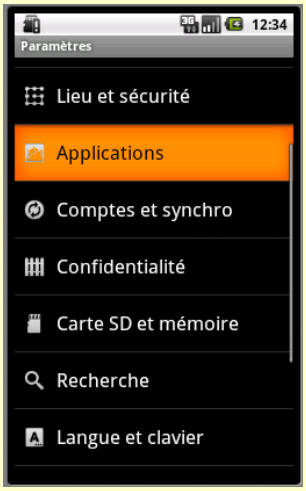
L'AVD (Android Virtual Device) est utilisé pour simuler les versions de logiciel et les spécifications de matériels disponibles sur différents appareils. Vous pouvez ainsi tester votre application sur plusieurs plate-formes matérielles sans avoir à acheter les téléphones correspondants.

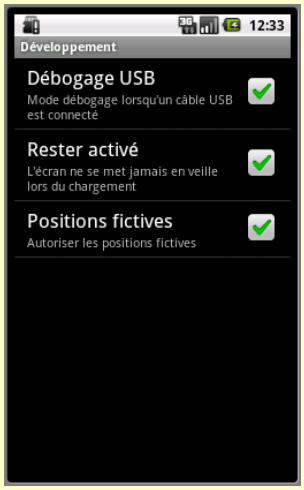


Une fois que votre projet est constitué, vous pouvez l'exécuter soit directement sur votre mobile ou prendre la version émulée :



*Pour que votre mobile accepte le déploiement de votre application, il faut le régler en conséquence. Pour cela, dans le menu **Paramètres**, sélectionner successivement **Applications** suivi de **Développement** et cocher ensuite toutes les cases concernées.*





QU'EST-CE QU'UNE ACTIVITÉ ANDROID ?

Une activité, comme nous l'avons déjà décrit, peut être assimilée à un écran qu'une application propose à son utilisateur. L'activité permet ainsi à l'utilisateur d'interagir avec l'application. Comme expliqué précédemment, plusieurs activités peuvent composer une même application. Pour chaque écran de votre application, vous devrez donc créer une activité.

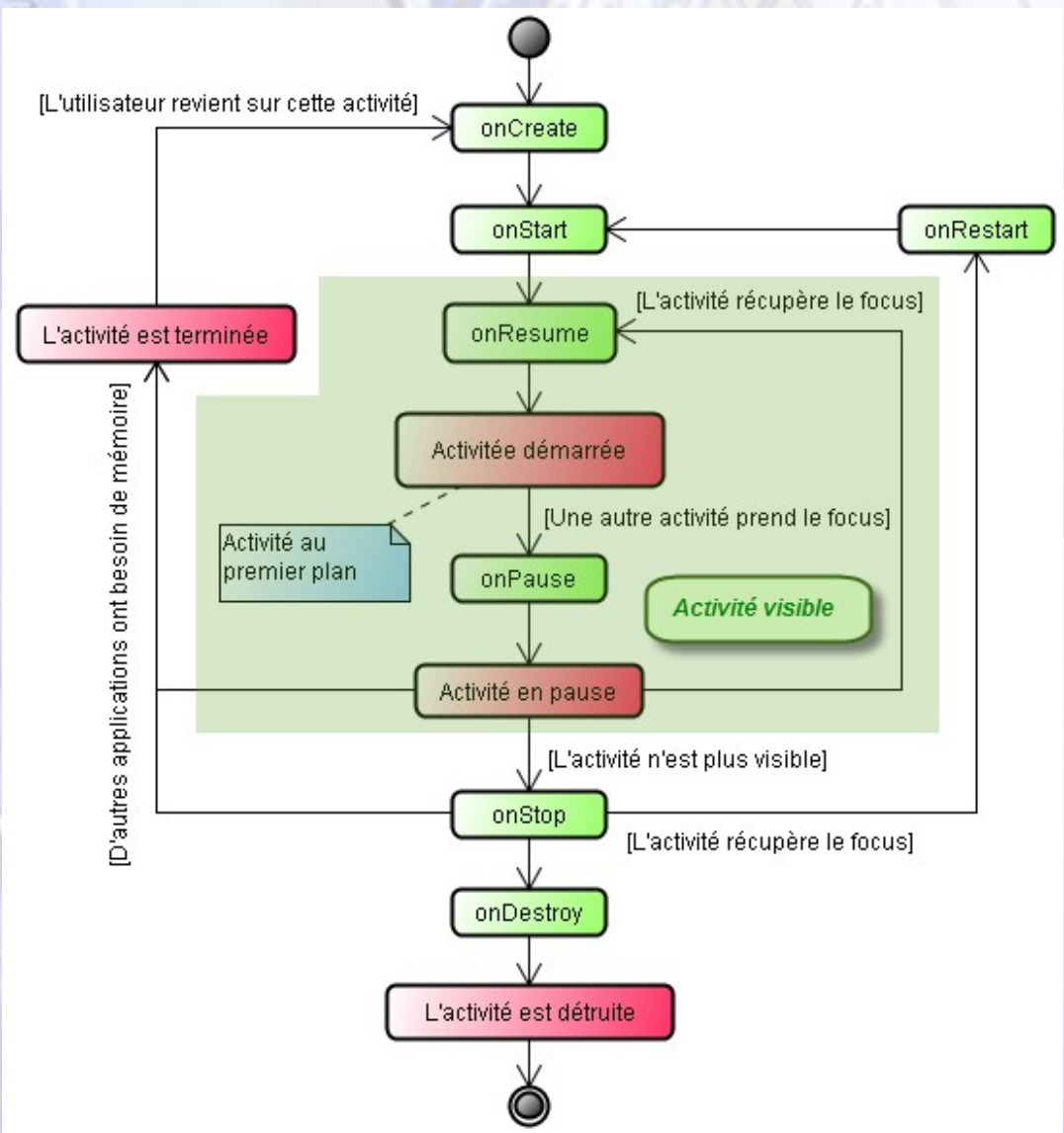
Il est préconisé de n'avoir qu'une interface visuelle par activité afin de faciliter la cinématique de l'application. En effet, la gestion du passage d'une activité à l'autre se voit ainsi mieux maîtrisée lors de l'évolution du cycle de vie de l'activité. La transition entre deux écrans correspond au lancement d'une activité ou au retour sur une activité placée en arrière plan.

Une activité est composée de deux volets :

- *La logique de l'activité et la gestion du cycle de vie de l'activité qui sont implémentés en Java dans une classe héritant de **Activity**.*
- *L'interface utilisateur, qui pourra être définie soit dans le code de l'activité soit de façon plus générale dans un fichier XML placé dans les ressources de l'application.*

CYCLE DE VIE D'UNE ACTIVITÉ

Une activité possède des méthodes de rappel (callback) appelées tout au long de son cycle de vie. Ces méthodes permettent de définir des passages d'un état à l'autre de l'activité par rapport à l'évolution des ressources et du comportement du système et de l'utilisateur. Le développeur peut ainsi définir des comportements particuliers à chaque état de celle-ci.

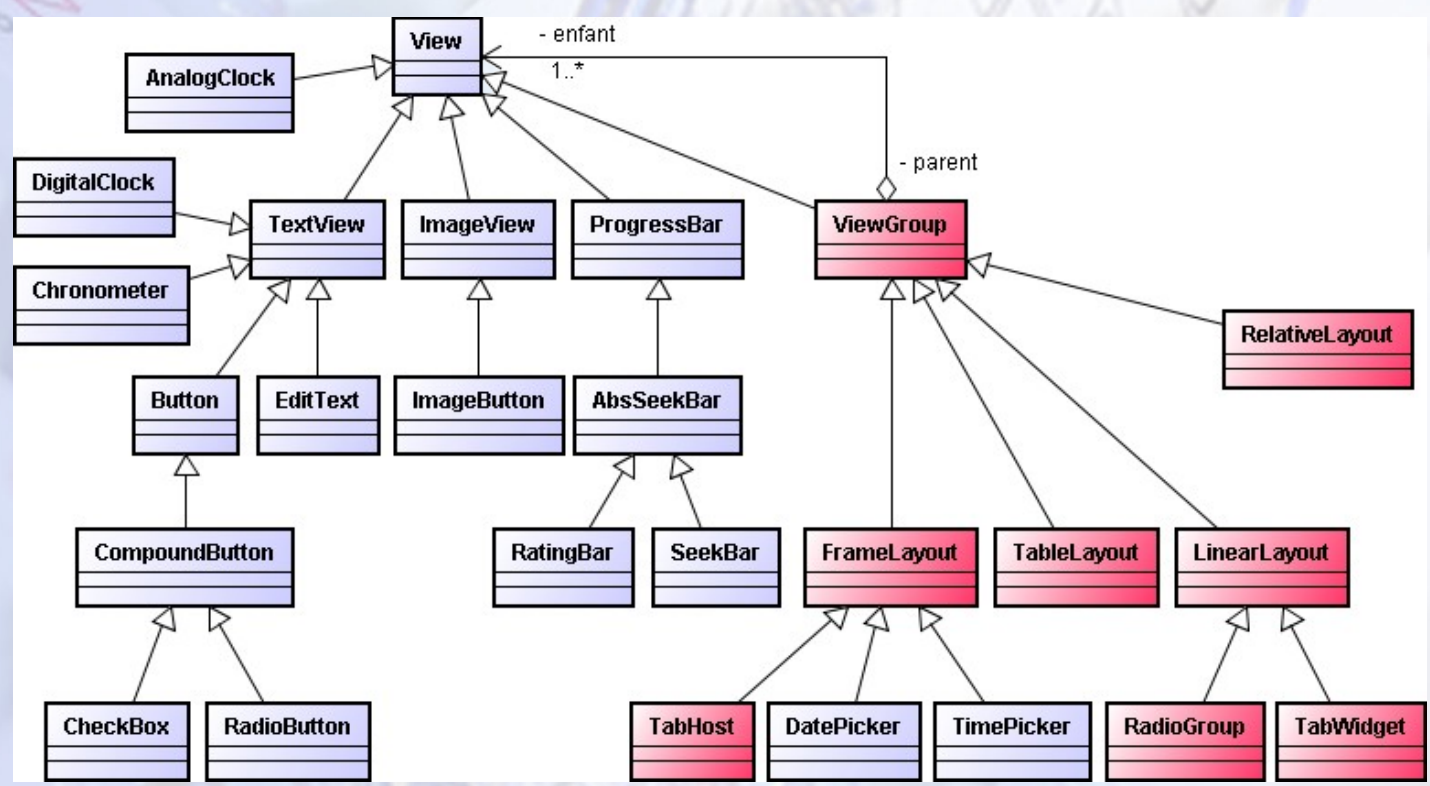


- ***L'activité démarre** : la méthode onCreate() est appelée.*

- **L'activité est active (démarrée)** : activité visible qui détient le focus et attend les entrées utilisateur. C'est l'appel à la méthode onResume(), à la création ou à la reprise après la pause qui permet à l'activité d'être dans cet état. Elle est ensuite mise en pause quand une autre activité devient active grâce à la méthode onPause().
- **Activité suspendue (en pause) :** activité au moins en partie visible à l'écran mais qui ne détient pas le focus. La méthode onPause() est invoquée pour entrer dans cet état et les méthodes onResume() ou onStop() permettent d'en sortir.
- **Activité arrêtée** : pendant l'utilisation d'une activité, l'utilisateur presse la touche Accueil, ou bien l'application téléphone, qualifiée comme prioritaire, interrompt l'activité en cours et prend en compte l'appel entrant. L'activité est arrêtée par l'appel de la méthode onStop(). Le développeur détermine l'impact sur l'interface utilisateur, par exemple la mise en pause d'une animation puisque l'activité n'est plus visible.
- **L'activité redémarre** : une fois l'appel téléphonique terminé, le système réveille l'activité précédemment mise en pause en appelant successivement les méthodes onRestart() et onStart().
- **Destruction de l'activité** : si l'activité reste trop longtemps en pause, le système a besoin de mémoire, il détruit l'activité au moyen de la méthode onDestroy().
- **Activité partiellement visible** : les méthodes onPause() et onResume() rajoutent un état à l'activité, puisqu'ils interviennent dans le cas d'activité partiellement visibles, mais qui n'ont pas le focus. La méthode onPause() implique également que la vie de cette application n'est plus une priorité du système. Donc, si celui-ci a besoin de mémoire, l'activité peut être fermée. Ainsi, il est préférable, lorsque nous utilisons cette méthode, de sauvegarder l'état de l'activité dans le cas où l'utilisateur souhaiterait y revenir avec la touche Accueil.

LES VUES

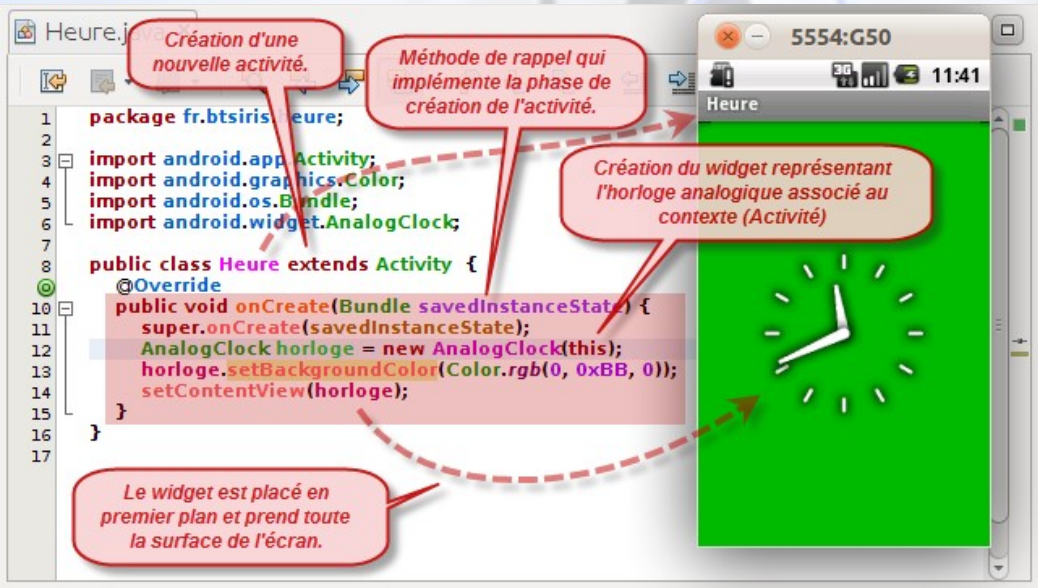
Une interface graphique est composée d'une multitude de composants graphiques : les vues. Sous Android, une vue est représentée, comme son nom l'indique, par la classe **View**. Tous les composants graphiques (boutons, images, case à cocher, etc.) d'Android héritent tous de cette classe **View** et sont communément appelées des ***widgets***. Vous avez ci-dessous la hiérarchie de quelques vues qui sont souvent utilisées :



- **TextView** : Cette vue permet d'afficher un texte à l'écran et gère de nombreuses fonctionnalités relatives à l'affichage du texte : couleur, police, taille, graisse, etc.
- **Button** : Comme son nom l'indique, cette classe représente un bouton. Puisque cette classe hérite de TextView, elle dispose de toutes les capacités de sa classe parente. La seule particularité d'une vue Button vis-à-vis d'une vue TextView est son apparence. Un Button est en fait un texte affiché dans un bouton (rectangle gris aux coins arrondis) changeant automatiquement d'apparence lorsque l'utilisateur interagit avec ce dernier (clic, focus).
- **EditText** : Cette vue permet à l'utilisateur d'entrer un texte que l'application pourra récupérer et utiliser par la suite.
- **ImageView** : Avec TextView, ImageView est l'une des vues les plus utilisées dans les interfaces graphiques. Elle permet d'afficher une image et dispose de plusieurs options intéressantes : teinte, redimensionnement, etc.

PREMIÈRE EXPÉRIENCE DE MISE EN ŒUVRE – HORLOGE ANALOGIQUE

Nous allons maintenant rentrer dans le vif du sujet en proposant notre première activité qui consiste à afficher l'heure courante au travers d'une horloge dite « Analogique ». Pour cela, il existe déjà un widget qui réalise cette opération et qui se nomme ***AnalogClock***.



UTILISATION DES RESSOURCES

Dans un projet Android standard, le code Java et les ressources peuvent être structurellement dissociés. Nous venons d'en faire l'expérience, les ressources peuvent être écrites en code Java, toutefois le code peut très vite devenir complexe si nous devons assurer la gestion des différents types de terminaux. Chaque ressource peut donc être décrite dans un fichier XML spécifique. Cela permet ainsi de découpler clairement la logique de votre application de sa présentation.

> *Cette gestion des configurations se fait ainsi de manière totalement automatique, sans prise en compte particulière de ces caractéristiques au niveau du code de l'application. Le lien entre le code Java et les ressources XML auxquelles il fait appel, reste assuré par le fichier automatiquement généré : **R.java**.*

Les ressources de l'application Android sont stockées dans des fichiers situés sous le répertoire ***res*** de votre projet. Ce répertoire sert de racine et contient lui-même une arborescence de dossiers correspondant à différents types de ressources.

- ***Les images : res/drawable :*** *ce dossier contient l'ensemble des ressources graphiques mise en œuvre dans le cadre de votre projet, des images (PNG, JPEG, etc.), des vidéos, des icônes, etc. A la construction, ces images peuvent être optimisées automatiquement. Si vous projetez de lire une image bit à bit pour réaliser des opérations dessus, placez-la plutôt dans les ressources brutes.*
- ***Mise en page : res/layout*** *: ce dossier contient les fichiers décrivant la composition des interfaces (les vues) de l'application, décrites en format XML. Ces fichiers XML sont convertis en mise en page d'écrans (ou de parties d'écrans), que l'on appelle aussi gabarits.*
- ***Les menus : res/menu*** *: ce dossier contient la description XML des menus de l'application.*
- ***Les ressources brutes : res/raw :*** *ce dossier contient les ressources autres que celles décrites ci-dessus qui seront empaquetées sans aucun traitement spécifique.*
- ***Les valeurs simples : res/values*** *: ce dossier contient des messages, des dimensions, les couleurs, etc. prédéfinis qui sont très facile de modifier par la suite. Cela permet de découpler le code principal des différents réglages et interventions utltérieures possibles. Grâce à ce type de ressource, les chaînes, les couleurs, les tableaux et les dimensions permettent d'associer des noms symboliques à ces types de constantes et de les séparer du reste du code (pour l'internationalisation et la localisation, notamment).*
- ***Les animations : res/anim :*** *de manière à rendre les interfaces utilisateurs moins statiques, ce type de ressources permet d'animer des composants de type vue : des textes, des images, des groupes de vues.*
- ***Données personnalisées : res/xml*** *: ce dossier contient les autres fichiers XML généraux que vous souhaitez soumettre. Ces fichiers statiques permettent de stocker vos propres données et structures.*

UTILISATION DES LAYOUTS XML

Bien qu'il soit techniquement possible de créer et d'attacher des composants widgets à une activité en utilisant uniquement du code Java comme nous venons de le faire, nous préférons généralement employer un fichier de positionnement (**layout**) codé en XML.

*Une des grandes forces d'Android est de permettre la définition d'interfaces graphiques dans un format XML. Cette définition se réalise dans des fichiers XML où nous spécifions des relations existant entre les widgets, ainsi qu'avec leurs conteneurs. C'est une ressource stockée dans le dossier prévu à cet effet : **res/layout**. A ce sujet, grâce à Netbeans, vous disposez déjà d'un fichier de disposition des composants **main.xml** qu'il suffit de modifier.*

main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <AnalogClock xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      android:background="#00BB00"
6  />
7
```

Il existe une balise correspondant à chaque objet graphique qui porte le même nom que la classe, donc ici <AnalogClock>.

Il suffit ensuite de régler ses différents attributs pour avoir l'aspect visuel voulu.

R.java

```
7
8   package fr.btsiris.heure;
9
10  public final class R {
11      public static final class attr {
12      }
13      public static final class drawable {
14          public static final int icon=0x7f020000;
15      }
16      public static final class layout {
17          public static final int main=0x7f030000;
18      }
19      public static final class string {
20          public static final int app_name=0x7f040000;
21      }
22  }
23
```

Constante entière qui identifie l'icône principale de l'application.

Constante entière qui identifie le fichier principal des dispositions des composants.

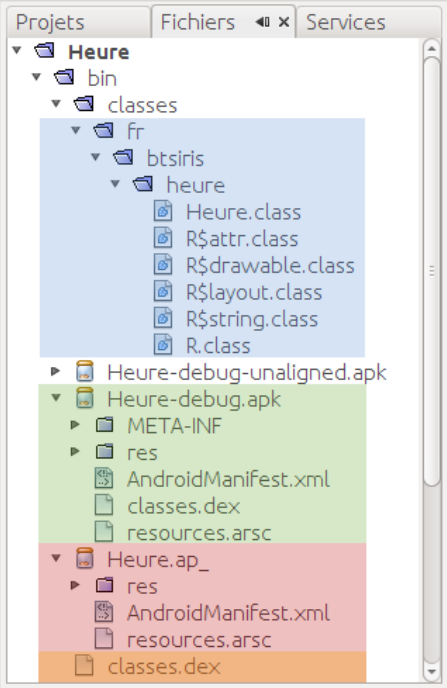
Constante entière qui identifie le nom de l'application.

Heure.java

```
1   package fr.btsiris.heure;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5
6   public class Heure extends Activity  {
7       @Override
8       public void onCreate(Bundle savedInstanceState) {
9           super.onCreate(savedInstanceState);
10          setContentView(R.layout.main);
11      }
12  }
13
```

L'activité se contente maintenant de prendre en compte ce qui est spécifié dans le fichier de description main.xml.

Puisqu'il s'agit d'une simple visualisation, aucun traitement particulier n'est proposé.



Cette façon de procéder est très largement utilisée puisqu'elle permet de façon simple de séparer l'aspect visuel du traitement en coulisse réalisé en code Java.

RÉSULTAT DE LA COMPILATION

Lorsque vous compilez un projet, le résultat est placé dans le répertoire **bin**, sous la racine de l'arborescence du projet :

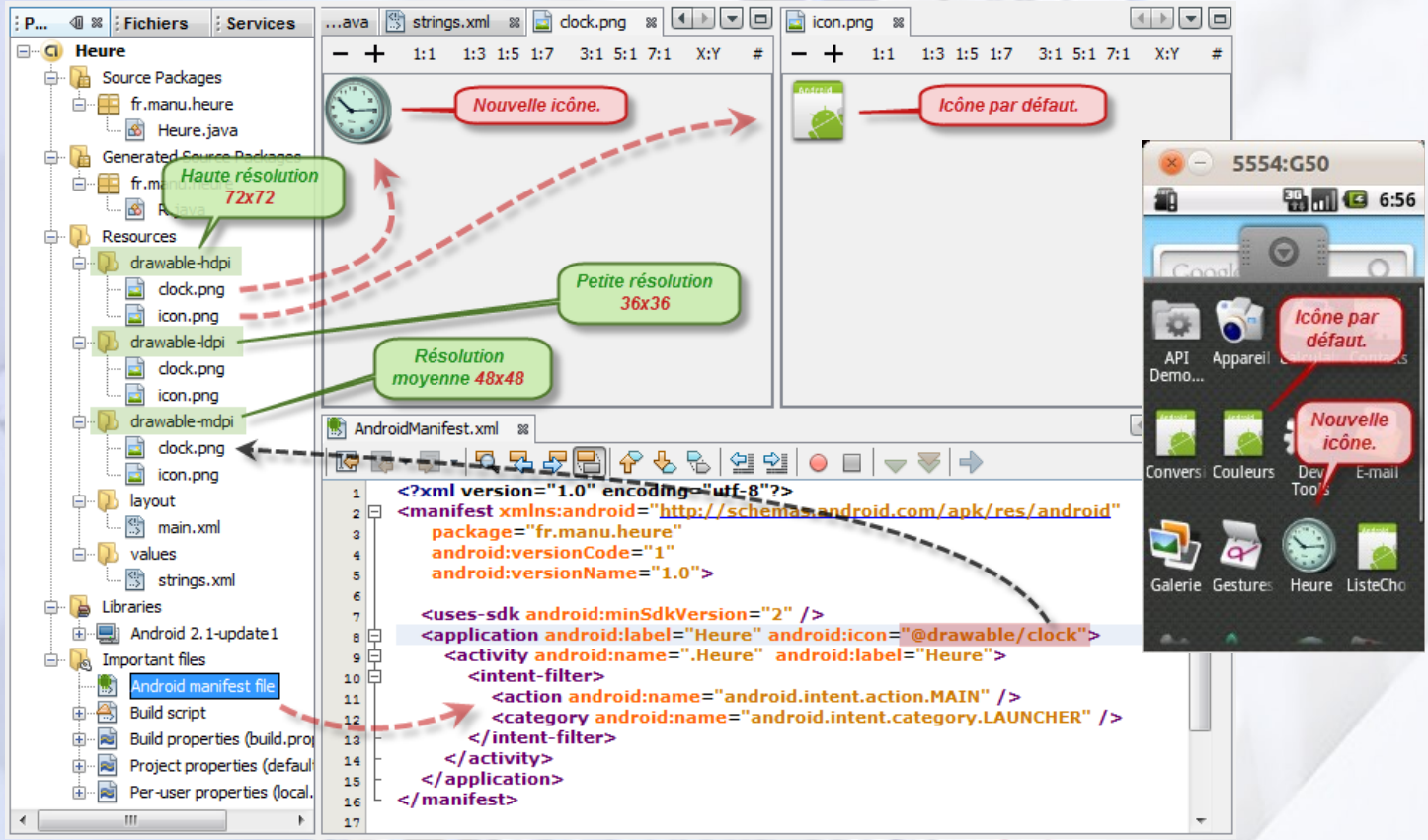
- ***bin/classes*** *: contient toutes les classes Java compilées de votre application.*
- ***bin/classes.dex*** *: contient l'exécutable créé à partir de ces classes compilées. C'est cet exécutable qui est lancé à partir de la machine virtuelle Dalvik.*
- ***bin/votre_application.ap_*** *: contient les ressources complètes de votre application, sous la forme d'un fichier ZIP.*
- ***bin/votre_application-debug.apk*** *ou* ***bin/votre_application-unsigned.apk*** *: est la*

véritable application Android déployable sur votre mobile.

*Le fichier **.apk** est une archive ZIP contenant le fichier **.dex**, la version compilée des ressources (**resources.arsc**), les éventuelles ressources non compilées (celles qui se trouvent sous **res/raw**, par exemple) et le fichier **AndroidManifest.xml**. Cette archive est signée : la partie **-debug** du nom du fichier indique qu'elle l'a été à l'aide d'une clé de débogage qui fonctionne avec l'émulateur alors que **-unsigned** précise que l'application a été construite pour être déployée directement vers le mobile.*

PROPOSER UNE NOUVELLE ICÔNE

Par défaut, lorsque vous élaborez un nouveau projet, une icône standard vous est proposée. Il est possible, bien entendu, de choisir sa propre icône pour son application.



*Pour cela, il suffit de récupérer ou de fabriquer votre icône et de la placer dans le répertoire **res/drawable**. Vous remarquez toutefois qu'il existe trois répertoires portant ce même nom avec un suffixe différent, respectivement **hdpi**, **ldpi** et **mdpi**, qui correspondent à la taille des icônes suivant la densité de l'écran choisi.*

- ***drawable-ldpi** : Utilisée pour stocker des images de faible capacité pour des écrans à densité de pixels comprise entre 100 et 140 dpi. Dans le cas des icônes, choisissez une résolution, soit de **32x32**, soit comme c'est le cas ici 36x36.*
- ***drawable-mdpi** : Utilisée pour stocker des images de capacité moyenne pour des écrans à densité de pixels moyenne entre 140 et 180 dpi. Dans le cas des icônes, choisissez une résolution en gros de **48x48**.*
- ***drawable-hdpi** : Utilisée pour stocker des images de forte capacité pour des écrans à densité élevée comprise entre 190 et 250 dpi. Dans le cas des icônes, choisissez une résolution en gros de **72x72** (double par rapport à la faible densité).*
- ***drawable** ou **drawable-nodpi** : Utilisée pour des images ne devant pas être mises à l'échelle, quelle que soit la densité de l'écran.*

Afin de correspondre à la totalité des situations, suivant le type de mobile choisi, il est souhaitable de prévoir une icône par densité d'écran possible.

*Le nom donné au fichier représentant l'icône s'appelle **icon.png**. Lorsque vous désirez changer l'icône de l'application, la meilleure solution consiste à supprimer celles qui existent et de proposer les nouvelles en prenant le même nom. Il est toutefois possible de choisir un autre nom de fichier et de garder éventuellement l'icône proposée par défaut. Dans ce cas, il faut avertir le manifeste de l'application afin qu'il prenne en compte ce nouveau nom de fichier.*

- *Dans la figure ci-dessus, vous remarquez que nous avons modifié l'attribut **android:icon** de l'élément **<application>** en spécifiant la valeur **@drawable/clock** en lieu et place de **@drawable/icon**.*

SÉPARATION DE LA VUE ET DU TRAITEMENT EN COULISSE DANS LE CODE JAVA

Je vous propose maintenant de réaliser une deuxième expérience qui va nous permettre de réaliser un traitement spécifique lorsque l'utilisateur clique sur un bouton. Nous découvrirons ainsi comment se gère les événements sous Android. Cette expérience va également bien démontrer l'intérêt de la séparation entre la vue décrite au moyen des fichiers de description XML, et le traitement en coulisse spécifié dans le code Java.

main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Button xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@+id/bouton"
4      android:layout_width="fill_parent"
5      android:layout_height="wrap_content"
6      android:text="Cliquez sur le bouton"
7      android:textColor="#FF9900"
8      android:textStyle="bold"
9    />
```

*Création d'un objet de type **Button**, identifié **bouton**, dont l'aspect visuel est réglé dans le fichier de description XML.*

Heure.java

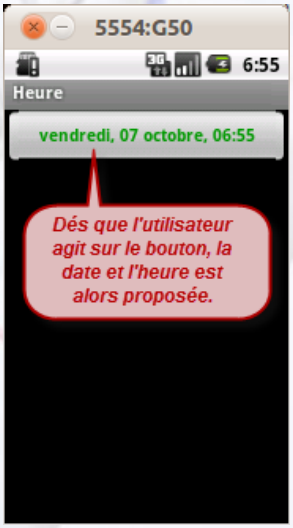
```
1  package fr.btsiris.heure;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.text.format.DateFormat;
6  import android.view.View;
7  import android.widget.Button;
8  import java.util.Date;
9
10 public class Heure extends Activity implements View.OnClickListener {
11   private Button bouton;
12
13   @Override
14   public void onCreate(Bundle savedInstanceState) {
15     super.onCreate(savedInstanceState);
16     setContentView(R.layout.main);
17     bouton = (Button) findViewById(R.id.bouton);
18     bouton.setOnClickListener(this);
19   }
20
21   public void onClick(View vue) {
22     CharSequence texte = DateFormat.format("EEEE, dd MMMM, hh:mm", new Date());
23     bouton.setText(texte);
24   }
25 }
26
```

L'activité prend en compte un événement de type Clic.

Récupération de l'objet généré lors de la lecture du fichier de description.

Le bouton est source de l'événement.

Traitement réalisé lorsque l'événement survient.



DÉFINITION D'UN IDENTIFIANT

Comme nous venons de le voir, Android propose la notion d'identifiant, largement utilisé en informatique, notamment pour les bases de données. Les identifiants permettent de définir, de manière unique, une entité, quelle qu'elle soit. Chaque vue Android peut ainsi disposer d'un identifiant défini grâce à l'attribut XML **android:id**.

*Nous avons découvert une syntaxe un peu particulière pour définir cet attribut. Toutefois, elle prend tout son sens après avoir décrit l'intérêt de ces symboles. Ainsi, lorsque nous écrivons **@+id/bouton**, voici à quoi cela correspond :*

- *@ : Ce caractère spécial dans les layouts Android exprime une indirection. Ainsi, le système comprend que la ressource pointée par l'identifiant n'est pas définie dans le fichier courant, mais dans un fichier externe.*
- *+ : Ce caractère permet d'indiquer à Android que l'identifiant peut être ajouté à la liste des identifiants de l'application si cela n'est pas déjà le cas.*
- ***id/** : En conjonction avec le @, le préfixe **id/** permet d'obliger Android à regarder dans la liste des identifiant déjà déclarés (par +).*
- ***bouton** : c'est le nom de l'identifiant que vous décidez de choisir. Attention, il doit bien entendu être unique.*

GÉRER LES ÉVÉNEMENTS

Comme beaucoup d'IHM, sous Android, toutes les actions de l'utilisateur sont perçues comme un événement, que ce soit une action sur un bouton d'une interface, le maintien de cette action, l'effleurement d'un élément de l'interface, etc. Ces événements peuvent être interceptés par les éléments de votre interface pour exécuter des traitements en conséquence.

Le mécanisme d'interception repose sur la notion d'écouteurs (listeners en anglais). Il permet d'associer un événement à une méthode à appeler en cas d'apparition de cet événement. Si un écouteur est défini pour un élément graphique avec un événement précis, la plate-forme Android appellera la méthode associée dès que l'événement se produira sur cet élément.

- *Notez que les types d'écouteurs sont prédéfinis. Ils sont représentés par des interfaces. Ils correspondent aux types d'événement à prendre en compte. Du coup, chaque type d'écouteur, possède une ou plusieurs méthodes qu'il est nécessaire de redéfinir pour préciser l'action à faire lors de la génération de l'événement.*
- *Comme nous venons de le voir lors de cette expérience, le type d'écouteur s'appelle **OnClickListener** et la méthode à prendre en compte et qu'il est nécessaire de redéfinir se nomme **onClick()**.*

SIMPLIFICATION DU CODE JAVA GRÂCE AUX SPÉCIFICATIONS ANDROID

Ce type d'événement est très fréquent sous Android. Les concepteurs ont prévu de simplifier le code Java grâce à l'utilisation de l'attribut **android:onClick** déclaré dans le document XML. Il suffit en effet de spécifier la méthode à appeler lors de la définition de cet attribut. Ainsi tout se passe en coulisse.

*De ce fait, dans le code Java, vous n'avez plus à créer une classe qui implémente l'interface **OnClickListener**. Il suffit juste de spécifier les actions à faire dans la méthode que vous avez choisi lors de la déclaration de cet attribut. Cette méthode doit toutefois posséder un attribut de type **View**, à l'image de la méthode **onClick()**.*

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/bouton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Cliquez sur le bouton"
    android:textColor="#FF9900"
    android:textStyle="bold"
    android:onClick="changerHeure"
/>
```

Nom de la méthode qui traite l'événement.

Heure.java

```
package fr.btsiris.heure;

import android.app.Activity;
import android.os.Bundle;
import android.text.format.DateFormat;
import android.view.View;
import android.widget.Button;
import java.util.Date;

public class Heure extends Activity {
  private Button bouton;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    bouton = (Button) findViewById(R.id.bouton);
  }

  public void changerHeure(View vue) {
    CharSequence texte = DateFormat.format("EEEE, dd MMMM, hh:mm", new Date());
    bouton.setText(texte);
  }
}
```

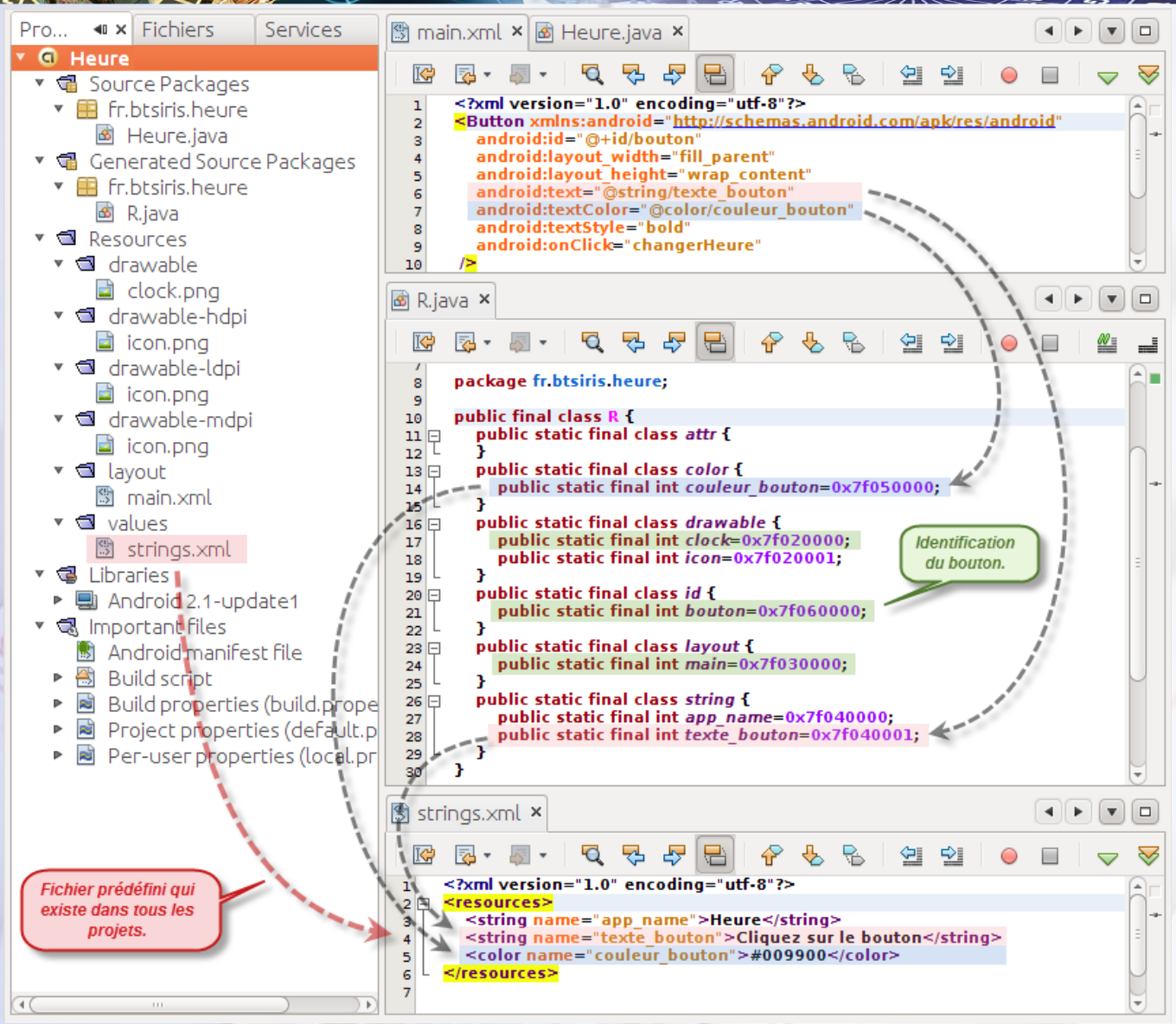
Plus besoin de spécifier la classe qui implémente l'interface OnClickListener.

Plus besoin de spécifier la source de l'événement.

Le nom de la méthode est à votre libre choix, mais doit posséder un paramètre de type View.

DIFFÉRENTS RÉGLAGES AU TRAVERS DE PLUSIEURS FICHIERS DE DESCRIPTION

Nous pouvons encore aller plus loin dans la séparation des différents réglages. Il existe un fichier de description prédéfini **strings.xml** dans lequel, comme son nom l'indique, vous pouvez spécifiez vos propres chaînes de caractères statiques qui peuvent servir pour les widgets et même pour le codage Java.



- Toutes les chaînes prédéfinies doivent être spécifiées à l'intérieur d'un élément **<string>**. Il suffit de désigner un identifiant au moyen de l'attribut **name** afin qu'elle puisse être prise en compte dans un autre document.
- Remarquez au passage que l'identifiant **app_name** à automatiquement été généré lors de l'élaboration du projet qui prend ainsi la valeur par défaut **Heure**. Il est bien sûr possible de changer ce nom, comme **"Obtention de l\'heure"** (Remarquez le caractère d'échappement dû à l'utilisation de l'apostrophe).
- Toutes les ressources que vous créez doivent être placées dans un document XML qui sert de descriptif dont l'élément racine doit impérativement être **<resources>**. Finalement, le nom du fichier lui-même importe peu, nous le découvrirons par la suite.
- Je profite de ce document de description pour définir une nouvelle couleur au moyen de la balise prédéfinie **<color>** qui elle aussi possède l'attribut **name** qui sert à identifier cette couleur spécifique.

Le choix du nom de ces balises prédéfinies est bien sûr d'une très grande importance (contrairement au nom du fichier lui-même), puisque c'est grâce à cette identification qu'il sera possible de récupérer le contenu de la chaîne de caractères prédéfinie ainsi que la couleur prédéfinie dans un autre document.

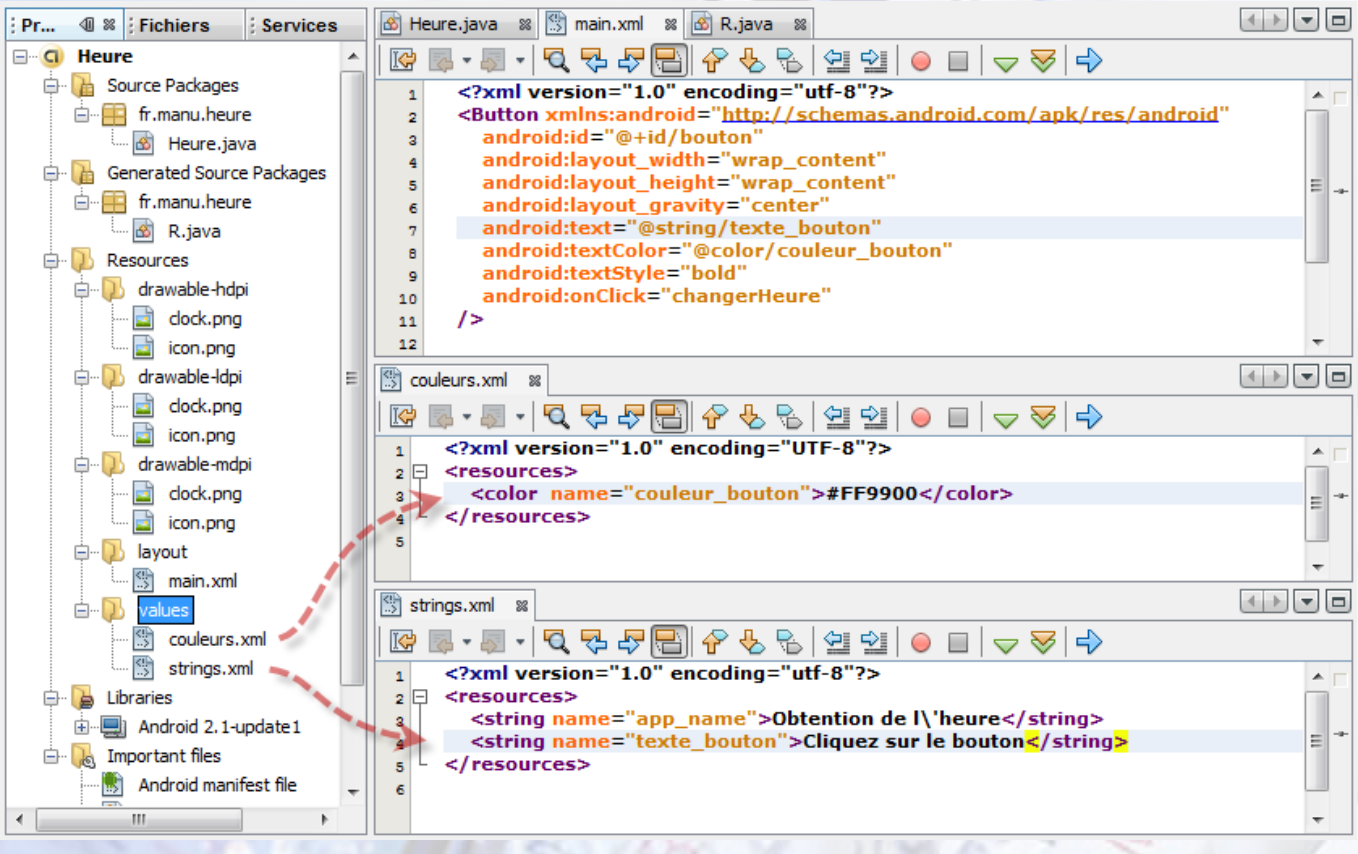
- Ainsi, pour récupérer le contenu d'une chaîne dans un document de description comme **main.xml**, il suffit d'écrire la syntaxe suivante **@string/** suivi de l'identifiant de la chaîne. Donc, pour récupérer le contenu "Cliquez sur le bouton" défini dans le descriptif **strings.xml**, vous devez spécifier la syntaxe suivante **@string/texte_bouton**.
- Dans le même ordre d'idée, pour prendre en compte la couleur définie dans le document **strings.xml**, il suffit d'utiliser la syntaxe **@color/** suivi de l'identifiant, donc ici **@color/couleur_bouton**.

Finalement la règle de cette syntaxe est très simple. Je rappelle que @ signifie que la ressource pointée par l'identifiant n'est pas définie dans le fichier courant, mais dans un fichier externe. Ensuite, nous devons mettre le type de ressource correspondant aux éléments que nous avons définis, donc **@string** relatif à l'élément **<string>** et **@color** relatif à l'élément **<color>**. Il suffit ensuite de placer l'identifiant que vous avez choisi en plaçant en intermédiaire le séparateur **"/"**.

CRÉATION D'UN DOCUMENT XML PROPRE À LA GESTION DES COULEURS

J'aimerais aller au bout de la démarche de séparation en proposant un fichier spécifique pour les couleurs afin de ne pas mélanger les types de ressource. En effet, comme son nom l'indique, le document **strings.xml** est prévu pour la description des chaînes de caractères prédéfinies dans votre application.

*Nous allons donc créer un fichier de description **couleurs.xml** que je place dans le même répertoire **res/values**. Ce fichier me servira à définir toutes les couleurs de l'application.*



POSITIONNER LES VUES AVEC LES GABARITS

Une interface graphique n'est pas uniquement composée de vues "feuilles". Il existe, en effet, des vues particulières permettant de contenir d'autres vues et de les positionner : les gabarits. Nous les appelons également par leur nom anglais : **layouts**. Avec Java SE, nous connaissons bien ce genre de technique qui consiste à positionner et dimensionner les composants automatiquement à l'aide de gestionnaires de dispositions.

*Ces vues particulières, ces gabarits, héritent toutes de la classe racine **ViewGroup** qui hérite elle-même de la classe **View**. Un gabarit est donc un conteneur qui aide à positionner les objets, qu'il s'agisse de vues ou d'autres gabarits au sein de votre interface. Vous pouvez imbriquer des gabarits les uns dans les autres, ce qui vous permettra de créer des mises en forme évoluées.*

- ***LinearLayout** : Ce gabarit aligne l'ensemble de ses enfants dans une direction unique, horizontale ou verticale. Les éléments se succèdent ensuite de gauche à droite ou de haut en bas. Ce gabarit n'affiche donc qu'un seul élément par ligne ou colonne (selon l'orientation choisie : horizontale ou verticale).*
- ***FrameLayout** : C'est le plus basique des gabarits. Ce gabarit empile les éléments fils les uns sur les autres tel une accumulation de calques. Par défaut, toutes les vues filles s'alignent sur le coin supérieur gauche, mais il est possible de modifier le positionnement avec le paramètre android:layout_gravity.*
- ***RelativeLayout** : Ses enfants sont positionnés les uns par rapport aux autres, le premier enfant servant de référence aux autres. Ce gabarit a l'avantage de permettre un positionnement précis et intelligent tout en minimisant le nombre de vues utilisées.*
- ***TableLayout** : Permet de positionner vos vues en lignes et colonnes à l'instar d'un tableau.*
- ***Gallery** : Affiche une ligne unique d'objets dans une liste déroulante horizontale.*

LINEARLAYOUT

Le conteneur **LinearLayout** est un modèle reposant sur des boîtes : les widgets ou les conteneurs fils sont alignés en colonnes ou en lignes, les uns après les autres, exactement comme avec le gestionnaire de disposition **FlowLayout** de Swing.

*Pour configurer correctement un **LinearLayout**, vous pouvez agir sur les propriétés suivantes :*

- ***android:orientation*** *: uniquement deux valeurs sont possibles : **horizontal** pour une disposition suivant une ligne et **vertical** pour une disposition sous forme de colonne. Par défaut, c'est une présentation horizontale qui est proposée.*
- ***android:layout_width*** *et **android:layout_height** : Ces valeurs peuvent s'exprimer de trois façons différentes :*
  - *une dimension précise, comme **125px**, afin d'indiquer que le widget devra occuper exactement 125 pixels,*
  - ***wrap_content**, afin de demander que le widget occupe sa place naturelle sauf s'il est trop gros, auquel cas Android coupera le texte entre les mots pour qu'il puisse tenir,*
  - ***fill_parent**, afin de demander que le widget occupe tout l'espace disponible de son conteneur après placement des autres widgets.*
- ***android:layout_weight*** *: cette propriété permet de partager l'espace disponible entre plusieurs widgets sur la même ligne ou la même colonne. La valeur à spécifier indique la proportion d'espace libre qui sera affiché au widget. Si cette valeur est la même pour les deux widgets (1, par exemple), l'espace libre sera partagé équitablement entre eux. Si la valeur est 1 pour un widget et 2 pour l'autre, le second utilisera deux fois plus d'espace libre que le premier, etc. Le poids d'un widget est fixé à zéro par défaut.*

***ATTENTION** : Pour que cela fonctionne correctement avec l'attribut **android:layout_weight**, vous devez impérativement, avec une disposition en ligne initialisez à zéro les valeurs **android:layout_width** de tous les widgets du layout, et avec une disposition en colonne initialisez à zéro les valeurs **android:layout_height** de tous les widgets du layout.*

- ***android:layout_gravity*** *: par défaut, les widgets s'alignent à partir de la gauche et du haut. Si vous créez une ligne avec un LinearLayout horizontal, cette ligne commencera donc à se remplir à partir du bord gauche de l'écran. Cette propriété indique au widget et à son conteneur comment l'aligner par rapport à l'écran.*

  *Pour une colonne de widgets, les gravités les plus courantes sont **left**, **center_horizontal** et **right** pour respectivement, aligner les widgets à gauche, au centre ou à droite.*

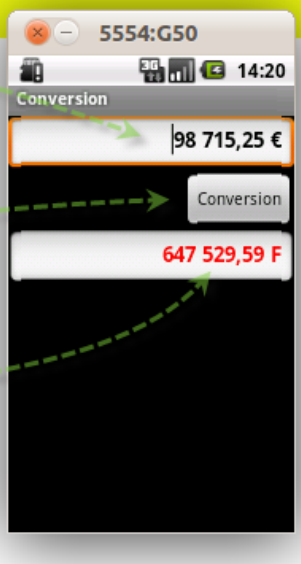
  *Pour une ligne, le comportement par défaut consiste à placer les widgets de sorte que leur texte soit aligné sur la ligne de base (la ligne invisible sur laquelle les lettres semblent reposer), mais il est possible de préciser une gravité **center_vertical** pour centrer verticalement les widgets dans la ligne.*
- ***android:padding*** *: les widgets sont, par défaut, serrés les uns contre les autres. Vous pouvez augmenter l'espace intercalaire à l'aide de cette propriété. La valeur de remplissage précise l'espace situé entre le contour de la cellule du widget et son contenu réel.*

*La propriété **android:padding** permet de préciser le même remplissage pour les quatre côtés du widget ; son contenu étant alors centré dans la zone qui reste. Pour utiliser des valeurs différents en fonction des côtés, utilisez plutôt les propriétés **android:paddingLeft**, **android:paddingRight**, **android:paddingTop** et **android:paddingBottom**.*

Afin de bien maîtriser ce type de conteneur, je vous propose de réaliser une activité qui permet, à partir d'une valeur en €uro, de la convertir en Franc.

Conversion.java × main.xml ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3    android:orientation="vertical"
4    android:layout_width="fill_parent"
5    android:layout_height="fill_parent"
6  >
7    <EditText
8      android:id="@+id/euro"
9      android:layout_width="fill_parent"
10     android:layout_height="wrap_content"
11     android:gravity="right"
12     android:text="0,00 €"
13     android:inputType="number|numberDecimal"
14     android:textStyle="bold"
15   />
16   <Button
17     android:layout_width="wrap_content"
18     android:layout_height="wrap_content"
19     android:layout_gravity="right"
20     android:text="Conversion"
21     android:onClick="calcul"
22   />
23   <EditText
24     android:id="@+id/franc"
25     android:layout_width="fill_parent"
26     android:layout_height="wrap_content"
27     android:text="0,00 F"
28     android:gravity="right"
29     android:editable="false"
30     android:textColor="#FF0000"
31     android:textStyle="bold"
32   />
33 </LinearLayout>
34
```



Conversion.java × main.xml ×

```
1  package fr.btsiris.conversion;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.view.View;
6  import android.widget.EditText;
7  import java.text.*;
8
9  public class Conversion extends Activity {
10   private EditText euro;
11   private EditText franc;
12   private NumberFormat enEuro = NumberFormat.getCurrencyInstance();
13   private DecimalFormat enFranc = new DecimalFormat("#,##0.00 F");
14   private final double TAUX = 6.55957;
15
16    @Override
17   public void onCreate(Bundle savedInstanceState) {
18      super.onCreate(savedInstanceState);
19      setContentView(R.layout.main);
20      euro = (EditText) findViewById(R.id.euro);
21      franc = (EditText) findViewById(R.id.franc);
22   }
23
24   public void calcul(View bouton) {
25    try {
26      Number valeur = (Number) enEuro.parse(euro.getText().toString());
27      euro.setText(enEuro.format(valeur.doubleValue()));
28      franc.setText(enFranc.format(valeur.doubleValue() * TAUX));
29    }
30    catch (ParseException ex) {  }
31   }
32 }
```

Une fois que le comportement souhaité correspond parfaitement à votre attente, il est tout à fait possible de modifier uniquement l'apparence de l'activité.

Comme la description de l'aspect visuel est totalement séparé du traitement en coulisse, codé en Java, il suffit d'intervenir uniquement sur le fichier de description ***main.xml****.*

Conversion.java × main.xml ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3    android:orientation="vertical"
4    android:layout_width="fill_parent"
5    android:layout_height="fill_parent"
6  >
7    <LinearLayout
8      android:layout_width="fill_parent"
9      android:layout_height="wrap_content"
10   >
11     <EditText
12       android:id="@+id/euro"
13       android:layout_width="0px"
14       android:layout_height="wrap_content"
15       android:gravity="right"
16       android:text="0,00 €"
17       android:inputType="number|numberDecimal"
18       android:textStyle="bold"
19       android:layout_weight="2"
20     />
21     <Button
22       android:layout_width="0px"
23       android:layout_height="wrap_content"
24       android:layout_gravity="right"
25       android:text="Franc"
26       android:onClick="calcul"
27       android:layout_weight="1"
28     />
29   </LinearLayout>
30   <EditText
31     android:id="@+id/franc"
32     android:layout_width="fill_parent"
33     android:layout_height="wrap_content"
34     android:text="0,00 F"
35     android:gravity="right"
36     android:editable="false"
37     android:textColor="#FF0000"
38     android:textStyle="bold"
39   />
40 </LinearLayout>
```