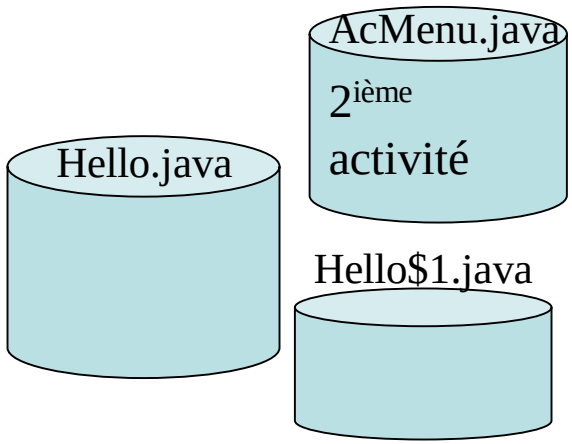
Synthèse: Développement et déploiement d'une application Android

2023 **Claude Vilvens / Christophe Charlet**

Activity de base

C:\android\adroid-sdk_rXX-windows\android-sdk-windows\tools\applics\hello

/src/my/apps/learning



Android crée des objets et des classes à la volée. En fait, on définit une classe qui n'a pas de nom. Exemple: dans les serveurs multithreads => new runnable=interface new runnable {} = classe sans nom instanciée sur le moment. Elle implémente runnable => pas Id de classe à gérer car anonyme.

```
package my.apps.learning;

public class Hello extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

L'objet instance est également à la volée => pas d'identifiant ⇒ Économie de place

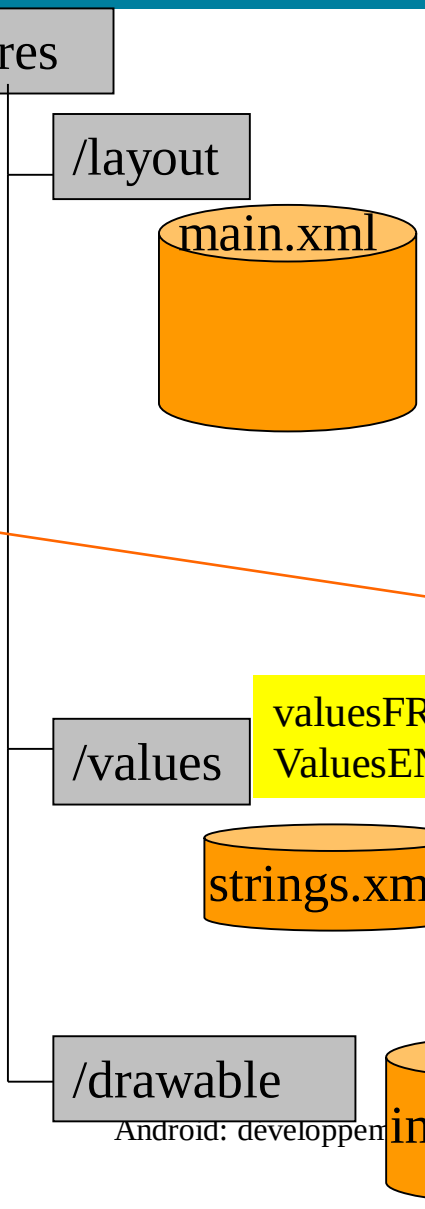
Création de fichier nom$x.class ne contenant chacun qu'une seule classe. (x est un numéro)

Pas de « new » ?! Pas de new ~~Jframe~~ view! Mais une référence vers un fichier XML

Ressources de l'activité

Partie:
- Graphique
- Chaînes de caractères
- (internationalisation)
- Images

Layout: objet dont le but est d'organiser les composants les uns par rapport aux autres => Assurer la portabilité: petits/moyens/grands écrans

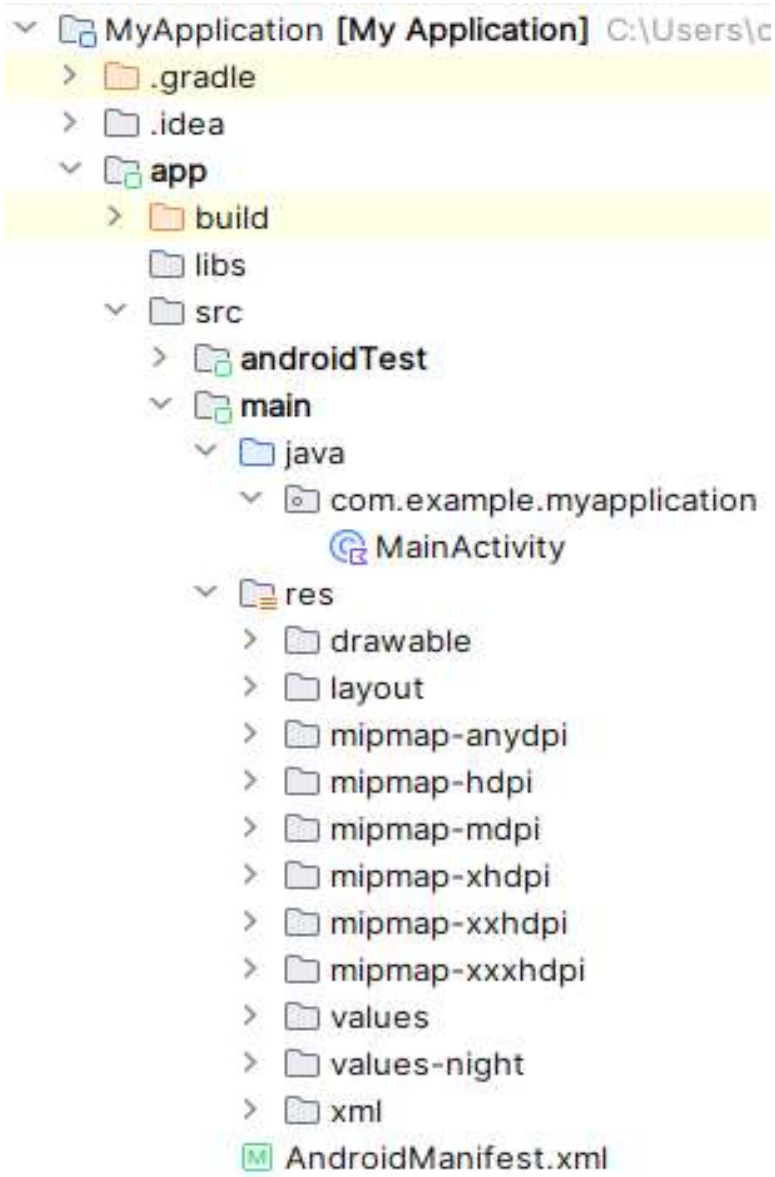



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout     TAG
xmlns:android="http://schemas.android.co
m/apk/res/android" ATTRIBUTS
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, hello"
    />
</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">hello</string>
</resources>
```


www.hepl.be

Exemple


MyApplication [My Application] C:\Users\c
.gradle
.idea
app
build
libs
src
androidTest
main
java
com.example.myapplication
MainActivity
res
drawable
layout
mipmap-anydpi
mipmap-hdpi
mipmap-mdpi
mipmap-xhdpi
mipmap-xxhdpi
mipmap-xxxhdpi
values
values-night
xml
AndroidManifest.xml

Lien entre les boutons, textView.. et le fichier XML

L'AndroidManifest de l'application


```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools">

    <application
            android:allowBackup="true"
            android:dataExtractionRules="@xml/data_extraction_rules"
            android:fullBackupContent="@xml/backup_rules"
            android:icon="@mipmap/ic_launcher"
            android:label="My Application"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/Theme.MyApplication"
            tools:targetApi="31">
        <activity
                android:name=".MainActivity"
                android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

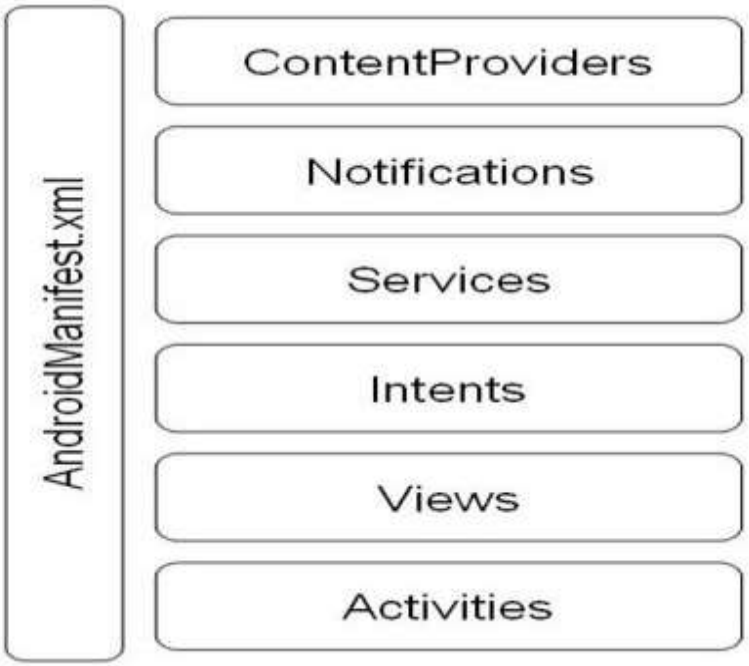
Plus riche qu'un simple manifest d'un .jar

9. Le développement en ligne de commande (12/31)

9.2 Le manifeste de l'application

Toute application Android doit posséder un fichier **AndroidManifest.xml** dans son répertoire racine. Son rôle est bien sûr de décrire l'application en termes de packages, classes, versions, permissions, etc.

Mais sa première raison d'être est de déclarer les composants utilisables par le système : **les composants qui n'y sont pas déclarés sont invisibles au système et ne pourront donc pas être exécutés**.



Seules les balises <**manifest**> et <**application**> sont requises.

Le développement en ligne de commande

AndroidManifest.xml

L'activity manager envoie un msg (INTENT) à la 1ere activité (LAUNCHER)

A remarquer que le nom de classe précédé d'un point (".MainActivity") est un raccourci pour signifier que ce nom doit être préfixé du nom du package.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.c
        lowBackup="true"
        taExtractionRules="@xml/data_extraction_rules"
        llBackupContent="@xml/backup_rules"
        on="@mipmap/ic_launcher"
10          android:label="@string/app_name"
11          android:roundIcon="@mipmap/ic_launcher_round"
12          android:supportsRtl="true"
13          android:theme="@style/Theme.MyApplication"
14          tools:targetApi="31">
15      <activity
16              android:name=".MainActivity"
17              android:exported="true">
18          <intent-filter>
19              <action android:name="android.intent.action.MAIN" />
21              <category android:name="android.intent.category.LAUNCHER" />
22          </intent-filter>
```

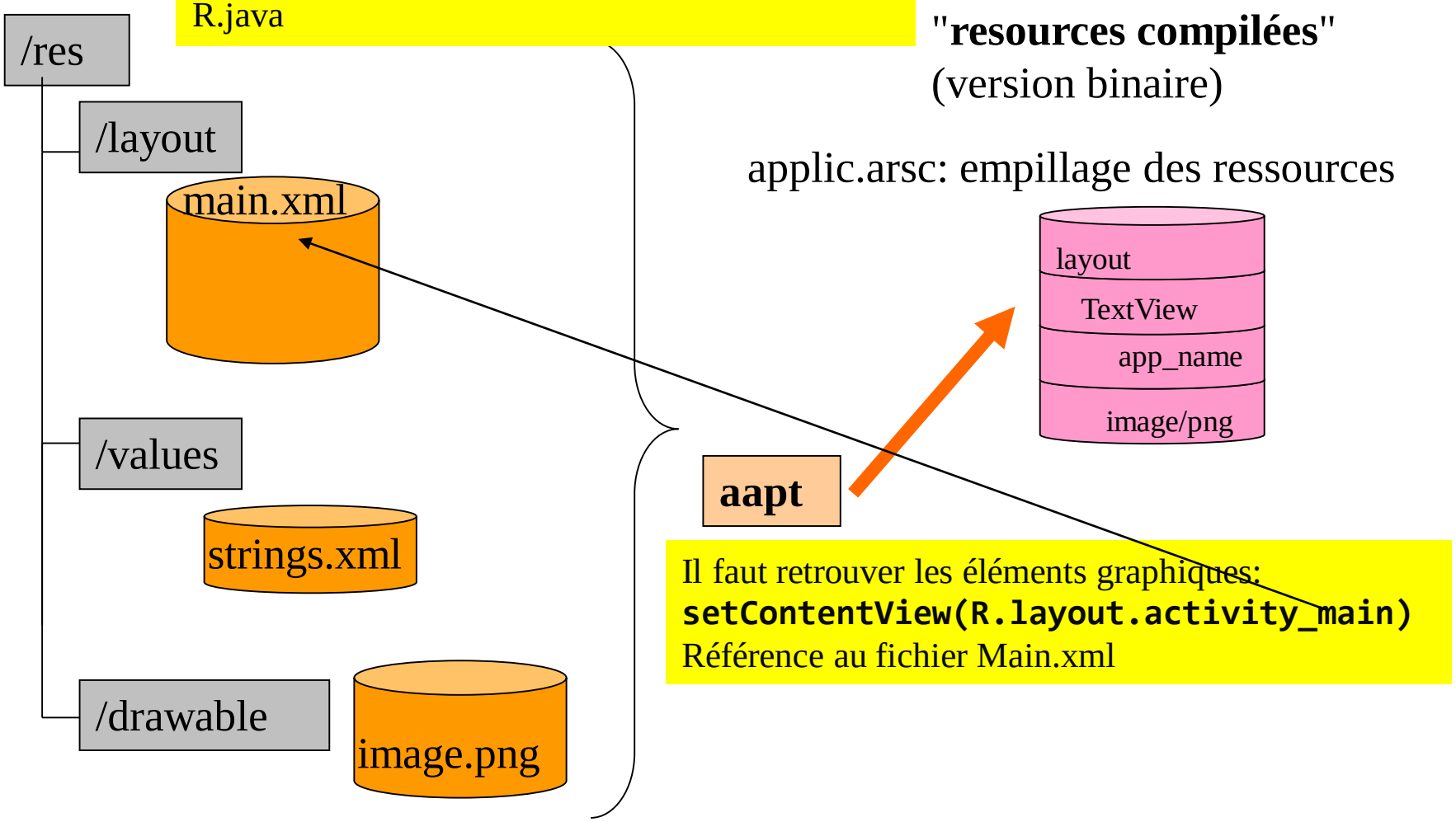
référence à "app_name" dans **res\values\strings.xml.** Le @ signifie, aller voir dans les ressources.

Peut contenir plusieurs TAGs activity, Une activité non déclarée n'est pas utilisable

A remarquer : la définition de l'**intent** = il spécifie qu'**il faut lancer** (LAUNCHER) **l'activité « MainActivity"** en tant que **composant initial** (MAIN). Chaque activité doit avoir un Filter. On peut ajouter aussi des permissions (envoyer SMS, utiliser Bluetooth, reseau (socket), …

es" (1)

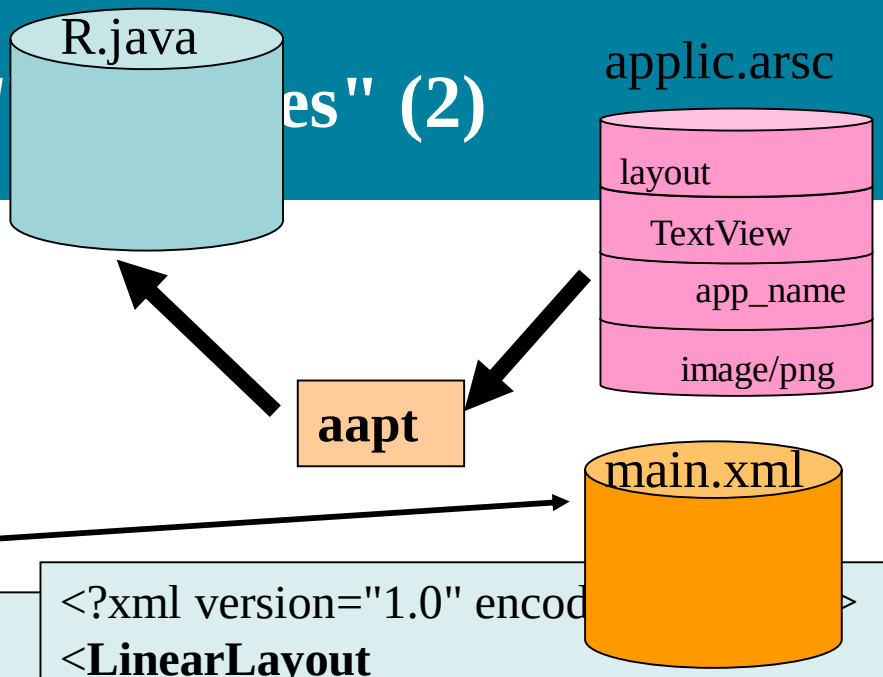
Parser (SAX = parser/DOM=builder) un fichier **prend du temps** => Trouver un système de ressources plus efficace.
L'aapt (Android Asset Packaging Tool) fabrique les ressources compilées + création d'un fichier R.java



es" (2)

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.…
 */
package my.apps.learning;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
}
```

```
public class hello extends Activity
{
public void onCreate(Bundle savedInstanceState
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```



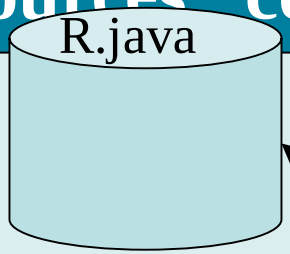

```
<?xml version="1.0" encod
<LinearLayout
xmlns:android="http://schemas.android.co
m/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, hello"
    />
</LinearLayout>
```

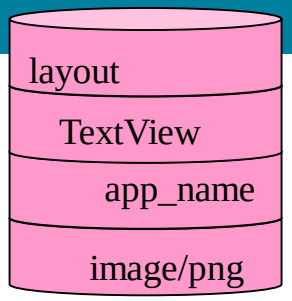
Fichiers R et .arsc

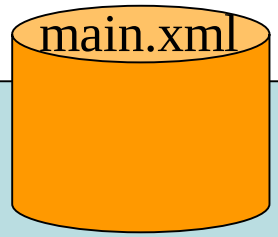
- **R sert de pont entre les fichiers xml et le code applicatif Java** en étant constitué de toute une série de sous-classes statiques (en fait, une par type de composant) contenant les identifiants des diverses ressources sous formes de variables de classe (comme "app_name" ou "texte_horloge").

- Le fichier **resources.arsc** contenant ces informations sous forme binaire, dans un format permettant un parcours efficace lors de l'exécution de l'application.

Les ressources "compilées" (3)








```
public final class R {
    …
    public static final class id {
        …
        public static final int texte_horloge=0x7f060002;
         …
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
   …
```

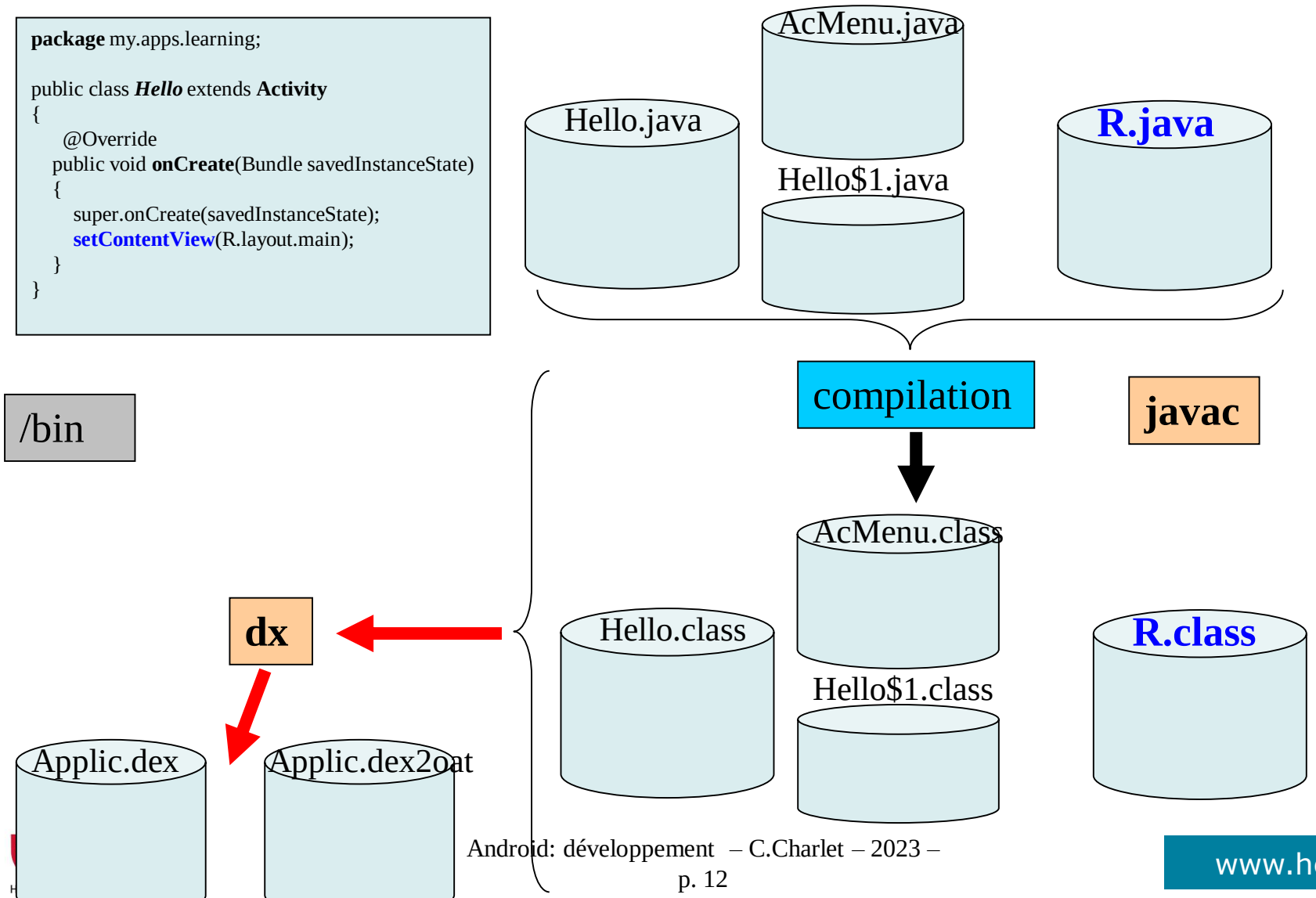
```
…
<TextView
  android:id="@+id/texte_horloge"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Heure de Bruxelles"
/>
```

```
public class hello extends Activity
{
    public void onCreate(Bundle savedInstanceSta
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView texte = (TextView)findViewById(R.id. texte_horloge);
texte.setText("All you need is Java");
     }
}
```

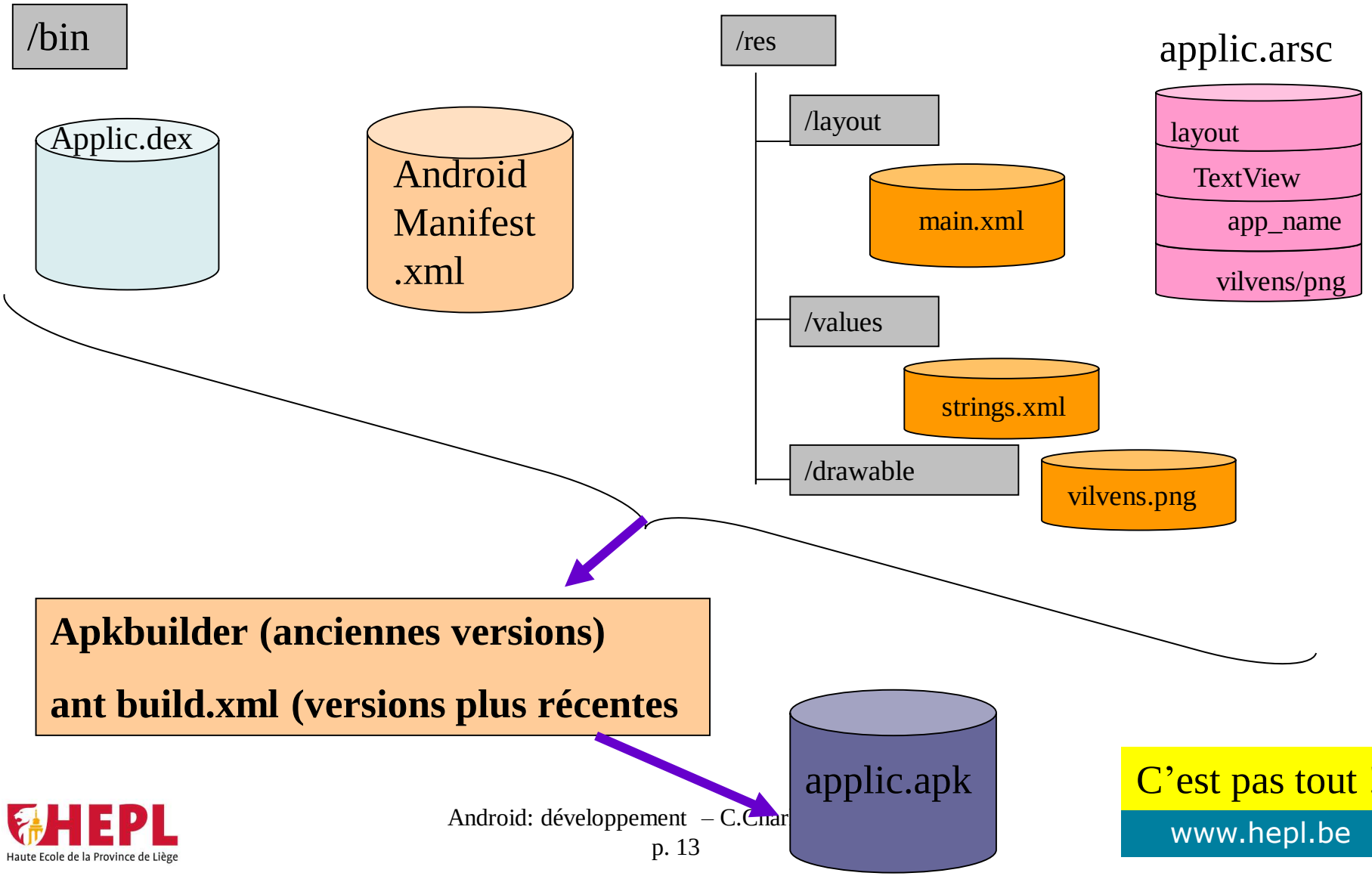
La compilation pour Dalvik/Art

```
package my.apps.learning;

public class Hello extends Activity
{
    @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
}
```



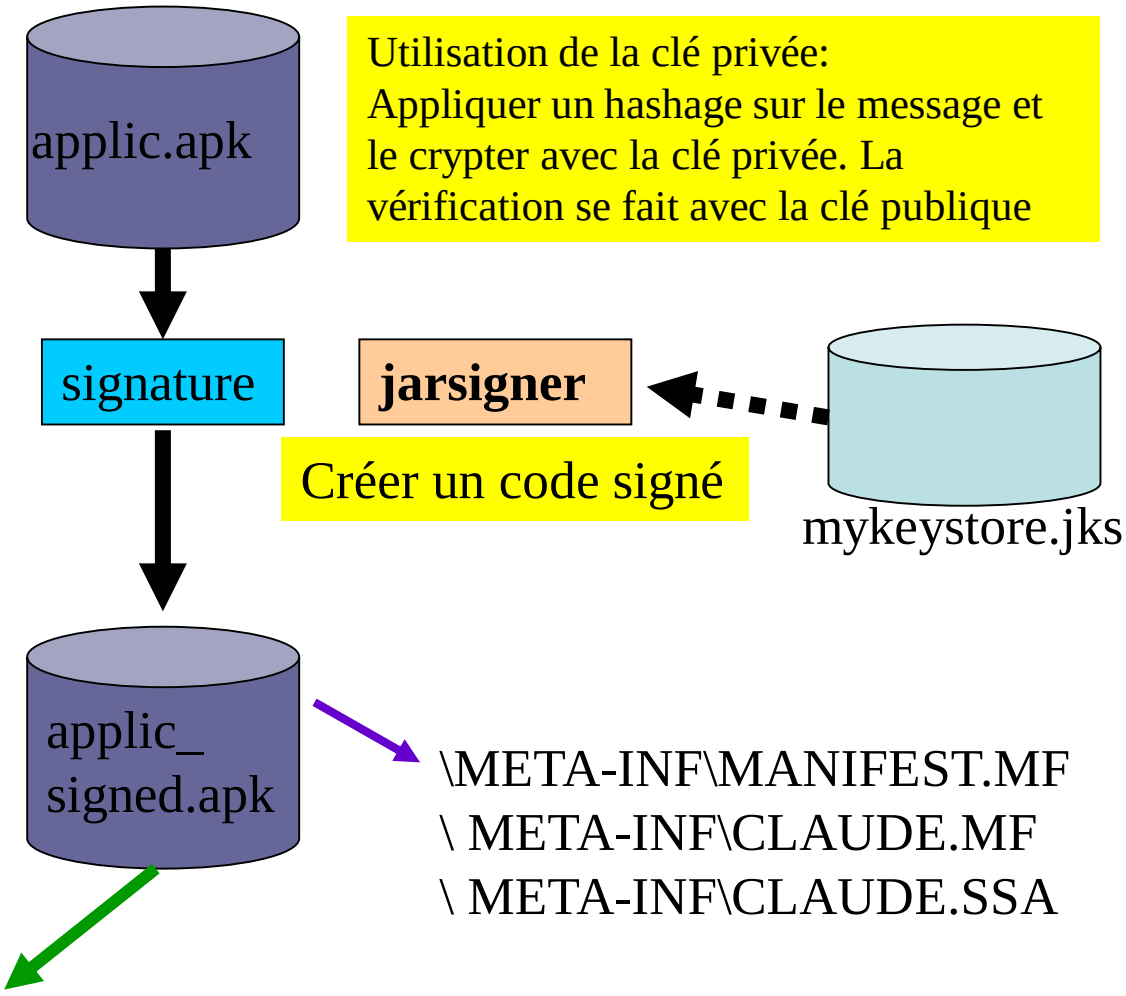
La construction de l'apk



C'est pas tout !

La signature de l'apk

/bin



La clé privée est dans le keystore
Il y a des entries => il faut donc spécifier l'entrée (label)

Fichiers supplémentaires (2) en + du manifest (.MF):
.SF et .DSA (claude : car c'est l'entrée)

.MF: ~~On a ajouté~~ dans le manifest la valeur des digests des fichiers (dans le .dex), il y a autant de lignes que de fichiers .class du dex => **.SF = fichier signature**, il contient le digest de chaques lignes du fichier manifest

La signature fabriquée par le jarsigner est sauvée dans le fichier **.DSA**
DSA est plus rapide que RSA pour la vérification.

Signons le jar avec notre clé privée se trouvant dans le keystore :

```
D:\java-forte-application\AppletSignee>jarsigner -keystore c:\vilvens\.keystore
AppletSigneeFichierReseau.jar claude
Enter Passphrase for keystore: beaugosse
Enter key password for claude: genius
```

Le jar a bien été modifié :

```
D:\java-forte-application\AppletSignee>jar tf AppletSigneeFichierReseau.jar
META-INF/MANIFEST.MF
META-INF/CLAUDE.SF
META-INF/CLAUDE.DSA
AppletSignee.class
AppletSignee$1.class
```

Dans le répertoire META-INF, on trouve à présent trois fichiers.

1) Le fichier **MF** (ManiFest) contient à présent la valeur des digests des fichiers class cités :

```
Name: AppletSignee$1.class
SHA1-Digest: CxmtcCTIdVofQW5K3sgwBrB4kLk=

Name: AppletSignee.class
SHA1-Digest: pjBOk6rXkaIK2sin1plgYGBSf7E=
```

2) Le fichier **SF** est le fichier de signature (Signature File – **SF**); il contient la valeur de hachage du manifeste ainsi que, pour chaque fichier, la valeur de hachage des lignes qui lui sont associées dans le manifeste :

```
Signature-Version: 1.0
SHA1-Digest-Manifest: SWsC7DuD+/e8AEQr0rndJ1ZNPxU=
Created-By: 1.3.1_02 (Sun Microsystems Inc.)

Name: AppletSignee$1.class
SHA1-Digest: R5JDSBzvfnb6pHbW1Xl7HxddDis=

Name: AppletSignee.class
SHA1-Digest: Is/9qJzXwlJnpLiI9s0X4+RN5VA=
```

3) Le fichier **DSA** (Digital Signature Algorithm – **DSA**) contient la signature du fichier .SF et, sous forme codée, le certificat (ou la chaîne de certificats) du signataire.

Le développement en ligne de commande

Installer l'application au moyen de l'outil **adb** (**A**ndroid **D**ebug **B**ridge) :

adb -d install -r <fichier apk>

Accessoirement, on peut visualiser le contenu du fichier apk avec l'outil aapt (ou winzip par exemple):

```
C:\java-android-application\Coucou\bin>aapt list -v Coucou.apk
Archive:  Coucou.apk
 Length  Method    Size  Ratio   Date   Time       CRC-32    Name
--------     ------       ------- -----             ----     ----         ------       ----
    2200    Stored     2200   0%    08-26-10 20:34  99a4f90b  res/drawable/icon.png
     640    Deflate    277    57%   08-26-10 20:34  8ae7db9b  res/layout/main.xml
    1424    Deflate    555    61%   08-26-10 20:34  17fb42dc  AndroidManifest.xml
    1060    Stored     1060   0%    08-26-10 20:34  4e2f08e7  resources.arsc
    1900    Deflate    932    51%   08-26-10 20:34  ad5379e5  classes.dex
     401    Deflate    267    33%   08-26-10 20:34  e7e2db67  META-INF/MANIFEST.MF
     454    Deflate    299    34%   08-26-10 20:34  bf3ab414  META-INF/CERT.SF
     776    Deflate    603    22%   08-26-10 20:34  8d19d2aa  META-INF/CERT.RSA
--------                 -------  ---                              -------
    8855                 6193  30%                                8 files
```

9. Le développement en ligne de commande

En synthèse :