Projet Android

À la création d'un *projet Android* (IDE AndroidStudio) tout un ensemble de répertoires et de fichiers sont engendrés. On en distingue 3 :

1. un fichier `AndroidManifest.xml` (dans le répertoire `manifests`) qui donne au système Android les caractéristiques de votre application ; dont, en particulier, la liste de ses *activités*.
2. un répertoire `java` où sont placés les codes JAVA de votre application.
3. une répertoire `res` contenant d'autres répertoire, dont `layout`. Dans ces répertoires sont placés des fichiers, au format XML, de description des ressources des interfaces graphiques de l'application.

XML et JAVA

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
 xmlns:android="http://schemas.android.com/apk/res/android"
 package="android.appdroid">
 <application
  android:icon="@mipmap/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme">
  <activity android:name=".MainActivity">
   <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
  </activity>
 </application>
</manifest>
```

MainActivity.java

Classe JAVA engendrée par défaut à la création d'un projet

```
package android.appdroid;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

}
```

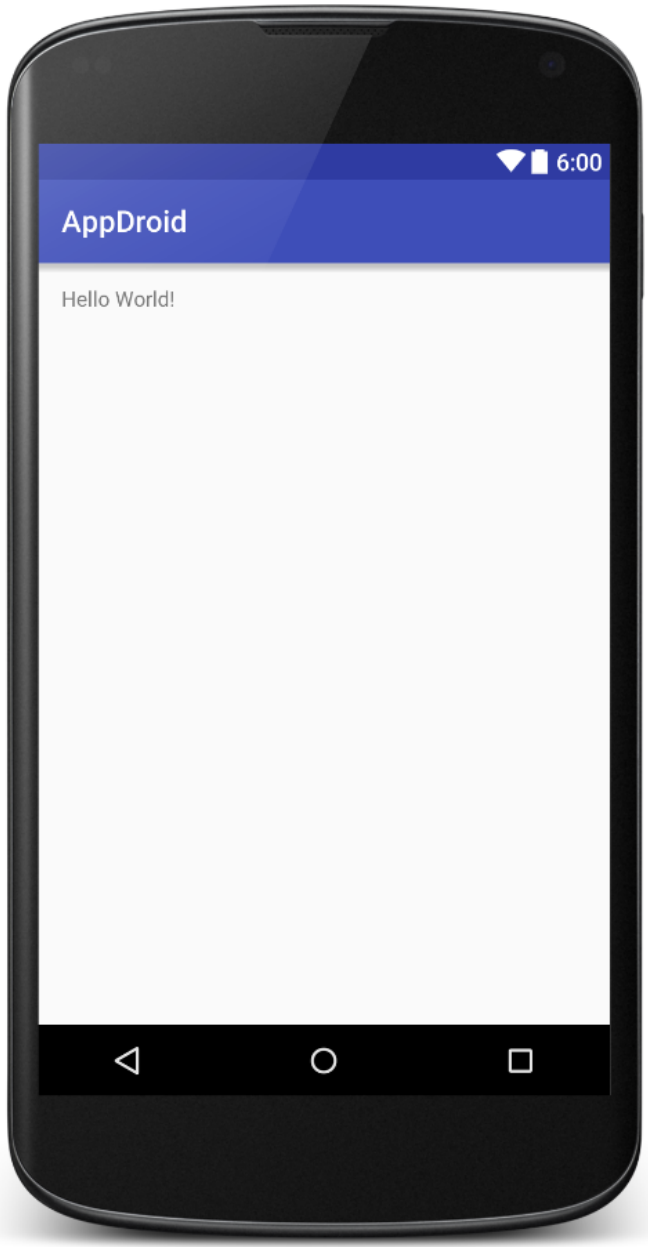
res/layout/activity_main.xml

Fichier XML engendré à la création du projet

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="android.appdroid.MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />
</RelativeLayout>
```

Visualisation sur un terminal



Soigner la présentation

- modifier le contenu du message
- centrer le message
- modifier la taille

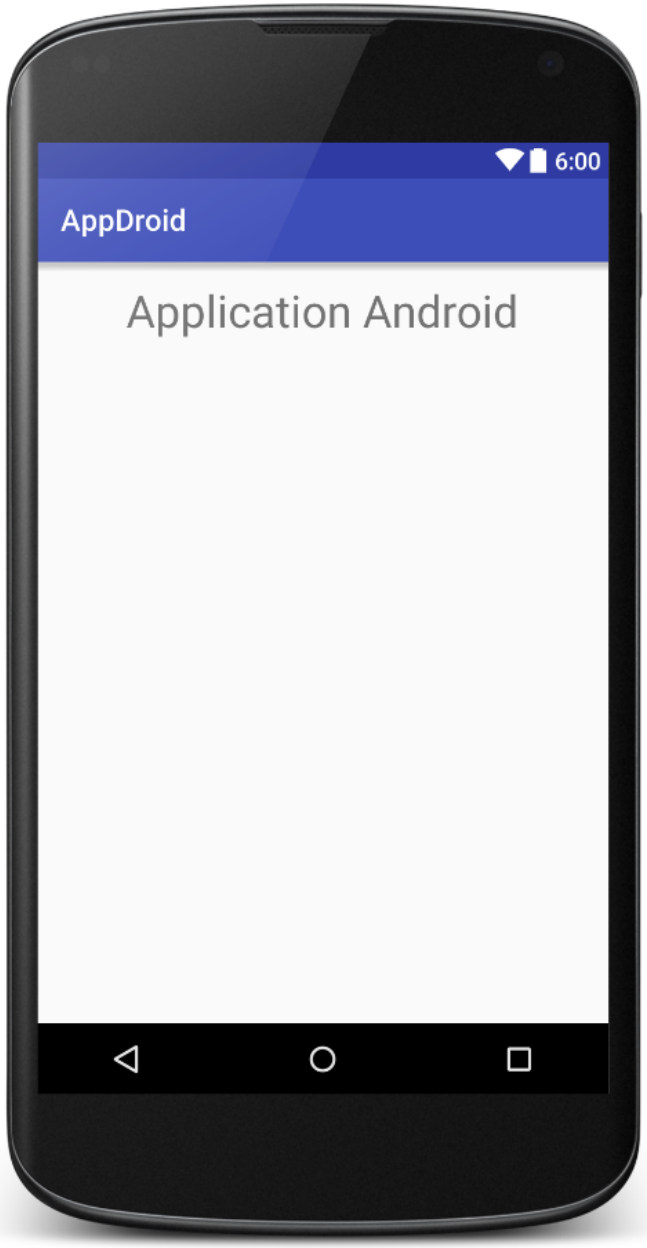
Editer `res/layout/activity_main.xml`, balise `TextView`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout [...]>

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30dp"
        android:layout_centerHorizontal="true"
        android:text="Application Android" />
</RelativeLayout>
```

\+ nommage du composant (`android:id`)

Visualisation

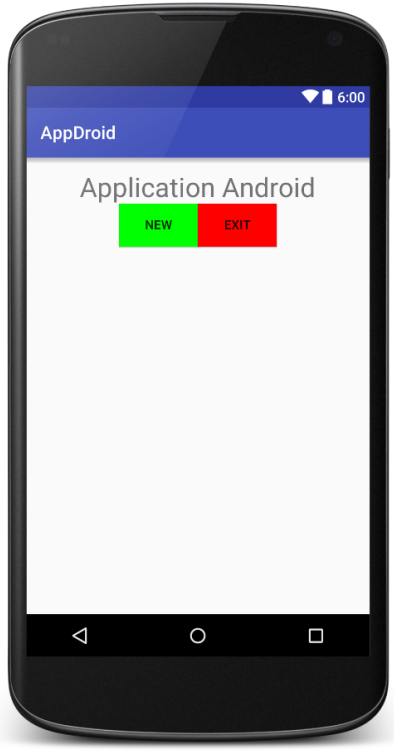


Ajouter des composants

Deux boutons : *NEW* et *EXIT*.
Méthode :

1. Ajouter une nouvelle «boîte» (un *layout*), sous le texte.
2. Y placer 2 boutons côte à côte.

Pratique :



- utiliser le mode *design* de l'IDE ;
- ajuster en mode *text* le XML engendré.

activity_main.txt

1. Ajouter une nouvelle «boîte»

```
<RelativeLayout [..] >
    <TextView
        android:id="@+id/title" [..] />
    <LinearLayout
        android:id="@+id/buttonsLayout"
        android:layout_below="@+id/title"
        android:orientation="horizontal"
        android:layout_centerHorizontal="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        [..]

    </LinearLayout>

</RelativeLayout>
```

Les boutons

2. Y placer 2 boutons

Widget button : classe/balise `Button`

```
<LinearLayout
  android:id="@+id/buttonsLayout" [..] >
   <Button
     android:id="@+id/buttonExit"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:background="#f00"
     android:text="EXIT"
     android:onClick="onClickExit" />
  <Button
     android:id="@+id/buttonNew" [..] />
</LinearLayout>
```

Comportement : `android:onClick`

(Ré)action associée au bouton

JAVA - XML

Dans la classe `MainActivity` : méthode `onClickExit` [1]

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        [..]
    }

    public void onClickExit(View view) {
        finish();
    }
}
```

Classe `View` : mère de toutes les interfaces graphiques

1. «new» plus tard

Une aire de jeu

Surface de dessin réactive : `SurfaceView` (extends View)

Personnalisée :
`GameView extends SurfaceView`
Contenu graphique (dessin) dynamique :
`implements SurfaceHolder`

1. définir `GameView` : classe JAVA
2. ajouter à l'interface graphique : balise dans (`activity_main.xml`)

Classe héritère de `View` $\Rightarrow$ une nouvelle balise

GameView : XML

Ajouter

```
<android.appdroid.GameView
    android:id="@+id/gameView"
    android:layout_below="@+id/buttonsLayout"
    android:layout_centerHorizontal="true"
    android:layout_margin="10dp"
    android:layout_width="500dp"
    android:layout_height="50dp" />
```

après (`android:layout_below`) les boutons
(`"@+id/buttonsLayout"`)
NOTA : taille fixe

GameView : JAVA

Signatures

```
public class GameView extends SurfaceView
                      implements SurfaceHolder.Callback {

public GameView(Context context, AttributeSet attrs)

void tryDrawing()

@Override
public void surfaceCreated(SurfaceHolder sh)
@Override
public void
   surfaceChanged(SurfaceHolder sh, int f, int w, int h)
@Override
public void surfaceDestroyed(SurfaceHolder sh)

@Override
public void onDraw(Canvas c)

}
```

GameView : le constructeur

```
public GameView(Context context, AttributeSet attrs) {
    super(context, attrs);
    getHolder().addCallback(this);
}
```

`getHolder` donne le «détenteur» de la surface
`addCallback` lui signifie qu'il peut adresser des messages à la surface aux moments clés de sa vie : création, changement, destruction

GameView : dessiner

Concurrence, section critique

```
void tryDrawing() {
    SurfaceHolder h = this.getHolder();
    Canvas c = h.lockCanvas();
    if (c != null) {
        this.onDraw(c);
        h.unlockCanvasAndPost(c);
    }
}
```

Le dessin proprement dit

```
@Override
 public void onDraw(Canvas c) {
    c.drawColor(Color.BLUE);
 }
```

GameView : les *«call back»*

Synchronisation : se dessiner au bon moment

```
@Override
public void surfaceCreated(SurfaceHolder sh) {
   tryDrawing();
}
@Override
public void
  surfaceChanged(SurfaceHolder sh, int f, int w, int h) {
   // rien
}
@Override
public void surfaceDestroyed(SurfaceHolder sh) {
   // rien
}
```

Synchroniser le dessin avec l'état du jeu

Un jeu idiot : la couleur change de manière aléatoire chaque fois que l'on touche la surface
Le modèle du jeu

```
public class GameModel {
  int color;
  Random ran;

  GameModel() { color = Color. BLUE; ran = new Random(); }
  void resetColor() { color = Color. BLUE; }
  void changeColor() {
    color = Color.rgb(ran.nextInt(255),
                      ran.nextInt(255),
                      ran.nextInt(255));
  }
  int getColor() { return color; }
}
```

Modèle persistant

Le modèle est détenu par *l'application*

$\Rightarrow$ personnaliser l'application Android :

```
public class GameApplication extends Application {

    GameModel theGame;

    @Override
    public void onCreate() {
        super.onCreate();
        theGame = new GameModel();
    }

    GameModel getGame() {
            return theGame;
    }


}
```

Manifester son application

Déclarer dans `AndroidManifest`

```
<manifest [..] >

    <application
        android:name="GameApplication"
        [..] >
        [..]
    </application>

</manifest>
```

Partager

Accéder à l'application depuis une activité :

```
public class MainActivity extends AppCompatActivity {
  GameApplication app;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    [..]
    app = (GameApplication)this.getApplication();
  }
```

Depuis un composant graphique de l'activité

```
public class GameView extends SurfaceView
                      implements SurfaceHolder.Callback {
  GameApplication app;
  public GameView(Context context, AttributeSet attrs) {
    [..]
    app = (GameApplication) (context.getApplicationContext());
  }
```

GameView : dessiner l'état du jeu

La couleur est donnée par le modèle

```
@Override
 public void onDraw(Canvas c) {

      c.drawColor(app.getGame().getColor());

 }
```

GameView : réagir au toucher (1)

Redéfinir

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN: {
            app.getGame().changeColor();
            tryDrawing();
            return true;
        }
        case MotionEvent.ACTION_MOVE: {
            return false;
        }
        case MotionEvent.ACTION_UP: {
            return false;
        }
        default:
            return false;
    }
}
```

Le bouton NEW

Réinitialiser l'état du jeu

XML (activity_main.xml)

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#0f0"
    android:text="NEW"
    android:onClick="onClickNew"
    android:id="@+id/buttonNew" />
```

JAVA (MainACtivity.java)

```
public void onClickNew(View view) {
    app.getGame().resetColor();
    ((GameView)findViewById(R.id.gameView)).tryDrawing();
}
```