Projet Android

À la création d'un *projet Android* (IDE AndroidStudio – *package* `ue2i013.appdroid` – thème empty) 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. un répertoire `res` contenant d'autres répertoires, 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="ue2i013.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 ue2i013.appdroid;

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

public class MainActivity extends Activity {

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

}
```

On ne définit ni n'utilise de constructeur d'activité

Nota Bene : oublier `AppCompatActivity` pour `Activity`

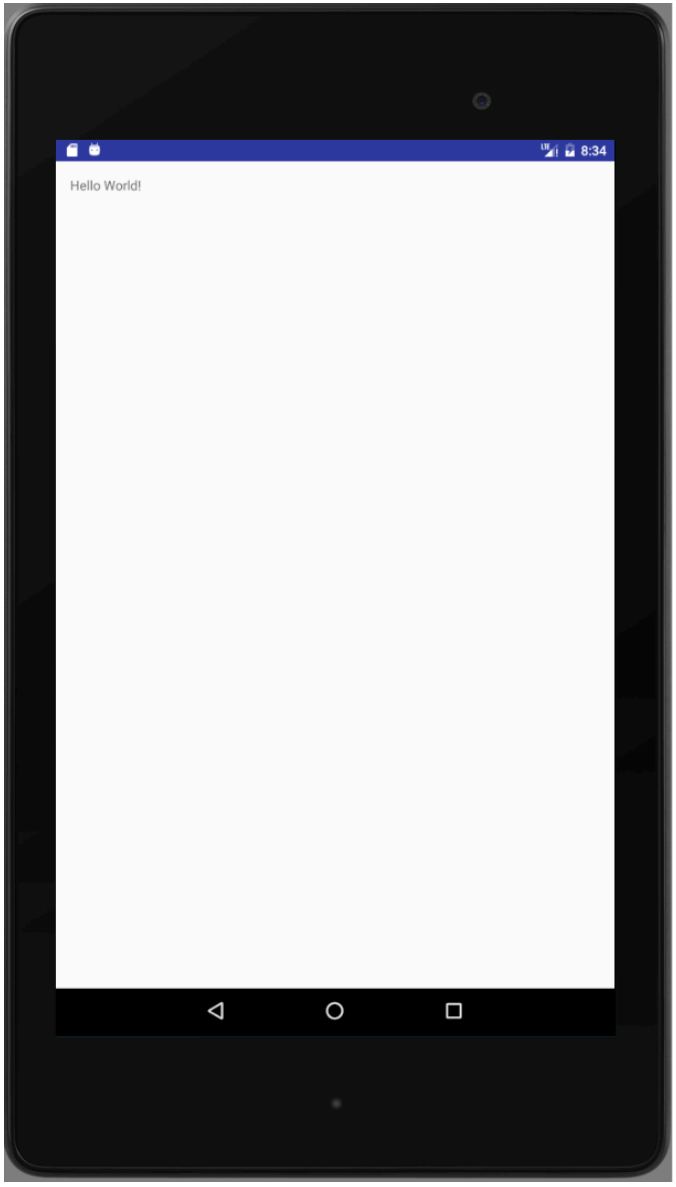
res/layout/activity_main.xml

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:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="ue2i0013.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

- center les éléments graphiques
- modifier la taille
- modifier le contenu du message

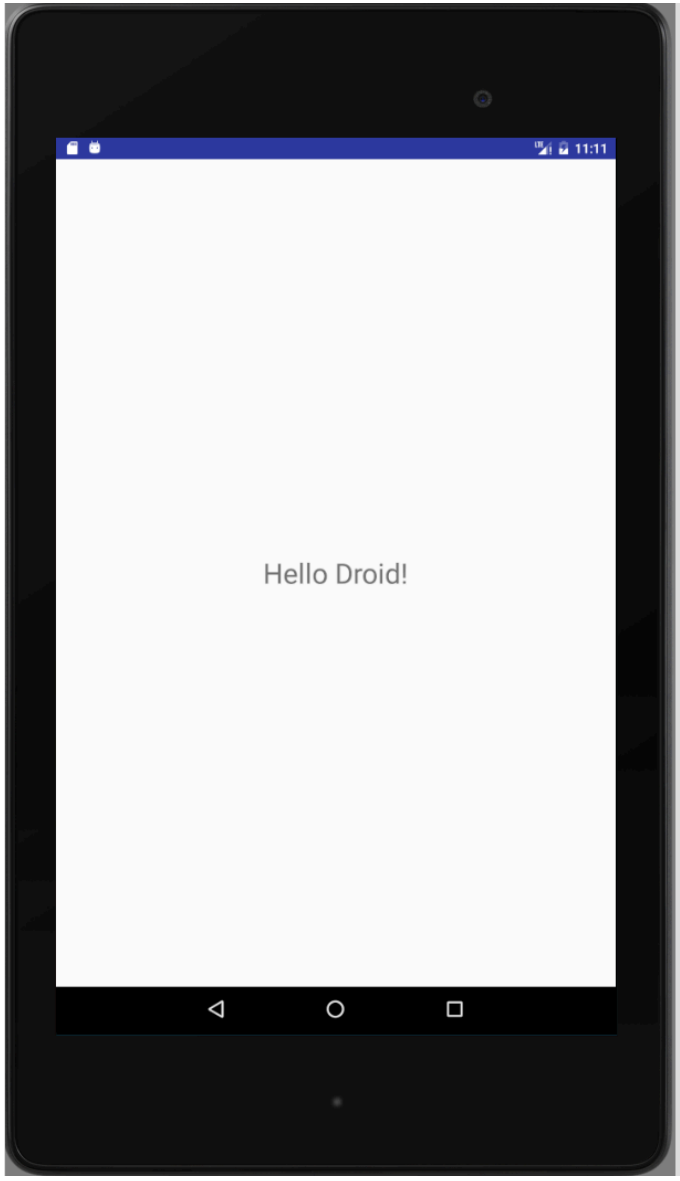
Editer `res/layout/activity_main.xml`, balise `TextView`

```
<RelativeLayout
    [...]
    android:gravity="center"
    tools:context="ue2i0013.appdroid.MainActivity">

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30dp"
        android:text="Hello Droid!" />
</RelativeLayout>
```

NOTA BENE : nommage du composant (`android:id`)

Visualisation



Ajouter un composant

Un *bouton EXIT* placé en colonne sous le titre pour quitter l'application.

1. ajouter le composant graphique à l'interface : modifier `activity_main.xml`.
2. définir l'action associée à l'activation du bouton : modifier `MainActivity`.

activity_main.xml

```
<Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@id/title"
      android:background="#ff0000"
      android:textSize="30dp"
      android:text="EXIT"
      android:onClick="onClickExit"/>
```

- `RelativeLayout` : placement relatif
  `layout_below="@id/title"`
- couleur de fond rouge : `background="#ff0000"` (RGB)
- action : `onClick=onClickExit` (c.f. classe `MainActivity`)

Visualisation

MainActivity

JAVA - XML

Dans la classe `MainActivity` : définir la méthode `onClickExit`

```
public class MainActivity extends Activity {

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

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

Paramètre `View v` : origine de l'invocation.

Une application : 2 activités

Ajouter une deuxième *activité* de jeu à l'application.
L'activité de jeu est lancée par un bouton depuis l'activité principale

1. déclarer l'activité comme ressource de l'application : modifier `AndroidManifest.xml`
2. ajouter un bouton `PLAY` à l'activité principale :
   2.1 modifier `activity_main.xml` : nouveau `Button`
   2.2 modifier `MainActivity` : nouvelle méthode `onClickPlay`
3. créer l'interface graphique de l'activité de jeu : nouveau fichier `activity_play.xml`
4. créer la classe `PlayActivity`

AndroidManifest

```
<manifest [..]>

    <application
        [..] >
        <activity android:name=".MainActivity">
            [..]
        </activity>
        <activity android:name=".PlayActivity" />
    </application>

</manifest>
```

Ajouter le bouton PLAY

Dans `activity_main.xml`

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/playButton"
    android:layout_below="@id/title"
    android:textSize="30dp"
    android:text="PLAY"
    android:onClick="onClickPlay"/>
```

Nota : modifier le `layout_below` du bouton `EXIT` (valeur : `@id/playButton`)

Dans `MainActivity` pour démarrer l'activité de jeu

```
public void onClickPlay(View v) {
    Intent playIntent = new Intent(this, PlayActivity.class);
    startActivity(playIntent);
}
```

Activité de jeu

Interface graphique : créer `res/layout/activity_play.xml`

```
<LinearLayout [..] >
    <Button
        [..]
        android:onClick="onClickQuit"/>
</LinearLayout>
```

Nota : une autre classe de *layout*

L'activité : créer `PlayActivity.java`

```
public class PlayActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_play);
    }
    public void onClickQuit(View v) {
        finish();
    }
}
```

Une aire de jeu

Surface de dessin réactive

JAVA+XML

1. définir une nouvelle classe JAVA : `GameBoard`
   - qui hérite de `SurfaceView` pour le dessin et l'interactivité ;
   - et qui implémente l'interface `SurfaceHolder.Callback` pour le contrôle de l'affichage.
2. intégrer la surface de jeu à l'interface graphique de l'activité
   - une nouvelle balise XML dans `activity_play`

Un (premier) jeu idiot : afficher un point (petit cercle) au touché de la surface

GameBoard : JAVA

Schéma

```
public class GameBoard extends SurfaceView
                        implements SurfaceHolder.Callback {
    public GameBoard(Context c) { [..] }
    public GameBoard(Context c, AttributeSet as) { [..] }
    public void reDraw() { [..] }
    @Override
    public void onDraw(Canvas c) { [..] }
    @Override
    public void surfaceCreated(SurfaceHolder holder) { [..] }
    @Override
    public void surfaceChanged
      (SurfaceHolder holder, int format, int width, int height)
      { [..] }
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) { [..] }
    @Override
    public boolean onTouchEvent(MotionEvent event) { [..] }
}
```

GameBoard : XML

Ajouter à `activity_play.xml`

```
<LinearLayout [..] >

<ue2i013.appdroid.GameBoard
    android:layout_width="match_parent"
    android:layout_height="256dp"
    android:id="@+id/boardSurface"
    />

    <Button [..] />

</LinearLayout>
```

- nouvelle balise : package+classe
- ATTENTION à la taille fixe : il faudra faire mieux...

GameBoard : JAVA

Les constructeurs

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

`getHolder` donne le contrôleur de la surface
`addCallback(this)` lui signifie qu'il peut adresser des messages à la surface elle-même aux moments clés de sa vie : création, changement, destruction

Pourquoi 2 constructeurs ? Mystère ...

GameBoard : JAVA

(Re)dessiner

Concurrence, section critique

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

Le dessin proprement dit : un simple fond gris (pour l'instant)

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

GameBoard : JAVA

les *«call back»*

Synchronisation : se dessiner au bon moment

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

`SurfaceHolder.Callback` : `surfaceChanged` n'est invoquée que lorsque la surface de dessin (*canvas*) est effective.
Nota : on connaît alors ses dimensions (`w` et `h`)

Un jeu et son interface

Un jeu idiot : afficher un point à chaque endroit touché
Une réalisation plus délicate :

- mémoriser l'ensemble des points touchés
- garder cette information persistante
- partager cette information entre différentes composantes ou méthodes

Une solution : un modèle de jeu logé au niveau de *l'application* qui peut être partagée par l'ensemble de composants de l'application (activités et composants graphiques)

Le modèle

Le bête modèle du jeu idiot.

```
public class Model {
    ArrayList<Position> xys;
    Model() {
        xys = new ArrayList<Position>();
    }
    void add(int x, int y) {
        xys.add(new Position(x,y));
    }
    ListIterator<Position> getAll() {
        return xys.listIterator();
    }
}
```

avec

```
public class Position {
    int x, y;
    Position(int x, int y) { this.x = x; this.y = y; }
    Integer getX() { return x; }
    Integer getY() { return y; }

}
```

Partage et persistance du modèle

Le modèle est détenu par *l'application*

⇒ personnaliser la classe `Application` d'Android :

```
public class TheApplication extends Application {
    Model m;
    @Override
    public void onCreate() {
        super.onCreate();
        m = new Model();
    }
    Model getModel() {
            return m;
    }
}
```

Déclarer (nommer) l'application au système : `AndroidManifest`

```
<manifest [..] >
    <application android:name="TheApplication" [..] >
        [..]
    </application>
</manifest>
```

Partager

Accéder à l'application depuis une activité :

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

Depuis un composant graphique de l'activité

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

Dessiner l'état du jeu

Affiner onDraw dans GameBoard

```
@Override
public void onDraw(Canvas c) {
    Model m = app.getModel();
    Paint p = new Paint();
    ListIterator<Position> it = m.getAll();
    c.drawColor(Color.LTGRAY);
    while(it.hasNext()) {
        Position xy = it.next();
        p.setColor(Color.DKGRAY);
        c.drawCircle(xy.getX(), xy.getY(),13,p); // ATTENTIO
    }
}
```

Notez l'utilisation de l'itérateur

Réagir au toucher

1. notifier au modèle
2. notifier au dessin

Dans `GameBoard` (re)définir `onTouchEvent`

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    int x = (int) event.getX();
    int y = (int) event.getY();
    int action = event.getAction();
    switch (action) {
        case MotionEvent.ACTION_DOWN: {
            app.getModel().add(x,y);
            reDraw();
            return true;
        }
        default:
            return false;
    }
}
```

Jouer encore

Ajouter à l'interface de jeu la possibilité de réinitialiser l'état du jeu.

Réinitialiser l'état du jeu (`Model`)

```
void reset() { xys.clear() }
```

Un bouton *reset* dans l'interface (`activity_play.xml`)

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="RESET"
    android:onClick="onClickReset"
    />
```

Réagir à la demande de réinitialisation (`PlayActivity.java`)

```
public void onClickReset(View v) {
   app.getMoel().reset();
   ((GameBoard)findViewById(R.id.boardSurface)).reDraw();
}
```

Notez `findViewById`