Android

Table des matières

[Installer Android studio 4](#_Toc472632709)

[Configurer SDK manager : 4](#_Toc472632710)

[Gradle : 4](#_Toc472632711)

[Menu : 4](#_Toc472632712)

[Mode développeur : 5](#_Toc472632713)

[Première interface 5](#_Toc472632714)

[Les ressources : 5](#_Toc472632715)

[Images 5](#_Toc472632716)

[Texte : 5](#_Toc472632717)

[Android à sa création : 5](#_Toc472632718)

[Unité : 6](#_Toc472632719)

[Layout : 6](#_Toc472632720)

[Les layouts les plus utilisés : 6](#_Toc472632721)

[Découpez votre application 7](#_Toc472632722)

[Point d’entrée : 7](#_Toc472632723)

[Activité : 7](#_Toc472632724)

[Démarrer une activité : 7](#_Toc472632725)

[Intent : 7](#_Toc472632726)

[Les manifestes : 7](#_Toc472632727)

[Implémenter une activité et la relier à l’interface 8](#_Toc472632728)

[Activité : 8](#_Toc472632729)

[Evènement : 8](#_Toc472632730)

[TP : créer une appli avec une interface évolutive 9](#_Toc472632731)

[Définir l’application : 9](#_Toc472632732)

[Rentre interactif : 12](#_Toc472632733)

[Gérer les tailles d’écran 17](#_Toc472632734)

[Configurations : 17](#_Toc472632735)

[Matériel : 17](#_Toc472632736)

[Logiciel : 17](#_Toc472632737)

[Situationnel : 17](#_Toc472632738)

[Insérer une image : 17](#_Toc472632739)

[Les mages : 18](#_Toc472632740)

[Les langues : 18](#_Toc472632741)

[Coder une application rectro-compatible 20](#_Toc472632742)

[Thème et style : 20](#_Toc472632743)

[Style : 20](#_Toc472632744)

[Thème : 20](#_Toc472632745)

[Afficher une action bar et un menu 22](#_Toc472632746)

[Ajouter un titre : 22](#_Toc472632747)

[Ajouter un menu : 22](#_Toc472632748)

[TextView et Web View 24](#_Toc472632749)

[HTML : 24](#_Toc472632750)

[Texte plus long : 24](#_Toc472632751)

[Première méthode : 24](#_Toc472632752)

[Pour afficher du html complexe : les webView : 25](#_Toc472632753)

[Deuxième méthode : les assets : 25](#_Toc472632754)

[Styliser : 25](#_Toc472632755)

[Le javaScript : 26](#_Toc472632756)

[Ouvrir des liens : 26](#_Toc472632757)

[Afficher des listes avec RecyclerView : 27](#_Toc472632758)

[Histoire : 27](#_Toc472632759)

[Avant lollipop : 27](#_Toc472632760)

[Lollipo : 27](#_Toc472632761)

[Mise en route : 27](#_Toc472632762)

[Afficher plus de donnée au clique : 31](#_Toc472632763)

[Rapide couleur sous le titre au clique : 31](#_Toc472632764)

# Installer Android studio

* Installer Java developpement kit
* Installer Android Studio <https://developer.android.com/studio/index.html>

Attention, sur Windows et Mac OS X, dans la fenêtre “License Agreement’ qui s’affiche lors de l’installation, vous devez accepter 2 licenses : “android-sdk-license” et “intel-android-extra-license” !

## Configurer SDK manager :

Installez [Intel HAXM pour Windows](https://software.intel.com/en-us/android/articles/installation-instructions-for-intel-hardware-accelerated-execution-manager-windows). Pour cela :

* Téléchargez HAXM dans le SDK Manager
* Avec l’explorateur de windows, rendez-vous dans le dossier d’installation du SDK Android (par défaut installé dans C:\Users\[votre-login]\AppData\Local\Android\sdk)
* Rendez-vous dans le sous-dossier extras\intel\Hardware\_Accelerated\_Execution\_Manager
* Lancez intelhaxm-android.exe et suivez les instructions ;)

API cible : api avec laquelle on va compiler note application. Dernière version stable disponible.

API minimale : Minimale requis, par exemple si on utilise une méthode qui n’est pas diponible dans API minimale, problème.

Dans l’interface, le mode project affiche les dossiers et sous-dossiers tels qu’ils sont sur le disque. L’affichage Android affiche une version plus optimisée pour le développement.

### Gradle :

Pour compiler une application, il faut précompiler le xml, copiler le java .. etc plusieurs tâches successives sont obligatoires. Pour faciliter tout ça, on utilise Gradle.

### Menu :

Dans le menu on a SDK manager, qu’on peut ouvrir pour rajouter des plugins ou autre fonctionnalité.

AVD manager pour gérer les machines virtuelles Android.

### Mode développeur :

Pour activer le mode développeur, appuyer 7 fois sur « A propos de l’appareil »

# Première interface

## Les ressources :

On peut les mettre dans deux catégories :

* Images
* XML

C’est tout ce qui n’est pas du code.

Type de ressource xml le plus important : interface 🡺 layout.

On les trouve dans res 🡺 layout.

Si on ouvre activity\_main.xml on obtient une interface. Elle est du type WYSWYG.

On peut configurer avec glisser-déposer.

Les propriétés sont en bas à droite, celles qui sont utilisées le plus souvent sont en gras.

* Aligner le texte : gravity

### Images

Pour insérer une image : on utilise le format PNG. On les trouve dans l’affichage project sous main/res/drawable. On doit copier l’image en restant appuyer sur CTRL. Le nom ne peut contenir que des minuscules, underscore …

### Texte :

En bas à gauche de l’interface, on voit :

* Design
* Text

Le texte est le code XML.

Le code en gris clair change en cliquant dessus. Avec CTRL click on change l’affichage pour avoir un detail.

## Android à sa création :

Android a défini un OS sans construrie les téléphones, il doit donc s’adapter.

Pour cela il se base sur 3 principes :

* Unités
* Layouts
* Configurations

### Unité :

Le px. Plus la densité est élevé plus il y a de PX sur une même distance. Donc pas la même distance pour 2 téléphones différents.

**Google a crée le DP pour *Display-independant Pixel*.**

**Pour les textes on utilise les SP pour *Scale-independent PixelI.***

SP environ = 1 DP.

### Layout :

Une vue qui affiche d’autres vues. Tous les layouts commencent par « layout\_ ». Ils doivent au moins définir :

* Layout\_width
* Layout\_height

Ils indiquent au layout parent comment les placer et les redimensionner. Ils peuvent avoir différentes valeurs :

* Match\_parent : pour indiquer qu’ils doivent prendre toute la taille possible.
* Wrap\_content : qui dit que la vue doit être le plus petit possible.
* Une valeur en dp pour une taille absolue.

### Les layouts les plus utilisés :

Pour créer un layout : clique droit sur layout 🡺 new layout resource file.

On donne un nom et on choisit le layout de base.

Le plus simple :

* FrameLayout layout qui n’arrange pas les éléments entre eux, ils peuvent se chevaucher.
* Linearlayout vertical ou horizontal pour les aligner. Le poid est defini par weight. Celui qui a le plus de poids, prend le plus de place et les autres au minimum.
* Relativelayout, puissant et complexe, positionne où on veut.

# Découpez votre application

## Point d’entrée :

En java, le point d’entrée est main.

Sur android, les activités. Permet d’afficher à l’écran et intéragir avec l’utilisateur.

### Activité :

C’est un écran, on navigue entre les activités.

Chaque activité peut être un point d’entrée. Exemple on clique sur Hangout, on clique sur partager, on clique sur une notification … on lance une activité de cette application.

### Démarrer une activité :

On utilise un *Intent*. Quand on clique sur une photo, l’activité en cours crée un intent et le donne à Android. Celui-ci renvoit une activité demandé par l’intent.

### Intent :

On veut partager une photo, on lance un intent. Toutes les activités cappable de récupérer cette intent s’affichent. L’utilisaeur choisit, l’application reçoit l’intent et le traite.

## Les manifestes :

Comment Android peut savoir les activités disponible ?

Pour déclarer les infos, on utilise un manifeste. Le manifeste est un xml qui déclare tout ce qui Android doit savoir sur l’application.

Une appli’ doit déclarer les permissions. On ne pourra accéder à internet que si on a la permission (à l’installation). Elles sont définies dans le manifeste.

Aujourd’hui, les permissions non-critiques seront données, les permissions critiques devra demander la permission.

# Implémenter une activité et la relier à l’interface

## Activité :

Une activité est une classe Java.

On va mettre notre code dans l’activité Java. Une classe activité ne doit pas avoir de constructeur.

Pour afficher une vue, on utilise la méthode setContentView(R.layout.nomdulayout). Ici R signifie resources.

### Evènement :

Pour enregistrer une callback sur la vue :

Quand on a créé une vue : findViewById().

Les ID sont des resources spéciales, ils ne sont pas déclarés. Si on prend le code XML, on voit android :id. Le @+ signifie qu’il doit mettre l’id à notre classe ou le crée s’il n’existe pas.

On obtient ce code pour main activity :

**package** com.oc.rss.oc\_rss;  
  
**import** android.content.Intent;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***bouton***);  
 Button myButton = (Button) findViewById(R.id.***button***);  
 myButton.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Log.*i*(**"oc\_rss"**, **"OK"**);  
  
 Intent myIntent = **new** Intent(MainActivity.**this**, NextActivity.**class**);  
 startActivity(myIntent);  
 }  
 });  
 }  
  
}

On récupère la vue avec setContentView.

On ajoute un Listener au bouton et on spécifie qu’il s’agit du click.

A l’intérieur on indique un Intent qui dit remplace notre activité par la suivante.

**Il ne faut pas oublier d’indiquer au manifeste notre seconde classe :**

<**activity android:name=".NextActivity"**/>

# TP : créer une appli avec une interface évolutive

## Définir l’application :

La première chose à faire quand on crée une application, c’est de bien définir son périmètre : son objectif, ses cas d’utilisation, ses caractéristiques. Cela vous permettra d’établir un cahier des charges précis et d’identifier les étapes nécessaires pour réaliser cette application.

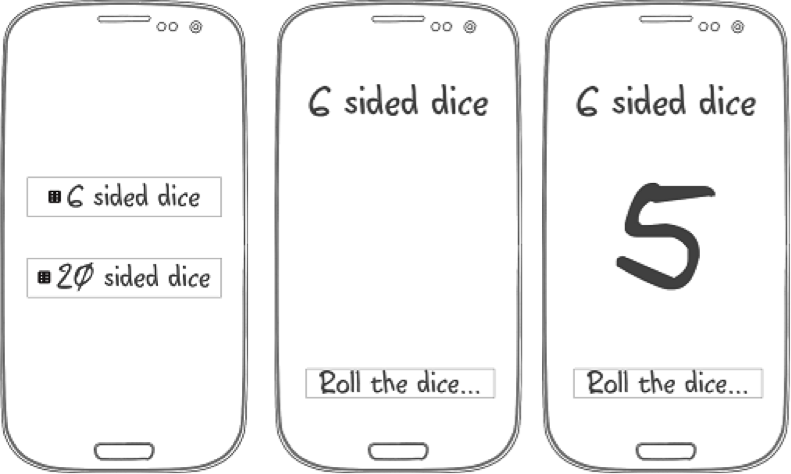
#### **Step 1 : Décrivez une utilisation type de l’application**

Voici ce que je vous propose dans ce TP :

Après avoir lancé l’application, l’utilisateur peut choisir un dé à 6 faces ou à 20 faces. Une fois le dé choisi, l’utilisateur est présenté à une interface qui contient un bouton permettant de lancer le dé. Enfin, une fois le dé lancé, un nombre aléatoire s’affiche dans l’interface entre 1 et le maximum du dé. L’utilisateur peut relancer le dé autant de fois qu’il le souhaite.

#### **Step 2 : Faites un brouillon rapide**

pour montrer à quoi ressembleront interfaces de l’application :



#### Step 3 : Faites un découpage technique de votre application

Ici, on va faire 2 activités :

* une activité correspondant à l’écran d’accueil (écran de gauche), et
* une activité correspondant aux écrans de lancer du dé (écran du centre) et du résultat (écran de droite).

Bon, on sait ce qu’on va faire et comment on va le faire, il est donc temps de le faire !

#### Activité 1 : homepage

Voici donc ma proposition pour l'activité représentant l’écran d’accueil de l’application (dans activity\_main.xml) :

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:padding="20dp">

<Space

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1" ></Space>

<FrameLayout

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1">

<Button

android:id="@+id/button6"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:padding="20dp"

android:text="6 sided dice"

android:textSize="18sp" ></Button>

</FrameLayout>

<FrameLayout

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1">

<Button

android:id="@+id/button20"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:padding="20dp"

android:text="20 sided dice"

android:textSize="18sp" ></Button>

</FrameLayout>

<Space

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1" ></Space>

</LinearLayout>

Ce code présente plusieurs éléments que nous avons déja vus :

* Le layout de base de l’interface est un LinearLayout vertical.
* Les éléments ont tous un poids (layout\_weight ) à 1, ce qui signifie que l’interface prendra toute la taille verticale de l’écran et que chaque élément se verra attribué une hauteur similaire.
* J’ai mis une grosse taille des textes des boutons (18sp).
* Les boutons sont intégrés dans des FrameLayout afin qu’ils puissent définir leur taille verticale.

Notez que les boutons ont les ID suivants : button6 et button20 . Ca sera important plus tard.

Il présente aussi plusieurs éléments nouveaux :

* J’ai utilisé une vue vide appelée Space pour remplir les espaces en haut et en bas. Cette vue ne fait littéralement rien d’autre que de prendre de l’espace ;).
* J’ai mis un padding aux LinearLayout principal afin que les boutons ne touchent pas le bord de l’écran. Pour le même effet, j’aurai pu mettre un layout\_margin aux boutons. (rendez-vous dans le cours de Mathieu Nebra sur HTML et CSS pour comprendre la [différence entre padding et margin](https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/le-modele-des-boites#/id/r-2302207)).

#### Activité 2 : lancer du dé et résultat

Et voici ma proposition pour la seconde activité (activity\_dice.xml) :

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:padding="20dp">

<TextView

android:id="@+id/textTitle"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

android:text="X sided dice"

android:textSize="30sp" ></TextView>

<TextView

android:id="@+id/textResult"

android:layout\_width="wrap\_content"

android:layout\_height="0dp"

android:layout\_gravity="center"

android:layout\_weight="1"

android:gravity="center"

android:text="X"

android:textSize="120sp" ></TextView>

<Button

android:id="@+id/buttonRoll"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:padding="20dp"

android:text="Roll the dice..."

android:textSize="18sp" ></Button>

</LinearLayout>

Ici, pas de triche, que des choses qu’on a déjà vues :

* Le TextView central qui affichera le résultat est en très gros (120sp) et prend toute la place (il est le seul à avoir l’attribut layout\_weight).
* Notez que j’ai mis des valeurs d’exemple (“comme X sided dice”) que l’utilisateur ne devra pas voir, nous devrons redéfinir le contenu des textviews lors de l’initialisation de l’activité.
* Notez que le TextView central définit les attributs layout\_gravity (la position de la vue au sein de son layout) ET gravity (la position du texte au sein du TextView).

Prenez une minute pour jouer avec les attributs de ces deux interfaces. Arrivez-vous a prédire ce qui va changer sur l’écran lorsque vous modifiez un élément ou un attribut ?

## Rentre interactif :

Lorsqu’on ouvre la classe Java MainActivity qui a déjà été créée par Android Studio, on constate qu’il y a déjà du code. En plus de OnCreate, Android Studio a déjà ajouté deux autres méthodes (onCreateOptionsMenu et onOptionsItemSelected). Ces deux fonctions se rapportent au menu, que nous verrons plus tard. Comme nous n’avons pas de menu dans notre interface, vous pouvez supprimer ces deux méthodes.

Android Studio étant très sympa, il nous a déjà mis l’appel à setContentView dans onCreate, donc si on lance déjà l’application, la première interface s’affiche. :)

Avant de nous intéresser aux boutons, créons notre deuxième activité. Faites un clic droit sur le package de votre application > New > Java Class, et nommez cette classe DiceActivity :

public class DiceActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_dice);

}

}

Enfin, n’oubliez pas de vous rendre dans le manifeste de l’application afin de déclarer l’activité (juste en dessous du </activity>) ;) :

<activity android:name=".DiceActivity" />

Retournons dans MainActivity. On va créer les listeners pour les deux boutons (dans la méthode onCreate, à la suite du setContentView) pour que lorsque l’on clique sur les boutons, on soit redirigé vers l’activity DiceActivity. Faites-le avant de regarder la correction. ;)

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Button button6 = (Button) findViewById(R.id.button6);

button6.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(MainActivity.this, DiceActivity.class);

startActivity(intent);

}

});

Button button20 = (Button) findViewById(R.id.button20);

button20.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(MainActivity.this, DiceActivity.class);

startActivity(intent);

}

});

}

}

Pour l’instant, les deux boutons font exactement la même chose. On s’occupera de les différencier plus tard.

L’étape suivante consiste à écrire le code de DiceActivity comme si l’on avait cliqué sur “6 sided dice”. Donc, écrire tout le fonctionnement de l’activité pour le dé à 6 faces.

Tout d’abord, dans la méthode onCreate, comme prévu, on va changer les textes. On va mettre le titre à “6 sided dice” et le résultat à “” (string vide) :

public class DiceActivity extends Activity {

private TextView textResult;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_dice);

TextView textTitle = (TextView) findViewById(R.id.textTitle);

textTitle.setText("6 sided dice");

textResult = ((TextView) findViewById(R.id.textResult));

textResult.setText("");

}

}

textResult est une propriété de la classe alors que textTitle est une variable. Il y a une raison à cela que vous verrez dans une minute. ;)

Il s’agit maintenant de changer le texte de textResult à une valeur comprise entre 1 et 6 (inclus) lorsqu’on clique sur le bouton “Roll the dice!”. Toujours le même conseil : essayez de le faire seul avant de regarder la correction. :)

public class DiceActivity extends Activity implements View.OnClickListener {

private TextView textResult;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_dice);

TextView textTitle = (TextView) findViewById(R.id.textTitle);

textTitle.setText("6 sided dice");

textResult = ((TextView) findViewById(R.id.textResult));

textResult.setText("");

Button buttonRoll = (Button) findViewById(R.id.buttonRoll);

buttonRoll.setOnClickListener(this);

}

@Override

public void onClick(View v) {

SecureRandom random = new SecureRandom();

int result = random.nextInt(6) + 1;

textResult.setText(String.valueOf(result));

}

}

Ici, au lieu d’utiliser une classe anonyme, j’ai modifié DiceActivity pour qu’il implémente l’interface View.OnClickListener. C’est la raison pour laquelle textResult, qui est appelée dans la méthode onClick, est une propriété. C’est une autre pratique très courante dans le développement Android d’avoir l’activité qui implémente les interfaces listeners nécessaires.

Lancez l’application, et constatez que ça marche :D.

Il nous reste plus qu’à différencier le jet du dé à 6 faces et celui du dé à 20 faces. Je sais que vous avez pensé à dupliquer l’activité. Allez… ne mentez pas !

On ne va évidemment PAS faire ca.

En fait, si l’on réfléchit, on se rend compte que ce serait plus pratique de passer un paramètre à l’activité DiceActivity pour qu’elle puisse afficher un dé à 6 ou à 20 face en fonction d’un paramètre. Comme ça, pas besoin de dupliquer tout le code d’une activité dès qu’on veut changer le nombre de faces du dé, il suffira de changer la valeur du paramètre !

Et, bien sûr, nous pouvons passer des paramètres à l’intent.

Je vous propose donc de commencer par modifier MainActivity pour paramétrer les intents. Il suffit d’appliquer la méthode putExtra aux intents lancés par les boutons button6 et button20 :

intent.putExtra("max", 6);

et

intent.putExtra("max", 20);

Pour récupérer la valeur d’un extra d’un intent dans une activité, il suffit d’utiliser la méthode getIntExtrade la façon suivante :

int max = getIntent().getIntExtra("max", 0);

(le second paramètre à 0 est la valeur retournée si l’extra n’est pas présent dans l’intent).

Il ne vous reste plus qu’à modifier l’activité DiceActivity pour pouvoir afficher “X sided dice” (avec X à la bonne valeur) et faire le lancé de dé aléatoire correctement.

Voici la correction :

public class DiceActivity extends Activity implements View.OnClickListener {

private TextView textResult;

private int max;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_dice);

max = getIntent().getIntExtra("max", 0);

TextView textTitle = (TextView) findViewById(R.id.textTitle);

textTitle.setText(max + " sided dice");

textResult = ((TextView) findViewById(R.id.textResult));

textResult.setText("");

Button buttonRoll = (Button) findViewById(R.id.buttonRoll);

buttonRoll.setOnClickListener(this);

}

@Override

public void onClick(View v) {

SecureRandom random = new SecureRandom();

int result = random.nextInt(max) + 1;

textResult.setText(String.valueOf(result));

}

}

# Gérer les tailles d’écran

Rappel : pour avoir un système d’interface cappable de s’adapter aux écran, Android utilise un système adaptatif basé sur 3 principes :

* Unités
* Layout
* Configuration

## Configurations :

Info définissant comment fonctionne periph android sur plan méteriel et logiciel.

### Matériel :

* Taille d’écran
* Résolution d’écran
* Clavier disponbile ?

### Logiciel :

* Taille du texte
* Langue
* Orientation de l’écriture
* …

### Situationnel :

Orientationé écran ?

Imaginons que l’on ait une image en 4 résolutions différentes. Le but du jeu est d’afficher la bonne image en fonction de la bonne résolution.

Autre exemple : même inteface mais en fonction de l’orientation de l’écran.

Autre exemple : différentes langues ?

## Insérer une image :

En mode project, on va dans app/src/main/res puis on fait new android resources directory.

Dans le type on sélectionne drawable vu qu’il s’agit d’image et on met en colonne de droite density pour dire qu’on va jouer sur la densité.

On doit maintenant choisir entre plusieurs valeurs.

Les capacités XXX-high DPI et low dpi sont assez rare, on va donc s’occuper des autres. Sur une très bonne tablette, Android va donc rpendre les configuration xx-high et les agrandir. En cas de low density, Android va prendre les configurations de medium DPI et les mettre plus petite.

On va donc créer les Drawable pour :

* Medium
* High
* X-high
* XX-high

**Les images doivent avoir exactement le même nom**

On va donc avoir en xml : android :src : ‘’@drawable/android\_logo ‘’

## Les mages :

A la création du projet, Android a déjà créé : « values-w820dp »

Les configurations XML qui s’appliquent dans ce dossiers s’appliquent pour ceux qui auront au moins 820dp de large. Le W signifiant width.

Un fichier est déjà présent dedans avec des marges horizontales à 60dp.

Dans values et non valuesw820dp : dimens.xml a lui deux dimensions : horizontale et verticale.

## Les langues :

Sous Android, la langue du téléphone est aussi une configuration. La langue de base est l’anglais, on peut ensuite créer d’autres parties pour le français, ce qui signifie que n’importe qui dans son pays télécharge l’applie elle sera en anglais sauf pour ceux qui la télécharge dans un pays où on a redéfini la langue, ici le français.

Dans values, il y a déjà un strings.xml, donc si on crée un string avec comme name hello\_world et comme valeur hello\_world, on copie colle dans values français et on change la valeur en Bonjour le monde.

Pour tester on fait une référence à un texte :

<TextView

Android :text= »@string/hello\_world »

Si on veut récupérer la string dans notre code, on fait : getString(R.string.hello\_world) ;

Si on repasse en vue Android, on ne voit pas les configurations que l’on vient de créér.

Il s’agit d’une vue intelligente dans laquelle tout n’est affiché qu’une fois. Par exemple, dans drawable, on voit les 4 fichiers avec leur configuration entre parenthèse et non chaque drawable créé.

# Coder une application rectro-compatible

Depuis Android 5 : materiel design.

Ce n’est pas le design android mais google. C’est un design language.

* Pour les designers : cartes, floating action button, actions bars …
* Pour l’utilisateur : La sémantique est claire et consistante.

Il n’impose aucune couleur et c’est au designer de les choisir.

Les deux couleurs principales à choisir sont :

* Primary : définie la couleur primaire de l’application
* Accent : couleur qui attirera l’oeil

Se référer à [android partie design](https://developer.android.com/design/index.html)

## Thème et style :

### Style :

Les style récupères les informations visuelles des layout pour les mettre dans res/style.

En cas de code réutilisable, on crée un style nommé par exemple title pour pouvoir réappliquer le style.

### Thème :

Ensemble d’attribut style qui s’applique à une vue.

Holo et Material sont donc des thèmes pour Android.

Android 5 (avec le thème Material) dispose aussi du thème Holo pour que les applications développées sous Holo puissent toujours fonctionner sur Material.

Chaque thème défini des sous-thèmes en 2 catégories.

* Dark
* Light

**La question est donc, comment faire une appli moderne avec des techniques modernes sur un téléphone vieux ?**

* La librairie de support.

C’est une libraire externe à Android pour back porter des fonctionnalités.

La librairie de support  *AppCompat* fourni le thème Material en l’émulant.

On la déclare dans le script gradle.

Rappel : Maeven (ajd gradle) récupère des libraire et les compile.

Donc en créant un nouveau projet, AppCompat s’est automatiquement rajouté et on peut le constater dans le manifeste de l’application.

# Afficher une action bar et un menu

L’action bar se situe en haut et a souvent un overflow menu parce qu’il n’y a pas assez de place dans l’action bar.

L’action bar n’est pas une vue du layout de l’activité mais affiché directement par android en fonction de la configuration de l’activité et du thème.

Depuis Android 5 : la toolbar c’est comme une vue, on peut la mettre où on veut.

Il faut déclarer le thème dans le manifest. Android :theme=’’@style/AppTheme’’.

### Ajouter un titre :

On change le titre avec setTitle(String) ; On le met dans onCreate.

### Ajouter un menu :

* On crée un fichier ressource
* On va sur res nouveau android ressource directory, on choisit menu sans contrainte.
* Sur le menu qui s’est créé, new menu ressource file 🡺 main\_menu

**On edit main\_menu.xml :**

*<?***xml version="1.0" encoding="utf-8"***?>*<**menu xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"**>  
  
 <**item  
 android:id="@+id/action\_edit"  
 android:title="Edit"  
 android:icon="@android:drawable/ic\_menu\_edit"  
 app:showAsAction="always"** />  
  
 <**item  
 android:id="@+id/action\_add"  
 android:title="Edit"  
 android:icon="@android:drawable/ic\_menu\_add"  
 app:showAsAction="ifRoom"**/>  
  
 <**item  
 android:id="@+id/action\_delete"  
 android:title="Delete"  
 android:icon="@android:drawable/ic\_menu\_delete"  
 app:showAsAction="never"**/>  
</**menu**>

Ici on utilise les icônes d’android, d’où @android :drawable et non directement drawable car nous n’avons pas importé d’icône.

showAsAction défini si elle s’affichera dans l’action bar ou le menu.

* Always
* ifRoom
* never

n’est pas préfixé par android mais app car elle est vient de librairie de support.

**Pour déclarer le munu, on Override onCreateOptionMenu**

@Override  
**public boolean** onCreateOptionsMenu(Menu menu) {  
 getMenuInflater().inflate(R.menu.***main\_menu***, menu);  
 **return true**;  
}

On remarque en lançant l’application que :

* Always s’affiche dans l’action bar
* ifRoom s’il y a de la place
* Never s’affiche directement dans l’overflow menu

Ensuite on OverRide la méthode sur la selection des item :

**public boolean** onOptionItemSelected(MenuItem item)  
{  
 **switch**(item.getItemId()){  
 **case** R.id.***action\_edit***:  
 **return true**;  
 **case** R.id.***action\_add***:  
 **return true**;  
 **case** R.id.***action\_delete***:  
 **return true**;  
 }  
 **return super**.onOptionsItemSelected(item);  
}

# TextView et Web View

## HTML :

Pour afficher du contenu html, on peut utiliser TextView, il faut renseigner un objet au lieu de renseigner un string.

Cet objet est *Html.fromHtml(« string »)*.

## Texte plus long :

On va charger du contenu depuis un fichier en local.

Pour cela 2 méthodes, on peut utliser les ressources d’Android et donc utiliser R. …

### Première méthode :

Sur res on fait new file directory class RAW.

Une fois celle-ci créée on fait new file : « content.html »

**Les nom des fichiers sont comme les autres ressources, ils ne comportent que des lettres minuscules, des chiffres et des underscores.**

Il ne peut pas y avoir de sous dossier.

On peut maintenant créer du texte **formaté en HTML** et non **une page web**. On n’aura donc pas les balises HTML, HEAD et BODY.

@Override  
**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 TextView textview = (TextView) findViewById(R.id.***textView***);  
 **final** InputStream stream = getResources().openRawResource(R.raw.***content***);  
 **final** String text;  
 **try**{  
 text = IOUtils.*toString*(stream);  
 }**catch** (IOException e)  
 {  
 **throw new** RuntimeException(e);  
 }  
 textview.setText(Html.*fromHtml*(text));

On utilise la librairie IOUtils de Apache. On doit le spécifier à Gradle :

compile 'org.apache.commons:commons-io:1.3.2'

## Pour afficher du html complexe : les webView :

Permet d’afficher des vue web donc HTML, CSS et JS.

Pour charger du HTML, on utilise la seconde méthode :

### Deuxième méthode : les assets :

Pour créer ce dossier :

* Vue project
* App/src/main clique doit sur main
* New directory 🡺 assets

On peut revenir en Android et dans assets faire new file page.html

**Une webView agit comme une page web dans notrz application**

On doit donc remettre les balises HTML, HEAD et BODY.

@Override  
**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***texte***);  
 WebView webView = (WebView) findViewById(R.id.***webView***);  
 webView.loadUrl(**"file:///android\_asset/page.html"**);

On met aussi le texte en webView :

*<?***xml version="1.0" encoding="utf-8"***?>*<**FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent" android:layout\_height="match\_parent"**>  
  
<**WebView  
 android:id="@+id/webView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**/>  
  
</**FrameLayout**>

### Styliser :

Toujours dans assets, on créer style.css

On rajoute cette ligne dans l’entête :

<**head**>  
 <**title**>Page de test</**title**>  
 <**link rel="stylesheet" type="text/css" href="style.css"**>  
</**head**>

### Le javaScript :

Par sécurité il n’est pas activé sur les webView.

Pour l’activer :

WebView myWebView = (WebView) findViewById(R.id.webview);  
WebSettings webSettings = myWebView.getSettings();  
webSettings.setJavaScriptEnabled(true);

### Ouvrir des liens :

myWebView.setWebViewClient(new WebViewClient() {

@Override

public boolean shouldOverrideUrlLoading(WebView view, String url) {

view.loadUrl(url);

return false;

}

});

# Afficher des listes avec RecyclerView :

**Une liste est une répétition d’éléments affichant la même interface mais avec des données différentes.**

On ne va pas créer toutes les vues d’un coup ou ça va ramer, on va construire autant de vue que le système peut en afficher. Quand on défile, la vue qui devient caché sera réutilisé pour une vue future vers le bas. C’est le recyclage des cellules.

Il implique 2 choses

* Une vue n’est pas lié à une donnée.
* Quand on récupère une vue recyclé, elle affiche encore la vue précédente.

Quand on écrira, on écrasera l’ancienne donc pas très grave mais si on doti remplacer un contact FAVORI vers un contzact NON-FAVORI, si on ne pense pas à désactiver l’étoile favori, elle sera toujours affichée.

## Histoire :

### Avant lollipop :

* ListView : avait quelques beugs.

### Lollipo :

* RecyclerView : fait la même mais en plus complexe. C’est un librairie fournie pas Google et non propre à Android car permet à fournir RecyclerView pour tous les Android et non que cette version. Permet aussi d’améliorer en externe d’Android.

## Mise en route :

Comme pour toute librairie, on commence avec la mise en place de Gradle :

compile 'com.android.support:recyclerview-v7:+'

Comme RecyclerView n’est pas propre à Android et vient d’une librairie, elle n’est donc pas affiché dans les éléments par défaut dans la construction des layouts.

Pour l’ajouter il faut aller dans CUSTOM/CUSTOMVIEW/RECYCLERVIEW.

Dans le code XML, c’est le nom complet qui s’affiche, c’est la seule façon d’utiliser des éléments issues de librairies.

**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 setTitle(**"Test des lists"**);  
  
 **final** RecyclerView rv = (RecyclerView) findViewById(R.id.***list***);  
  
 rv.setLayoutManager(**new** LinearLayoutManager(**this**));  
 rv.setAdapter(**new** MyAdapter());

RecyclerView est plus puissant que ListView, on doot donc définir plus de paramètre.

* On récupère la vue avec son id
* On lui définit un layout : ici LineatLayoutManager() qui indique que les éléments seront disposés en ligne.
* setAdapter() définit le contenu.

Le code de l’adapter :

**package** com.oc.rss.oc\_rss;  
  
**import** android.app.AlertDialog;  
**import** android.app.Dialog;  
**import** android.support.v7.widget.RecyclerView;  
**import** android.util.Pair;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.TextView;  
  
**import** java.util.Arrays;  
**import** java.util.List;  
  
**public class** MyAdapter **extends** RecyclerView.Adapter<MyAdapter.MyViewHolder> {  
  
 **private final** List<Pair<String, String>> **characters** = Arrays.*asList*(  
 Pair.*create*(**"Lyra Belacqua"**, **"Brave, curious, and crafty, she has been prophesied by the witches to help the balance of life"**),  
 Pair.*create*(**"Pantalaimon"**, **"Lyra's daemon, nicknamed Pan."**),  
 Pair.*create*(**"Roger Parslow"**, **"Lyra's friends"**),  
 Pair.*create*(**"Lord Asriel"**, **"Lyra's uncle"**),  
 Pair.*create*(**"Marisa Coulter"**, **"Intelligent and beautiful, but extremely ruthless and callous."**),  
 Pair.*create*(**"Iorek Byrnison"**, **"Armoured bear, Rightful king of the panserbjørne. Reduced to a slave of the human village Trollesund."**),  
 Pair.*create*(**"Serafina Pekkala"**, **"Witch who closely follows Lyra on her travels."**),  
 Pair.*create*(**"Lee Scoresby"**, **"Texan aeronaut who transports Lyra in his balloon. Good friend with Iorek Byrnison."**),  
 Pair.*create*(**"Ma Costa"**, **"Gyptian woman whose son, Billy Costa is abducted by the \"Gobblers\"."**),  
 Pair.*create*(**"John Faa"**, **"The King of all gyptian people."**)  
 );  
  
 @Override  
 **public int** getItemCount() {  
 **return characters**.size();  
 }  
  
 @Override  
 **public** MyViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType) {  
 LayoutInflater inflater = LayoutInflater.*from*(parent.getContext());  
 View view = inflater.inflate(R.layout.list\_cell, parent, **false**);  
 **return new** MyViewHolder(view);  
 }  
  
 @Override  
 **public void** onBindViewHolder(MyViewHolder holder, **int** position) {  
 Pair<String, String> pair = **characters**.get(position);  
 holder.display(pair);  
 }  
  
 **public class** MyViewHolder **extends** RecyclerView.ViewHolder {  
  
 **private final** TextView **name**;  
 **private final** TextView **description**;  
  
 **private** Pair<String, String> **currentPair**;  
  
 **public** MyViewHolder(**final** View itemView) {  
 **super**(itemView);  
  
 **name** = ((TextView) itemView.findViewById(R.id.name));  
 **description** = ((TextView) itemView.findViewById(R.id.description));  
  
 itemView.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **new** AlertDialog.Builder(itemView.getContext())  
 .setTitle(**currentPair**.**first**)  
 .setMessage(**currentPair**.**second**)  
 .show();  
 }  
 });  
 }  
  
 **public void** display(Pair<String, String> pair) {  
 **currentPair** = pair;  
 **name**.setText(pair.**first**);  
 **description**.setText(pair.**second**);  
 }  
 }  
  
}

Et list\_celle :

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout 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:layout\_margin="15dp"  
 android:orientation="vertical"**>  
  
 <**TextView  
 android:id="@+id/name"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center\_horizontal"  
 android:layout\_marginBottom="15dp"  
 android:textAppearance="?android:attr/textAppearanceLarge"  
 tools:text="Personnage"** ></**TextView**>  
  
 <**TextView  
 android:id="@+id/description"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center\_horizontal"  
 android:lines="1" //Ne s’affiche que sur une seule ligne  
 android:ellipsize="end" // si + qu’une ligne, se termine par …  
 android:textAppearance="?android:attr/textAppearanceSmall"  
 tools:text="Description"** ></**TextView**>  
</**LinearLayout**>

L’adapter s’occupe de l’ensemble du conteneur.

ViewHolder s’occupe de la spécificité d’une cellule.

**public** MyViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType)

Le premier argument sert à créer la vue, le second pour définir le type de la cellule. Elles ont toutes le même ici donc on ne s’en occupe pas.

Pour créer une layout depuis un xml 🡺 LayoutInflater.

**public void** onBindViewHolder(MyViewHolder holder, **int** position)

Méthode qui applique une donnée à une vue

**On dit qu’on BIND la donnée à la vue.**

Prend en paramètre la ViewHolder qui va recevoir la donnée et la position de la cellule. onBindViewHolder sera toujours appelé pour affiché une vue, qu’elle soit nouvelle ou recyclée.

**public class** MyViewHolder **extends** RecyclerView.ViewHolder

Le constructeur prend en paramètre la vue à afficher.

On récupère ausis les 2 TextView car le ViewHolder se voit assigner une vue qui ne changera jamais. Comme récupérer une vue est une action lourde, on ne le fait qu’une fois, quand on la construit, pour aléger le code.

**public void** display(Pair<String, String> pair

Affiche les données de la pair fournie.

### Afficher plus de donnée au clique :

Rappel : une cellule peut afficher différents éléments en fonction de son état (cliquée ou non).

**public void** onClick(View view) {  
 **new** AlertDialog.Builder(itemView.getContext())  
 .setTitle(**currentPair**.**first**)  
 .setMessage(**currentPair**.**second**)  
 .show();  
 }

On utilise un listener pour ouvrir une boite de dialogue au clique.

### Rapide couleur sous le titre au clique :

Dans res/drawable

*<?***xml version="1.0" encoding="utf-8"***?>*<**selector xmlns:android="http://schemas.android.com/apk/res/android"**>  
  
 <**item android:state\_pressed="false" android:drawable="@android:color/transparent"** ></**item**>  
  
 <**item android:state\_pressed="true" android:drawable="@color/colorPrimary"** ></**item**>  
  
</**selector**>

Dans list :

android:background="@drawable/list\_cell\_background"

# Cycle de vie des activités

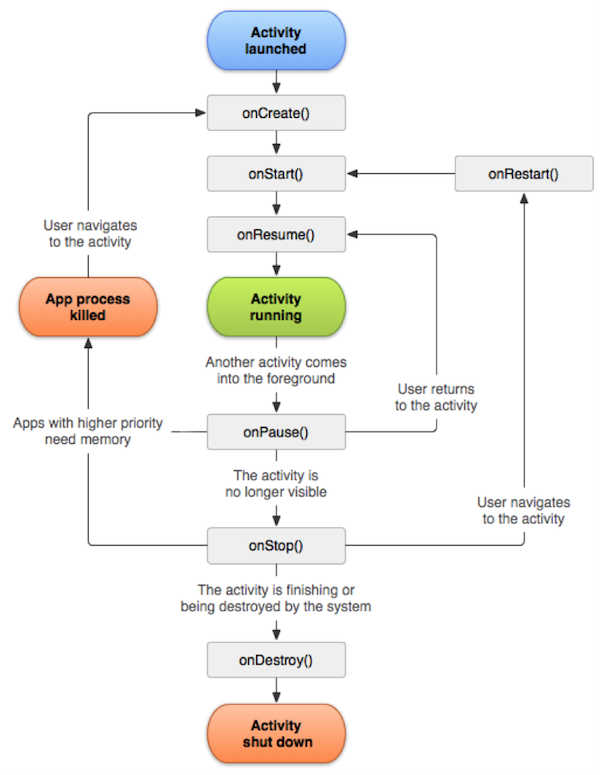
Rappel : Une activité est lancé par un intent, déclaré par une autre activité. Une application est une pile d’activité.

Quand une appli se termine (bouton précédent ou fin d’activité, on retrouve la précédente dans la pile.

### Le problème :

Plusieurs applications, chacune avec sa pile d’activité.

### Le cycle de vie :



Par défaut, dès que l’appli n’est plus visible, onPause et onStop sont appelées.

Si une appli a besoin de mémoire, il va arrêter complétement l’appli et reprendre à zéro quand on la reprend. (cas de gauche). Arrive quand l’appli est en background depuis longtemps et qu’il est temps de libérer de la mémoire.

Exemple :

On écrit un mail et on souhaite lire l’article qui s’affiche en notification (lourd en image, texte et vidéo). Android tue l’appli de mail.

Avant de tuer une appli’, Android fait un « saveInstanceState » et le mail n’est donc pas remis à zéro. Le mail est sauvegardé dans un Bundle qui sera appelé dans onCreate pour remettre l’ensemble de l’appli’ comme elle l’était. C’est comme un dico clé-valeur. Les Layouts peuvent le faire toute seul si chaque élément a un ID unique.

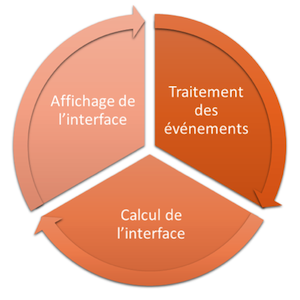
### Configuration :

Quand on passe de portrait à paysage ou l’inverse, Android relance l’appli’, il est donc important de TOUJOURS sauvegarder (cas de plus à droite).

# Réaliser des tâches lourdes sans bloquer l’appli’

### Boucle d’affichage :

Ensemble de trois tâches que le système Android affichera en boucle :



Elles s’effectuent au moins à 60 fps pour que ce soit fluide. C’est le nombre nécessaire à l’œil pour une interface fluide.

Avec notre code, tout se joue dans les traitements. Si le traiement est long, il bloque l’application et on dépasse les 16ms.

## Les threads :

C’est une file d’éxecution.

Si on a une animation qui s’exécute sur un thread, on doit utiliser un autre thread pour executer autre chose en même temps.

### Fonctionnement :

* On crée un objet thread
* Dedans on override la méthode run
* Dedans on fait le traitement lourd
* A la fin du thread, thread.start().

Mais ça plante : Android ne permet pas de modifier l’interface visuel autre que dans le Thread UI. Donc on doit dire que l’on effectue nos changements DANS le thread UI.

On fait donc à la fin de la méthode run, runOnUiThread(new Runnable() {.

Dedans on remet public void run() { avec l’affichage à modifier. On effectue tous les calculs et l’affichage final se met dans cette méthode run.

## AsyncTask<Entrée, Intermédiaire, Sortie>

Fourni par Android. Prend 3 paramètrs généreiques :

* Entrée, type d’argument passé au démarage lourd
* Si on ne veut pas informer l’état d el’avancement : Void
* Sortie : généré en résultat de la tache lourde

**package** com.dice.newsrss;  
  
**import** android.os.AsyncTask;  
**import** android.widget.TextView;  
  
*/\*\*  
 \* Created by Alexandre on 20/01/2017.  
 \*/***public class** MyLongTreatmentTask **extends** AsyncTask<Integer, Void, String> {  
 **private final** TextView **\_textView**;  
  
 **public** MyLongTreatmentTask(TextView textView)  
 {  
 **\_textView** = textView;  
 }  
  
 @Override  
 **protected** String doInBackground(Integer... params) {  
 **int** seconds = params[0];  
  
 **try**{ Thread.*sleep*(seconds\*1000);}  
 **catch** (InterruptedException ignored){}  
 **return "coucou"**;  
 }  
  
 @Override  
 **protected void** onPostExecute(String result){  
 **\_textView**.setText(result);  
 }  
}

**Ici on a donc un Integer en entrée, un String en sortie et on ne souhaite pas renseigner l’avancer.**

**Dans le constructeur on passe le textview à modifier à la fin du traitement.**

**doInBackgroud est appelé pour effectuer la fameuse tâche à executer en arrière plan, elle peut être appelé avec 0,1 ou plusieurs entiers. Comme celle-ci ne sera toujours appelé qu’avec un seul entier, on se contente de récupérer le premier.**

**onPostExecute est exécuté post-traitement. Il affecte le texte dans la view.**

Pour exécuter ce code : par exempel dans onClick() :

Final MyLongTreatmentTask task = new MyLongTreatmentTask(textView) ;

Task.execute(2).

Cette AsyncTask permet plus que ça (doc). En tout cas on ne peut en faire tourner qu’une seule en même temps.