

# Développement d'applications mobile

**Dr. Hamza DRID**

**E-mail:** [h.drid@univ-batna2.dz](mailto:h.drid@univ-batna2.dz)

**Année 2021/2022**

Source : Université de Rennes 1 (France)

# Plan du module

---

- ▶ **Plan du module**
  - ▶ Objectifs
- ▶ **Introduction aux concepts d'Android**
  - ▶ Introduction
  - ▶ Qu'est-ce qu'Android
  - ▶ Historique
  - ▶ Remarque sur les versions d'API
  - ▶ Distribution des versions
  - ▶ Programmation d'applications
  - ▶ Composants applicatifs
  - ▶ Applications et activités
  - ▶ Cycle de vie d'une activité
- ▶ **Création d'interfaces utilisateur**
- ▶ **Les Intents**
- ▶ **Persistance des données**



# Objectifs

---

- ▶ **À la fin de ce cours, vous serez capable de :**
  - ▶ Bâtir l'interface d'une application mobile
  - ▶ Naviguer et faire communiquer des applications
  - ▶ Manipuler des données (fichiers, BDD, ...)
  - ▶ Services, threads ...



# Objectifs

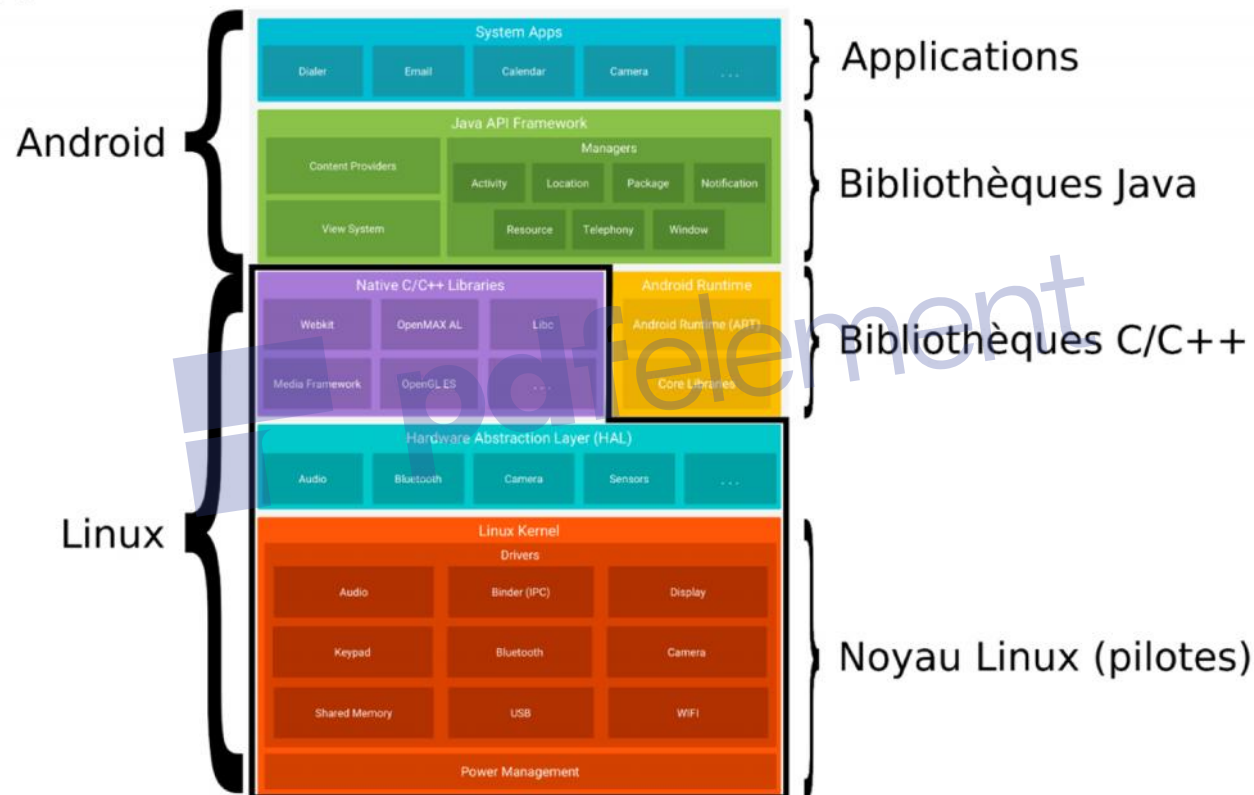
---

- ▶ Il y aura 8 semaines de cours, chacune comptant 1h30 CM et 1h30 TP.
- ▶ Cette semaine nous allons découvrir l'environnement de développement Android :
  - ▶ Le SDK Android et Android Studio
  - ▶ Création d'une application simple
  - ▶ Communication avec une tablette.



# Qu'est-ce qu'Android ?

- ▶ Android est une sur-couche au dessus d'un système Linux



- ▶ <https://developer.android.com/guide/platform>

# Historique

---

- ▶ Né en 2004
- ▶ Racheté par Google en 2005
- ▶ Publié en 2007, version 1.5
- ▶ En 2015, Android est le SE mobile le plus utilisé dans le monde, devant iOS d'Apple
- ▶ Quelques chiffres :
  - ▶ juillet 2011: **550 000** activations
  - ▶ décembre 2011: **700 000** activations
  - ▶ sept. 2012: **1.3 millions** d'activations ([Wikipedia](#))
  - ▶ avril 2013: **1.5 millions** d'activations ([Wikipedia](#))
  - ▶ Il y aurait donc un parc de **400 millions** d'appareils Android.



# Historique

---

- ▶ De nombreuses versions depuis.
- ▶ On en est à la version 12 (oct.2021) et l'API 32.
  - ▶ La version 12 est le numéro pour le grand public, et les versions d'API sont pour les développeurs.
    - ▶ Exemples :
      - 4.1 JellyBean = API 16,
      - 6.0 Marshmallow = API 23,
      - 11 RedVelvetCake = API 30
- ▶ Une API (Application Programming Interface) est un ensemble de bibliothèques de classes pour programmer des applications. Son numéro de version donne un indice de ses possibilités.



## Remarque sur les versions d'API

---

- ▶ Chaque API apporte des fonctionnalités supplémentaires.
- ▶ Il y a compatibilité ascendante.
- ▶ On souhaite toujours programmer avec la dernière API (fonctions plus complètes et modernes), mais les utilisateurs ont souvent des smartphones plus anciens, qui n'ont pas cette API.
- ▶ Les smartphones restent toute leur vie avec l'API qu'ils ont à l'naissance.
- ▶ Les développeurs doivent donc choisir une API qui correspond à la majorité des smartphones existant sur le marché.
- ▶ Vous êtes condamné(e) à une autoformation permanente.





# Distribution des versions

Android Platform/API Version Distribution

ANDROID PLATFORM VERSION	API LEVEL	
4.1 Jelly Bean	16	Nougat
4.2 Jelly Bean	17	User Interface
4.3 Jelly Bean	18	Multi-window Support
4.4 KitKat	19	Notifications
5.0 Lollipop	21	Quick Settings Tile API
5.1 Lollipop	22	Custom Pointer API
6.0 Marshmallow	23	Performance
7.0 Nougat	24	Profile-guided JIT/AOT Compilation
7.1 Nougat	25	Quick Path to App Install
8.0 Oreo	26	Sustained Performance API
8.1 Oreo	27	Frame Metrics API
9.0 Pie	28	Battery Life
10. Q	29	Doze on the Go
		Project Svelte: Background Optimizations
		SurfaceView
		Wireless & Connectivity
		Data Saver
		Number Blocking
		Call Screening
		Graphics
		Vulkan API
		System
		Direct Boot
		Multi-locale Support, More Languages
		ICU4J APIs in Android
		APK Signature Scheme v2
		Scoped Directory Access
		Keyboard Shortcuts Helper
		Virtual Files
		Android for Work
		Work profile security challenge
		Turn off work
		Always on VPN
		Customized provisioning
		Accessibility
		Vision Settings on the Welcome screen
		Security
		Key Attestation
		Network Security Config
		Default Trusted Certificate Authority
		VR
		Platform support and optimizations for VR Mode
		Printing Framework
		Print service enhancements

# Programmation d'applications

---

- ▶ **Actuellement, les applications sont :**
  - ▶ «natives», c'est à dire programmées en Java, C++, Kotlin, compilées et fournies avec leurs données sous la forme d'une archive Jar (fichier APK). C'est ce qu'on étudiera ici.
  - ▶ «hybrides», elles sont développées dans un framework comme Flutter, React Native. . . Ces frameworks font abstraction des particularités du système : la même application peut tourner à l'identique sur différentes plateformes (Android, iOS, Windows, Linux. . . )



# Applications natives

---

- ▶ Une application native Android est composée de :
  - ▶ Sources Java(ou Kotlin) compilés pour une machine virtuelle appelée «ART», amélioration de l'ancienne machine «Dalvik»(versions 4.4).
  - ▶ Fichiers appelés ressources:
    - ▶ format XML : interface, textes. . .
    - ▶ format PNG : icônes, images. . .
  - ▶ Manifeste= description du contenu du logiciel
    - ▶ version minimale du smartphone,
    - ▶ fichiers présents dans l'archive avec leur signature,
    - ▶ demandes d'autorisations, durée de validité, etc.
- ▶ Tout cet ensemble est géré à l'aide d'un IDE (environnement de développement) appelé Android Studio qui s'appuie sur un ensemble logiciel (bibliothèques, outils) appelé SDK Android



# Programmation d'applications Android

---

- ▶ Une application Android est composée de :
  - ▶ Sources Java (ou Kotlin) compilés pour une machine virtuelle appelée «ART », amélioration de l'ancienne machine «Dalvik» (versions 4.4).
  - ▶ Fichiers appelés ressources :
    - ▶ format XML : interface, textes...
    - ▶ format PNG : icônes, images...
  - ▶ Manifeste = description du contenu du logiciel
    - ▶ Chaque projet contient à sa racine un fichier AndroidManifest.xml
    - ▶ Nomme le package de l'application (ID unique de l'application)
    - ▶ Déclare les composants applicatifs de l'application (Activités, services...)
    - ▶ Déclare les permissions que l'appli doit avoir pour fonctionner (passer des appels, droit d'accéder à Internet, GPS...)
    - ▶ Déclare le niveau min du SDK pour que l'application fonctionne
- ▶ Tout cet ensemble est géré à l'aide d'un IDE (environnement de développement) appelé Android Studio qui s'appuie sur un ensemble logiciel (bibliothèques, outils) appelé SDK Android.



# SDK et Android Studio

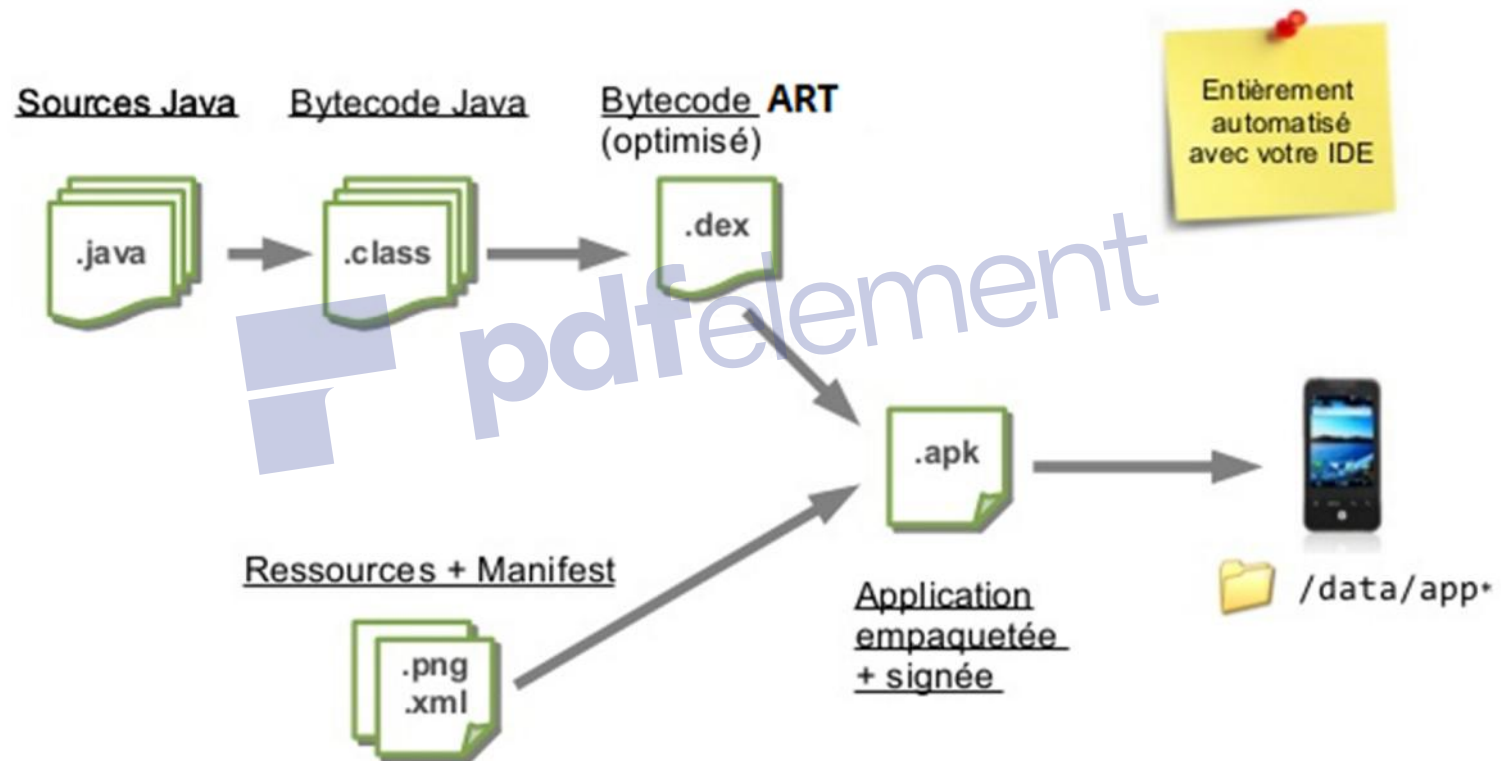
---

- ▶ Le SDK contient :
  - ▶ des librairies Java pour créer des applications
  - ▶ des outils de mise en boîte des applications
  - ▶ AVD : un émulateur de tablettes pour tester les applications
  - ▶ ADB : un outil de communication avec les vraies tablettes
- ▶ Android Studio offre :
  - ▶ un éditeur de sources et de ressources
  - ▶ des outils de compilation : gradle
  - ▶ des outils de test et de mise au point

(Vous allez voir tout cela en TP)



# Compilation et déploiement



# Composants applicatifs

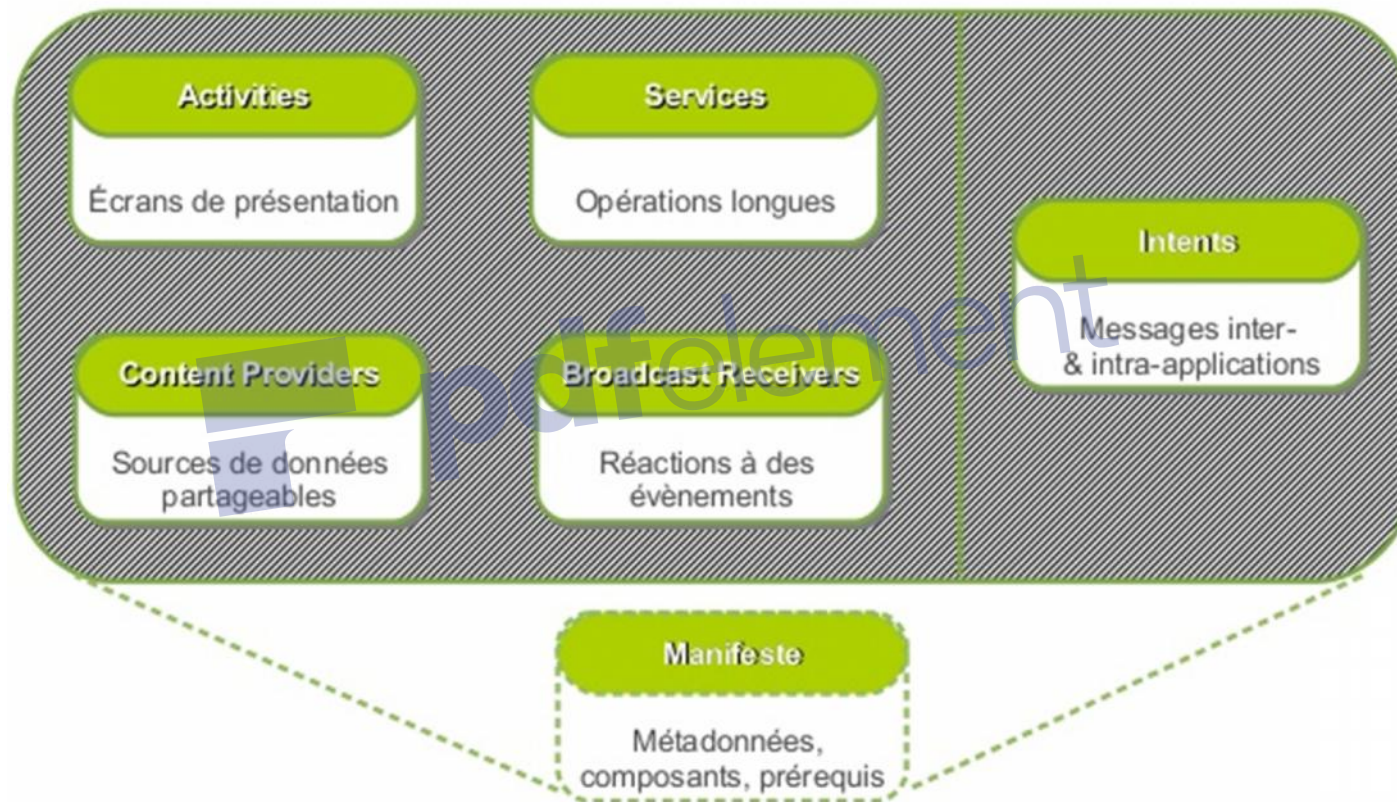
---

- ▶ Une application Android peut être composée des éléments suivants:
    - ▶ des activités (**android.app.Activity**): il s'agit d'une partie de l'application présentant une vue à l'utilisateur
    - ▶ des services (**android.app.Service**): il s'agit d'une tâche de fond sans vue associée
    - ▶ des *Intents* (**android.content.Intent**): permet d'envoyer un message pour un composant externe sans le nommer explicitement
    - ▶ des récepteurs d'*Intents* (**android.content.BroadcastReceiver**): permet de déclarer être capable de répondre à des *Intents*
    - ▶ des notifications (**android.app.Notifications**): permet de notifier l'utilisateur de la survenue d'événements
    - ▶ des widgets (**android.appwidget.**): une vue accrochée au Bureau d'Android
    - ▶ .....
- 



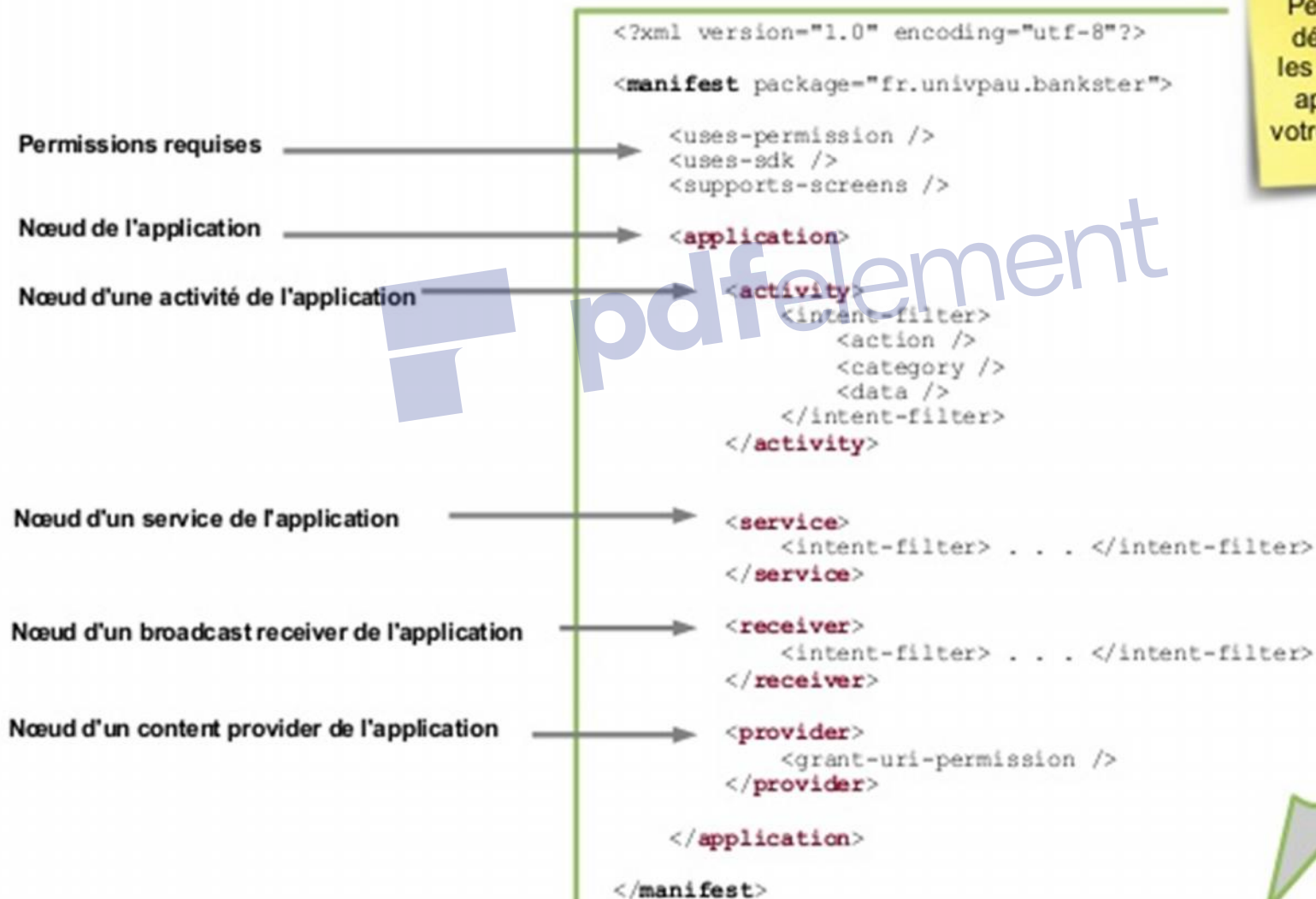


# Composants applicatifs





# AndroidManifest.xml



Pensez à bien déclarer **tous** les composants applicatifs de votre application !

# AndroidManifest.xml

---

- ▶ Before the Android system can start an app, the system must know that the component exists by reading the app's AndroidManifest.xml file.
- ▶ The app must declare all its components in this file, which must be at the root of the app project directory.



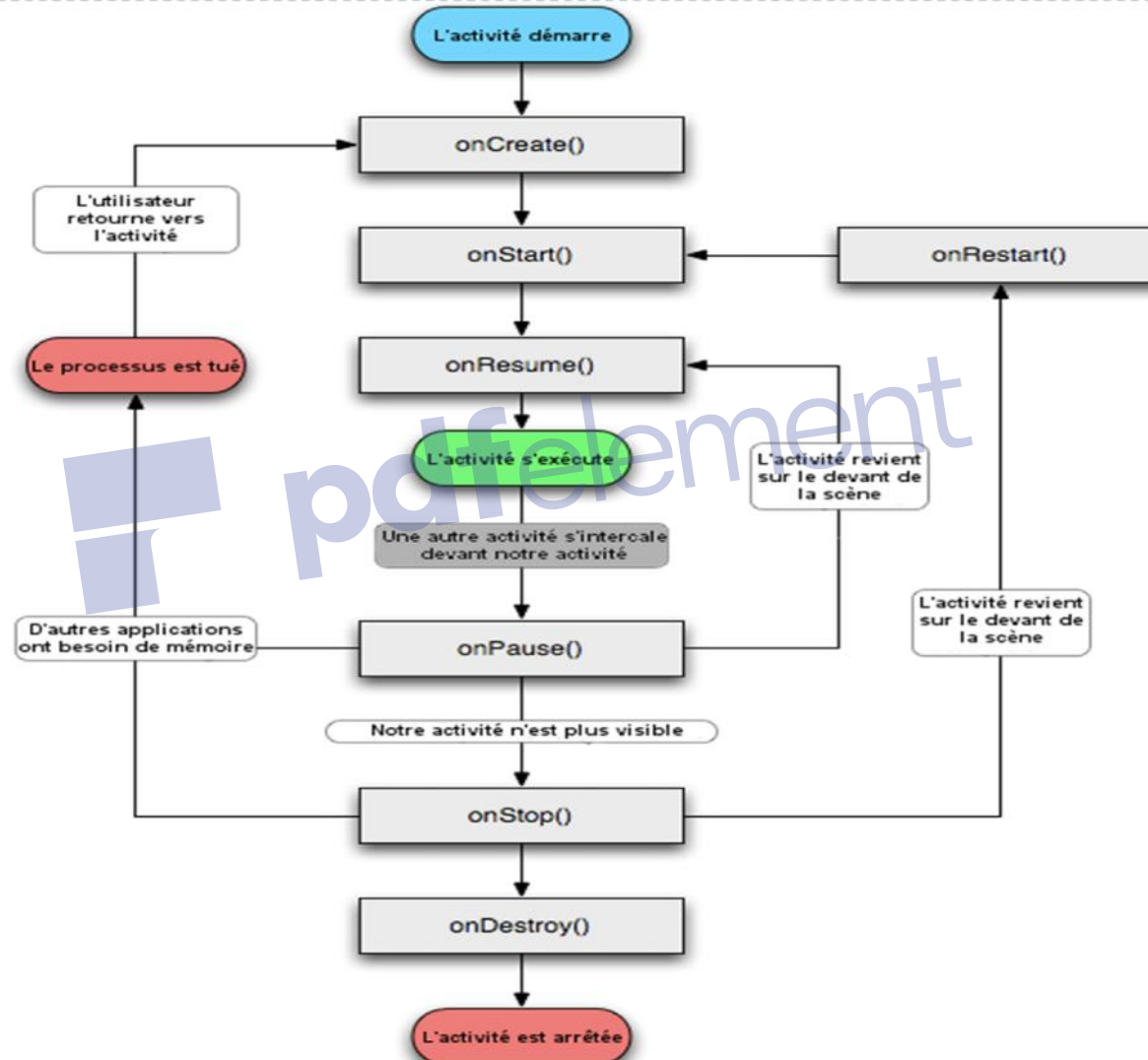
# Applications et activités

---

- ▶ Composition d'une application
  - ▶ L'interface utilisateur d'une application Android est composée d'écrans.
  - ▶ Un écran correspond à une activité, ex :
    - ▶ afficher des informations
    - ▶ éditer des informations
  - ▶ Android permet de naviguer d'une activité à l'autre, ex :
    - ▶ une action de l'utilisateur, bouton, menu ou l'application fait aller sur l'écran suivant
    - ▶ le bouton back ramène sur l'écran précédent.



# Applications et activités



# Cycle de vie d'une activité

---

- ▶ **onCreate() / onDestroy():** permet de gérer les opérations à faire avant l'affichage de l'activité, et lorsqu'on détruit complètement l'activité de la mémoire. On met en général peu de code dans **onCreate()** afin d'afficher l'activité le plus rapidement possible.
  - ▶ **onStart() / onStop():** ces méthodes sont appelées quand l'activité devient visible/invisible pour l'utilisateur.
  - ▶ **onPause() / onResume():** une activité peut rester visible mais être mise en pause par le fait qu'une autre activité est en train de démarrer, par exemple B. **onPause()** ne doit pas être trop long, car B ne sera pas créé tant que **onPause()** n'a pas fini son exécution.
  - ▶ **onRestart():** cette méthode supplémentaire est appelée quand on relance une activité qui est passée par **onStop()**. Puis **onStart()** est aussi appelée. Cela permet de différencier le premier lancement d'un relancement.
  - ▶ Le cycle de vie des applications est très bien décrit sur la page qui concerne les [Activity](#).
- 





# Développement d'applications mobile

---

- ▶ Le cours de cette semaine explique la création d'interfaces utilisateur :
  - ▶ Activités
  - ▶ Relations entre un source Java et des ressources
  - ▶ Layouts et vues



# Composition d'une application

---

- ▶ L'interface utilisateur d'une application Android est composée d'écrans.
- ▶ Un « écran » correspond à une activité, ex :afficher des informations éditer des informations
- ▶ Les dialogues et les pop-up ne sont pas des activités, ils se superposent temporairement à l'écran d'une activité.
- ▶ Android permet de naviguer d'une activité à l'autre, ex :
  - ▶ une action de l'utilisateur, bouton, menu ou l'application fait aller sur l'écran suivant
  - ▶ le bouton back ramène sur l'écran précédent.





# Structure d'une interface utilisateur

---

- ▶ L'interface d'une activité est composée de :
  - ▶ vues élémentaires : boutons, zones de texte, cases à cocher. . .
  - ▶ vues de groupement qui permettent l'alignement des autres vues : lignes, tableaux, onglets, panneaux à défilement. . .
  - ▶ Chaque vue d'une interface est gérée par un objet Java
    - ▶ Il y a une hiérarchie de classes dont la racine est **View**.
    - ▶ Elle a une multitude de sous-classes, dont par exemple **TextView**, **Button**.
    - ▶ Les propriétés des objets sont généralement visibles à l'écran : titre, taille, position, etc.



# Création d'une interface

---

- ▶ Ces objets d'interface pourraient être créés manuellement, voir plus loin, mais :
  - ▶ C'est très complexe, car il y a une multitude de propriétés à définir,
  - ▶ ça ne permet pas de localiser, c'est à dire adapter une application à chaque pays (sens de lecture de droite à gauche)
    - ▶ Alors, on préfère définir l'interface par l'intermédiaire d'un fichier XML qui décrit les vues à créer.
    - ▶ Il est lu automatiquement par le système Android lors du lancement de l'activité et transformé en autant d'objets Java qu'il faut.
    - ▶ Chaque objet Java est retrouvé grâce à un identifiant appelé « identifiant de ressource ».



# Création d'un écran

---

- ▶ Chaque écran est géré par une instance d'une sous-classe de **Activity** que vous programmez.
- ▶ Il faut au moins surcharger la méthode `onCreate` selon ce qui doit être affiché sur l'écran :

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

- ▶ C'est l'appel ***setContentView(...)*** qui met en place l'interface.
  - ▶ Son paramètre ***R.layout.main*** est l'identifiant d'une disposition de vues d'interface.
- 



# Ressources

---

## ► Définition

- Les ressources sont tout ce qui n'est pas programme (classes, bibliothèques) dans une application.
- Dans Android, ce sont les textes, messages, icônes, images, sons, interfaces, styles, etc.
- C'est une bonne séparation, car cela permet d'adapter une application facilement pour tous les pays, cultures et langues.
- On n'a pas à bidouiller dans le code source et recompiler chaque fois.
  - C'est le même code compilé, mais avec des ressources spécifiques.
  - Le programmeur doit simplement prévoir des variantes linguistiques des ressources qu'il souhaite permettre de traduire.
  - Ce sont des sous-dossiers, ex: values-fr, values-en, values-de, etc et il n'y a qu'à modifier des fichiers XML



# Ressources

---

- ▶ Le problème est alors de faire le lien entre les ressources et les programmes : par un identifiant.
  - ▶ Par exemple, la méthode **setContentView** demande l'identifiant de l'interface à afficher dans l'écran :  
**R.layout.main**
  - ▶ Cet identifiant est un entier qui est généré automatiquement par le SDK Android.
  - ▶ Comme il va y avoir de très nombreux identifiants dans une application
    - ▶ chaque vue possède un identifiant (si on veut)
    - ▶ chaque image, icône possède un identifiant
    - ▶ chaque texte, message possède un identifiant
    - ▶ chaque style, thème, etc. etc

Ils sont tous regroupés dans une classe spéciale appelée **R**

---



# Ressources

---

- ▶ Génération de la classe **R**
  - ▶ Le SDK Android construit automatiquement cette classe statique appelée **R**. Elle ne contient que des constantes entières groupées par catégories : *id*, *layout*, *menu*:

```
public final class R {  
    public static final class string {  
        public static final int app_name=0x7f080000;  
        public static final int message=0x7f080001;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    public static final class menu {  
        public static final int main_menu=0x7f050000;  
        public static final int context_menu=0x7f050001;  
    }  
    ...  
}
```

# Ressources

---

## ► La classe **R**

- Cette classe **R** est générée automatiquement
- Certaines de ces ressources sont des fichiers XML, d'autres sont des images PNG.
- Par exemple, le fichier ***res/values/strings.xml***:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Exemple</string>
    <string name="message">Bonjour !</string>
</resources>
```

- Cela rajoute automatiquement deux entiers dans R.string:  
***app\_name*** et ***message***



# Ressources

---

- ▶ Espaces de nommage dans un fichier XML
  - ▶ Dans le cas d'Android, il y a un grand nombre d'éléments et d'attributs normalisés. Pour les distinguer, ils ont été regroupés dans le *namespace* android.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">  
  <item  
    android:id="@+id/action_settings"  
    android:orderInCategory="100"  
    android:showAsAction="never"  
    android:title="Configuration"/>  
</menu>
```



# Ressources

---

- ▶ Ressources de type chaînes
  - ▶ Dans *res/values/strings.xml*, on place les chaînes de l'application, au lieu de les mettre en constantes dans le source :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="main_menu">Menu principal</string>
    <string name="action_settings">Configuration</string>
    <string name="bonjour">Demat !</string>
</resources>
```

- ▶ Intérêt : pouvoir traduire une application sans la recompiler.



# Ressources

---

- ▶ Traduction des chaînes (localisation)
  - ▶ Lorsque les textes sont définis dans *res/values/strings.xml*, il suffit de faire des copies du dossier values, en values-us, values-fr, values-de, etc. et de traduire les textes en gardant les attributs name.
  - ▶ Voici par exemple res/values-de/strings.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="main_menu">Hauptmenü</string>
    <string name="action_settings">Einstellungen</string>
    <string name="bonjour">Guten Tag</string>
</resources>
```

- ▶ Le système android ira chercher automatiquement le bon texte en fonction des paramètres linguistiques configurés par l'utilisateur.
- 



# Ressources

---

- ▶ Emploi des ressources texte dans un programme
  - ▶ Dans un programme Java, on peut très facilement placer un texte dans une vue de l'interface :

```
TextView tv = ... // ... voir plus loin pour les vues  
tv.setText(R.string.bonjour);
```

- ▶ *R.string.bonjour* désigne le texte dans le fichier `res/values*/strings.xml`
- ▶ Cela fonctionne car `TextView.setText()` a deux surcharges :
  - ▶ `void setText(String text)` : on peut fournir une chaîne quelconque
  - ▶ `void setText(int idText)` : on doit fournir un identifiant de ressource chaîne, donc forcément l'un des textes du fichier `res/values/strings.xml`



# Ressources

---

- ▶ Emploi des ressources texte dans un programme, suite
- ▶ Par contre, si on veut récupérer l'une des chaînes des ressources pour l'utiliser dans le programme :

```
String message = getResources().getString(R.string.bonjour);
```

- ▶ ***getResources()*** est une méthode de la classe ***Activity*** qui retourne une représentation de toutes les ressources du dossier *res*.
- ▶ Chacune de ces ressources, selon son type, peut être récupérée avec son *identifiant*.



# Ressources

---

- ▶ Maintenant, dans un fichier de ressources décrivant une interface, on peut également employer des ressources texte :

```
<RelativeLayout>  
    <TextView android:text="@string/bonjour"/>  
    <Button android:text="Commencer" />  
</RelativeLayout>
```

- ▶ Le titre du TextView sera pris dans le fichier de ressource des chaînes,
- ▶ par contre, le titre du Button sera une chaîne fixe hard coded, non traduisible, donc Android Studio mettra un avertissement.

@string/nom est une référence à la chaîne du fichier res/values\*/strings.xml ayant ce nom.



# Ressources

---

- ▶ Images : R.drawable.nom
  - ▶ De la même façon, les images PNG placées dans res/drawable et res/mipmaps-\* sont référençables

```
<ImageView  
    android:src="@drawable/velo"  
    android:contentDescription="@string/mon_velo" />
```

- ▶ La notation @drawable/nom référence l'image portant ce nom dans l'un des dossiers.
- ▶ NB: les dossiers res/mipmaps-\* contiennent la même image à des définitions différentes, pour correspondre à différents téléphones et tablettes.
  - ▶ Ex: mipmap-hdpi contient des icônes en 72x72 pixels.



# Ressources

---

- ▶ Tableau de chaînes : `R.array.nom`
  - ▶ Voici un extrait du fichier `res/values/arrays.xml` :

```
<resources>
  <string-array name="planetes">
    <item>Mercure</item>
    <item>Venus</item>
    <item>Terre</item>
    <item>Mars</item>
    ...
  </string-array>
</resources>
```

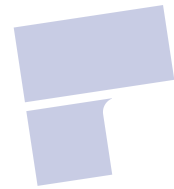
- ▶ Dans le programme Java, il est possible de faire :

```
Resources res = getResources();
String[] planetes = res.getStringArray(R.array.planetes);
```



---

**SEMAINE 3**



pdfelement





# Mise en page (layouts)

# Mise en page (layouts)

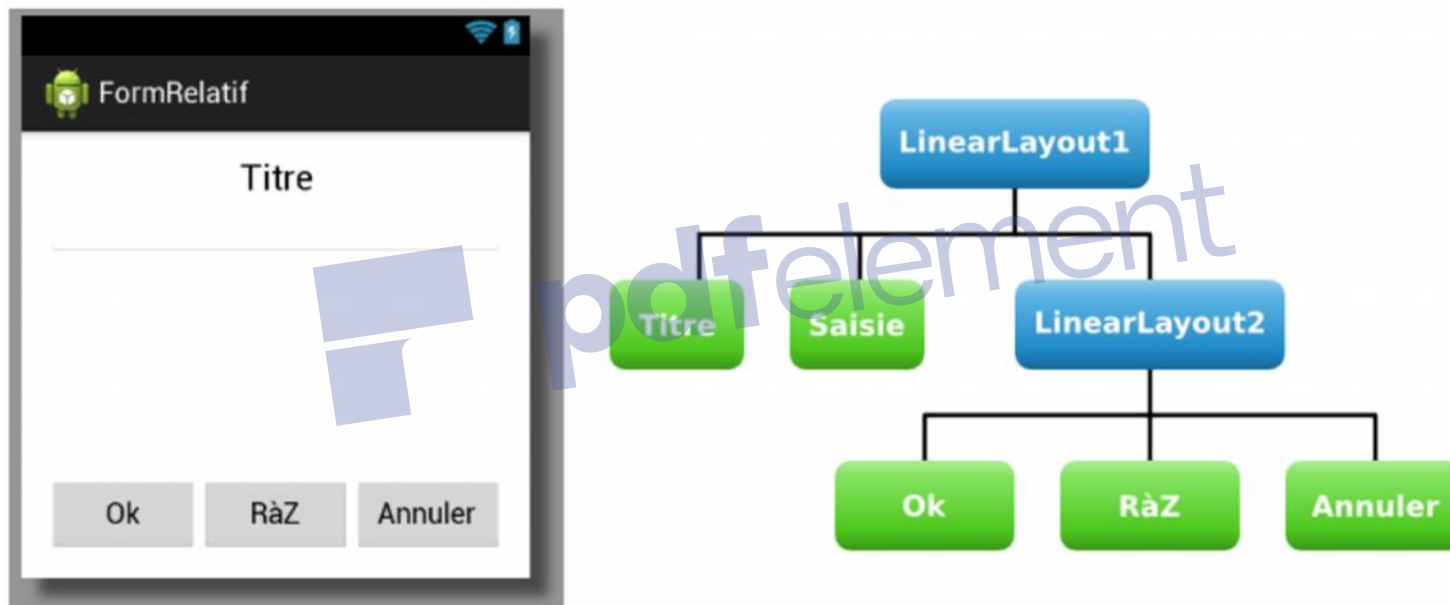
---

- ▶ Un écran Android de type formulaire est généralement composé de plusieurs vues. Entre autres :
  - ▶ TextView, ImageView : titre, image
  - ▶ EditText : texte à saisir
  - ▶ Button, CheckBox : bouton à cliquer, case à cocher
- ▶ Ces vues sont alignées à l'aide de groupes sous-classes de ViewGroup, éventuellement imbriqués :
  - ▶ LinearLayout : positionne ses vues en ligne ou en colonne
  - ▶ RelativeLayout, ConstraintLayout : positionnent leurs vues l'une par rapport à l'autre
  - ▶ TableLayout : positionne ses vues sous forme d'un tableau



# Mise en page (layouts)

- Les groupes et vues forment un arbre :



# Mise en page (layouts)

---

- ▶ Création d'une interface par programme
  - ▶ Il est possible de créer une interface par programme, comme avec JavaFX et Swing, mais c'est assez compliqué :

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    TextView tv = new TextView(this);  
    tv.setText(R.string.bonjour);  
    LinearLayout rl = new LinearLayout(this);  
    LayoutParams lp = new LayoutParams();  
    lp.width = LayoutParams.MATCH_PARENT;  
    lp.height = LayoutParams.MATCH_PARENT;  
    rl.addView(tv, lp);  
    setContentView(rl);  
}
```



# Mise en page (layouts)

---

- ▶ Ressources de type layout
  - ▶ Il est donc préférable de stocker l'interface dans un fichier `res/layout/main.xml` :

```
<LinearLayout ...>  
    <TextView android:text="@string/bonjour" ... />  
</LinearLayout>
```

- ▶ qui est référencé par son identifiant `R.layout.nom_du_fichier` (donc ici c'est `R.layout.main`) dans le programme Java :

```
protected void onCreate(Bundle bundle) {  
    super.onCreate(bundle);  
    setContentView(R.layout.main);  
}
```

- ▶ La méthode `setContentView` fait afficher le layout indiqué.
- 



# Mise en page (layouts)

---

- ▶ Lorsque l'application veut manipuler l'une de ses vues, elle doit utiliser R.id.symbole, ex :

```
TextView tv = findViewById(R.id.message);
```

- ▶ avec la définition suivante dans res/layout/main.xml :

```
<LinearLayout ...>  
    <TextView  
        android:id="@+id/message"  
        android:text="@string/bonjour" />  
</LinearLayout>
```

- ▶ La notation @+id/nom définit un identifiant pour le TextView.
- 



# Mise en page (layouts)

---

## ► @id/nom ou @+id/nom ?

- Dans les fichiers layout.xml, il y a deux notations à ne pas confondre :
  - **@+id/nom** pour définir (créer) un identifiant
  - **@id/nom** pour référencer un identifiant déjà défini ailleurs
- Exemple, le **Button** btn se place sous le **TextView** *titre* :

```
<RelativeLayout xmlns:android="..." ... >
    <TextView ...
        android:id="@+id/titre"
        android:text="@string/titre" />
    <Button ...
        android:id="@+id/btn"
        android:layout_below="@id/titre"
        android:text="@string/ok" />
</RelativeLayout>
```

# Mise en page (layouts)

---

- ▶ Toutes les vues doivent spécifier ces deux attributs :
  - ▶ `android:layout_width` largeur de la vue
  - ▶ `android:layout_height` hauteur de la vue
- ▶ Ils peuvent valoir :
  - ▶ `"wrap_content"` : la vue prend la place minimale
  - ▶ `"match_parent"` : la vue occupe tout l'espace restant
  - ▶ `"valeurdp"` : une taille fixe, ex : `"100dp"` mais c'est peu recommandé, sauf `0dp` pour un cas particulier
- ▶ Les dp sont indépendants de l'écran (explications). 100dp font 100 pixels sur un écran de 160 dpi (160 dots per inch) tandis qu'ils font 200 pixels sur un écran 320 dpi. Ça fait la même taille apparente quelque soit la finesse des pixels.

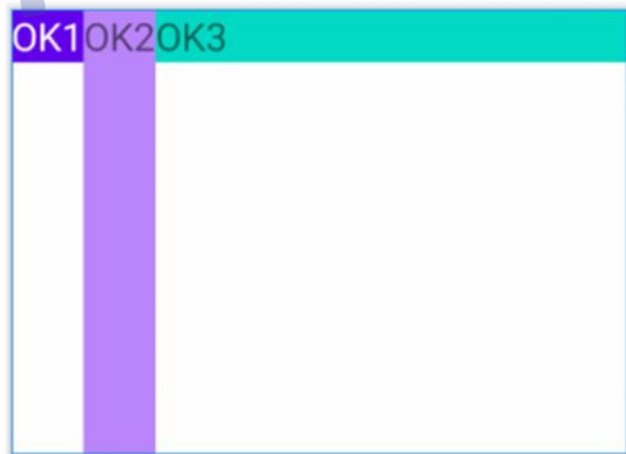




# Mise en page (layouts)

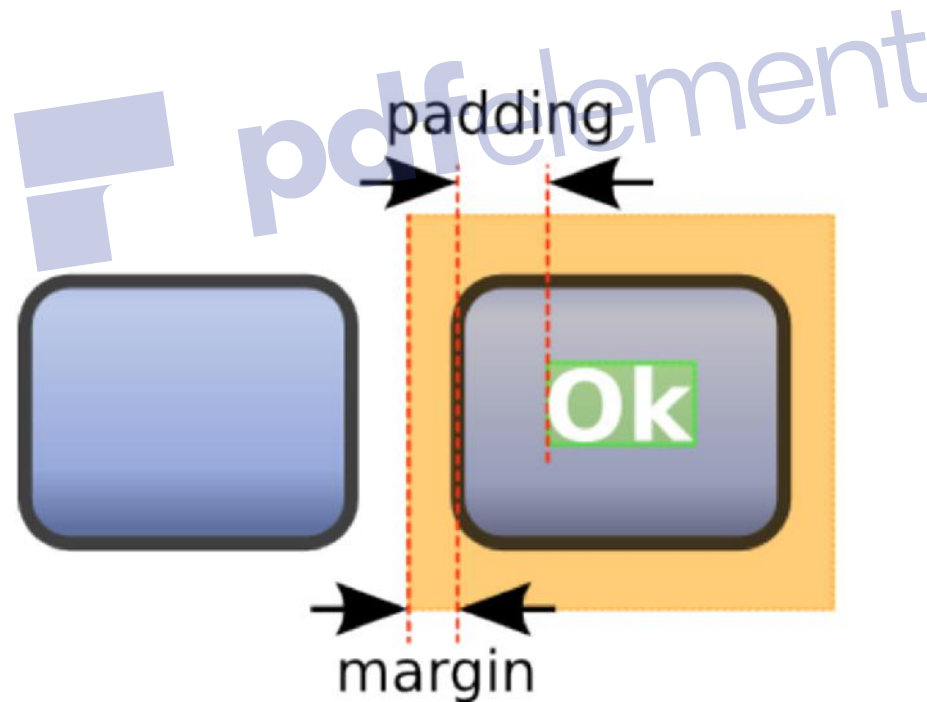
- ▶ Paramètres obligatoires, suite
  - ▶ Par exemple, trois boutons dans un LinearLayout horizontal :

Bouton	layout_width	layout_height
OK1	wrap_content	wrap_content
OK2	wrap_content	match_parent
OK3	match_parent	wrap_content



# Mise en page (layouts)

- ▶ Autres paramètres géométriques
  - ▶ Il est possible de modifier l'espacement des vues :
    - ▶ **Padding** espace entre le texte et les bords, géré par chaque vue
    - ▶ **Margin** espace autour des bords, géré par les groupes



# Mise en page (layouts)

---

- ▶ Marges et remplissage
  - ▶ On peut définir les marges et les remplissages séparément sur chaque bord (Top, Bottom, Left, Right), ou identiquement sur tous :

```
<Button  
    android:layout_margin="10dp"  
    android:layout_marginTop="15dp"  
    android:padding="10dp"  
    android:paddingLeft="20dp" />
```

- ▶ C'est très similaire à CSS.



# Mise en page (layouts)

---

- ▶ Groupe de vues LinearLayout
  - ▶ Il range ses vues soit horizontalement, soit verticalement

```
<LinearLayout android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <Button android:text="Ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button android:text="Annuler"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

- ▶ Il faut seulement définir l'attribut android:orientation à "horizontal" ou "vertical".



# Mise en page (layouts)

- ▶ **LinearLayout** : Il range ses vues soit **verticalement**, soit horizontalement

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:layout_width="368dp"
    android:layout_height="495dp"
    android:orientation="vertical"

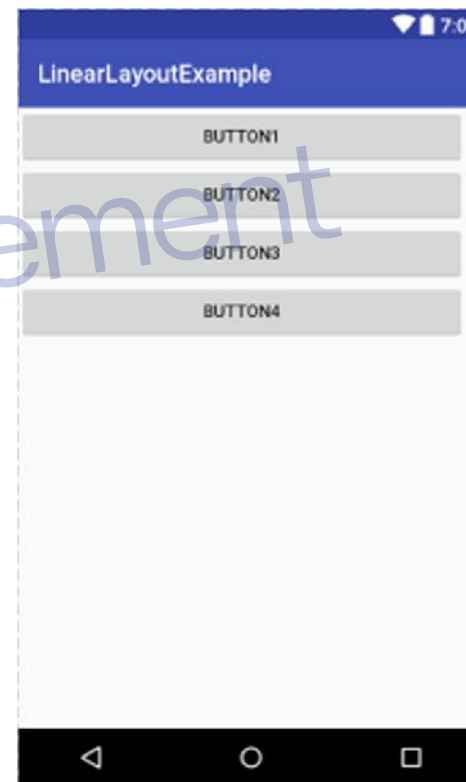
    <Button
        android:id="@+id/button5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button1" />

    <Button
        android:id="@+id/button6"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button2" />

    <Button
        android:id="@+id/button7"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button3" />

    <Button
        android:id="@+id/button8"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button4" />

</LinearLayout>
```



# Mise en page (layouts)

## ► LinearLayout : horizontalement

```
<LinearLayout android:layout_width="368dp"
    android:layout_height="495dp"
    android:orientation="horizontal"

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button4" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button3" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button2" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button1" />

</LinearLayout>
```



# Mise en page (layouts)

---

- ▶ Groupe de vues `TableLayout`
  - ▶ C'est une variante du `LinearLayout` : les vues sont rangées en lignes de colonnes bien alignées. Il faut construire une structure XML comme celle-ci. Voir sa doc Android.

```
<TableLayout ...>
  <TableRow>
    <vue 1.1 .../>
    <vue 1.2 .../>
  </TableRow>
  <TableRow>
    <vue 2.1 .../>
    <vue 2.2 .../>
  </TableRow>
</TableLayout>
```



# Mise en page (layouts)

- ▶ **TableLayout:** C'est une variante du LinearLayout où les vues sont rangées en lignes de colonnes bien tabulées.

```
<TableLayout ...>
  <TableRow>
    <item 1.1 .../>
    <item 1.2 .../>
  </TableRow>
  <TableRow>
    <item 2.1 .../>
    <item 2.2 .../>
  </TableRow>
</TableLayout>
```





# Mise en page (layouts)

---

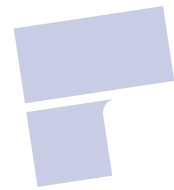
- ▶ Largeur des colonnes d'un TableLayout
  - ▶ Ne pas spécifier android:layout\_width dans les vues d'un TableLayout, car c'est obligatoirement toute la largeur du tableau. Seul la balise exige cet attribut.
  - ▶ Deux propriétés intéressantes permettent de rendre certaines colonnes étirables. Fournir les numéros (première = 0).
    - ❑ android:stretchColumns : numéros des colonnes étirables
    - ❑ android:shrinkColumns : numéros des colonnes reductibles

```
<TableLayout  
    android:stretchColumns="1,2"  
    android:shrinkColumns="0,3"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" >
```



---

**SEMAINE 4**



pdfelement



---

# Composant d'une interface



# Composant d'une interface

---

- ▶ Vues

- ▶ **TextView**

- ▶ Le plus simple, il affiche un texte statique, comme un titre. Son libellé est dans l'attribut android:text.

```
<TextView  
    android:id="@+id/tvTitre"  
    android:text="@string/titre"  
    ... />
```

- ▶ On peut le changer dynamiquement :

```
TextView tvTitre = findViewById(R.id.tvTitre);  
tvTitre.setText("blablaba");
```



# Composant d'une interface

---

## ► Vues

### ► **Button**

- L'une des vues les plus utiles est le Button :

```
<Button  
    android:id="@+id/btnOk"  
    android:text="@string/ok"  
    ... />
```

- En général, on définit un identifiant pour chaque vue active, ici :  
 android:id="@+id/btnOk"
  - Son titre est dans l'attribut android:text.
  - Voir la semaine prochaine pour son activité : réaction à un clic.



# Composant d'une interface

---

- ▶ Vues

- ▶ **EditText**

- ▶ Un EditText permet de saisir un texte :

```
<EditText  
    android:id="@+id/email_address"  
    android:inputType="textEmailAddress"  
    ... />
```

- ▶ L'attribut android:inputType spécifie le type de texte : adresse, téléphone, etc. Ça définit le clavier qui est proposé pour la saisie.



# Gestion d'événements déclaratifs et programmématiques

# Gestion d'événements déclaratifs

---

- ▶ Le processus de gestion des événements de manière déclarative peut être décomposé comme suit.
  1. Définissez l'objet de vue (par exemple, une vue Bouton).
  2. Choisissez l'événement auquel vous souhaitez que le programme réponde: par exemple, un événement click (certains objets de vue peuvent répondre à une plage d'événements: clic long, glisser)
  3. Associez l'événement à une méthode Java: par exemple, ajoutez l'attribut onClick sur l'élément XML de la vue Bouton android: onClick = "sayHello".
  4. Implémentez la méthode Java dans le fichier de programme principal (MainActivity.java). Le nom de la méthode doit être identique à celui défini dans l'inspecteur d'attributs.





## ► Fichier XML

```
<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="sayHello"
    android:text="Button" />
```

## ► Code Java

```
public class MainActivity extends Activity {

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

    void sayHello(View v) {
        System.out.println("Hello");
    }
}
```

# Gestion d'événements programmatiques

---

- Pour gérer les événements par programme, les étapes peuvent être décomposées comme suit.

1. Définissez l'objet de vue (par exemple, une vue Bouton)
2. Dans le programme principal, déclarez qu'une variable contient un objet de la vue Bouton

**Button objButton;**

3. Obtenez une référence par programme à l'objet de vue Button défini dans le fichier de présentation

**objButton = (Button) findViewById (R.layout.button).**

4. Choisissez l'événement auquel vous souhaitez répondre et spécifiez l'objet écouteur correspondant (par exemple, Cliquez).

**objButton.setOnClickListener (new View.OnClickListener () {});**

1. Remplacez la méthode onClick:

**void onClick (Afficher v) {// fait quelque chose ici}**



# Gestion d'événements programmatiques

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
  
    protected void onCreate(Bundle savedInstanceState) {  
  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.activity_main);  
  
        Button objButton = (Button) findViewById(R.id.button);  
        assert objButton != null;  
  
        objButton.setOnClickListener(new View.OnClickListener() {  
  
            @Override  
  
            public void onClick(View view) {  
  
                System.out.println("Hello World");  
  
            }  
        });  
    }  
}
```

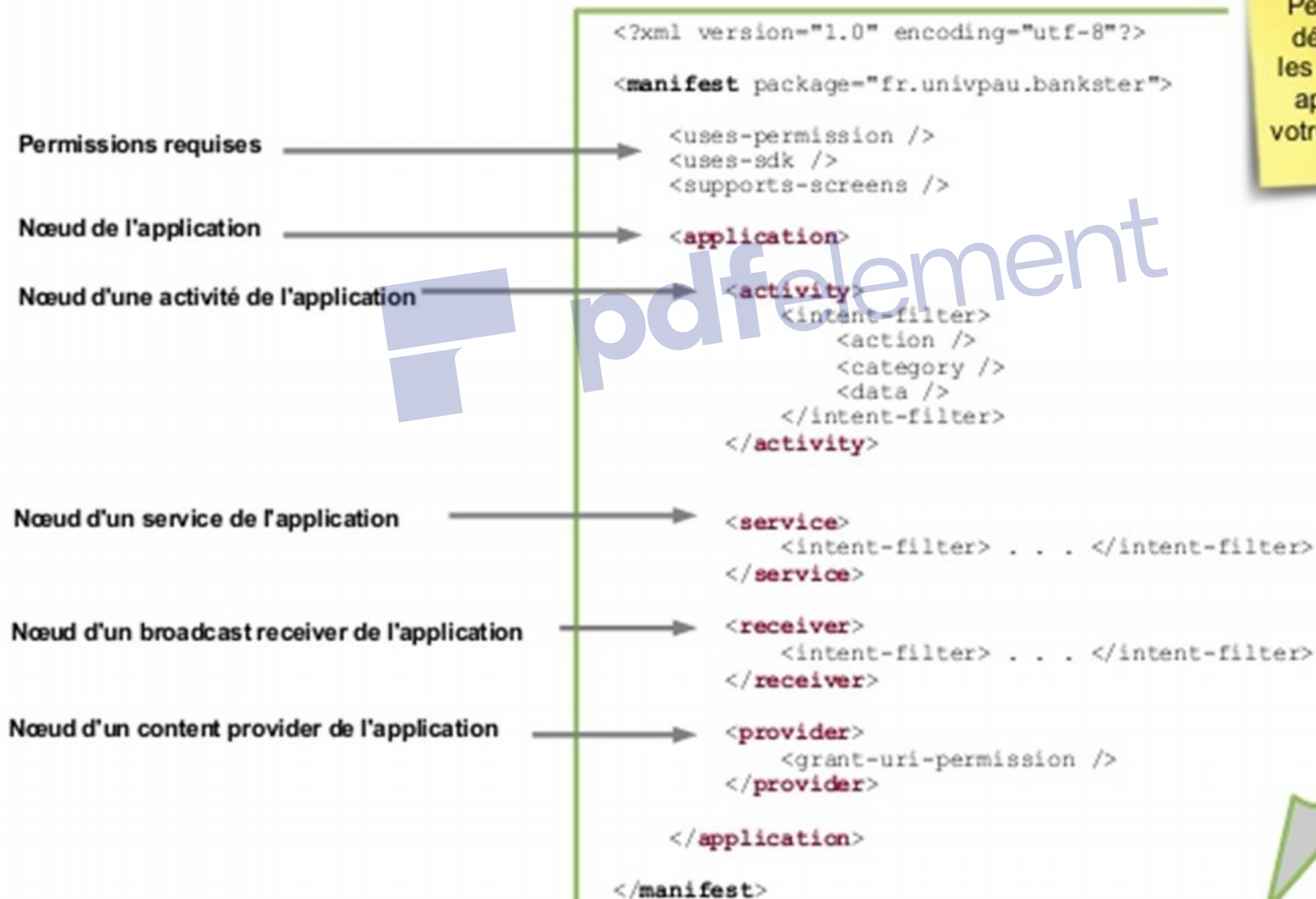
# Une application android

---

- ▶ Activités : couche de présentation (interface graphique) écrans que voit l'utilisateur + gestion des interactions.
- ▶ Services : tâches de fond invisibles p. ex. lecture de musique, téléchargements en cours, calculs longs ou surveillance du téléphone.
- ▶ Fournisseurs de contenu : gestion des données persistantes interface base de donnée, partage de données entre applications.
  - ▶ Exemples typiques : la liste des contacts, la galerie photos.
- ▶ Broadcast receivers : écoute d'événements globaux ou spécifiques p. ex. batterie faible, extinction de l'écran, nouvelle photo. Sert surtout de point d'entrée pour les autres composants.
- ▶ Notifications utilisateur : alerte les utilisateurs sans prendre le focus (téléchargement terminé, mise à jour disponible, etc.).
- ▶ Intents : Outil de communication entre les activités.
  - ▶ Permet 'a une activité de formuler publiquement une demande, par exemple prendre une photo, récupérer un numéro, etc. ou lancer une autre activité déterminée.
- ▶ Fichier de manifest : La glue entre tous ces composants (liste des activités, des services, etc.). Décrit leur propriétés et leurs relations.
  - ▶ Indique aussi les exigences de l'application (matériel, version, permissions)



# AndroidManifest.xml



# Les Intents

# Introduction aux intents

---

- ▶ Une application est composée d'une ou plusieurs activités.
- ▶ Chacune gère un écran d'interaction avec l'utilisateur et est définie par une classe Java héritant de Activity.
- ▶ Les vues d'une activité (boutons, menus, actions) permettent d'aller sur une autre activité.
- ▶ Le bouton back < permet de revenir sur une précédente activité.
  - ▶ C'est la navigation entre activités.
- ▶ Une application complexe peut aussi contenir :
  - ▶ des services: ce sont des processus qui tournent en arrière-plan,
  - ▶ Des fournisseurs de contenu: ils représentent une sorte de base de données (ex: contacts, . . . ),
  - ▶ des récepteurs d'annonces: pour gérer des messages envoyés d'une application à une autre (ex: notifications, . . . ).



# Déclaration d'une application

---

- ▶ Le fichier AndroidManifest.xml déclare les éléments d'une application, avec un '.' devant le nom de classe des activités:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application ...>
        <activity android:name=".MainActivity" ... />
        <activity android:name=".ConfigActivity" ... />
        <activity android:name=".DessinActivity" ... />
        ...
    </application>
</manifest>
```

- ▶ Une activité qui n'est pas déclarée dans le manifeste ne peut pas être lancée (ActivityNotFoundException).





# Démarrage d'une application

---

- L'une des activités est désignée comme étant « principale », démarrable de l'extérieur, grâce à un sous-élément `<intent-filter>`

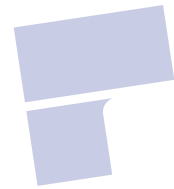
```
<activity android:name=".MainActivity" ...>
  <intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity>
```

- Un `<intent-filter>` déclare les conditions de démarrage d'une activité. Celui-ci indique l'activité principale, celle qu'il faut lancer quand on clique sur son icône.
- Les activités sont démarrées à l'aide d'intents.
- Un Intent contient une demande destinée à une activité, par exemple, composer un numéro de téléphone ou lancer l'application



---

**SEMAINE 5-6**



pdfelement



# Une application android

---

- ▶ Activités : couche de présentation (interface graphique) écrans que voit l'utilisateur + gestion des interactions.
- ▶ Services : tâches de fond invisibles p. ex. lecture de musique, téléchargements en cours, calculs longs ou surveillance du téléphone.
- ▶ Fournisseurs de contenu : gestion des données persistantes interface base de donnée, partage de données entre applications.
  - ▶ Exemples typiques : la liste des contacts, la galerie photos.
- ▶ Broadcast receivers : écoute d'événements globaux ou spécifiques p. ex. batterie faible, extinction de l'écran, nouvelle photo. Sert surtout de point d'entrée pour les autres composants.
- ▶ Notifications utilisateur : alerte les utilisateurs sans prendre le focus (téléchargement terminé, mise à jour disponible, etc.).
- ▶ Intents : Outil de communication entre les activités.
  - ▶ Permet 'a une activité de formuler publiquement une demande, par exemple prendre une photo, récupérer un numéro, etc. ou lancer une autre activité déterminée.
- ▶ Fichier de manifest : La glue entre tous ces composants (liste des activités, des services, etc.). Décrit leur propriétés et leurs relations.
  - ▶ Indique aussi les exigences de l'application (matériel, version, permissions)



# Les Intents (intentions)

---

- ▶ Principe des Intents
- ▶ Types d'intents
  - ▶ Intents implicites
  - ▶ Intents Explicite
- ▶ Intents Explicite
  - ▶ Intents pour une nouvelle activité
  - ▶ **Retour d'une activité**



## Intents Explicite

- ▶ Les applications sont rarement limitées à une unique activité avec un unique visuel utilisateur
  - ▶ **contenant plusieurs activités**
- ▶ une nouvelle activité revient à créer une nouvelle classe héritant de Activity
  - ▶ lui associer un visuel et la déclarer dans le *Manifest*.



```
interface_2.xml x Activity2.java x  
  
    android:theme="@style/Theme.L3SI_Test">  
    <activity android:name=".Activity2"></activity>  
    <activity android:name=".MainActivity">  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
  
            <category android:name="android.intent.category.LAUNCHER" />  
        </intent-filter>  
    </activity>
```

# Intents Explicite

---

- ▶ Déclarer une deuxième activité la rend callable via un intent **explicite**
- ▶ Pour l'OS, lancer une activité consiste à créer une nouvelle fenêtre, à la placer sur le dessus de la pile des fenêtres et à lui passer la main.
  - ▶ Lorsque celle-ci sera finie, l'activité précédente reprendra la main.
- ▶ Pour lancer une activité on doit passer par un **appel système** et un **descripteur d'intent**
- ▶ Un Intent est un objet qui permet de décrire le composant que l'on souhaite lancer.



# Intents Explicite

---

- ▶ Un Intent est un objet qui permet de décrire le composant que l'on souhaite lancer.
- ▶ Si on passe la main à une activité interne à l'application, on peut créer l'Intent et passer la classe de l'activité ciblée par l'Intent:

```
Intent login = new Intent(Source.this, Destination.class);  
startActivity(login);
```



# Intents Explicite

---

## ► Lancement d'une activité avec paramètres

- Lorsqu'on lance une activité, nous pouvons vouloir lui transmettre des informations.
  - Il est donc nécessaire de les ajouter dans l'Intent avant le lancement de l'activité.
  - Les données complémentaires circulant dans un Intent sont appelées les extras et sont des couples (*clef, valeur*).
  - Syntaxiquement, il suffit de choisir un nom de paramètre (de clef) et de faire appel à la méthode `putExtra` pour ajouter l'information dans l'appel :

```
Intent i = new Intent(Srouce.this, Dest.class);  
i.putExtra(Dest.EXTRA_PARAM1, "une chaîne");  
i.putExtra(Dest.EXTRA_PARAM2, 42);  
startActivity(i);
```





# Intents Explicite

---

- ▶ Pour récupérer une valeur reçue par l'activité appelée, il suffit de récupérer une copie de l'Intent d'appel avec ***getIntent*** et d'y lire les **extras**,
  - ▶ via les méthodes getXXXExtra dédiées aux différents types de paramètre :

```
Intent intent = getIntent();  
String param1 = intent.getStringExtra(EXTRA_PARAM1);  
int param2 = intent.getIntExtra(EXTRA_PARAM2, defaultValue: 0); // on précise la valeur par défaut si paramètre non pr
```

- ▶ **Convention** : les noms des clefs des extras doivent être définis en **constantes** dans le composant acceptant cette information en entrée.
    - ▶ Le **nom** de la **constante** doit commencer par EXTRA\_
    - ▶ et le **nom** des **clefs** doit être **pleinement qualifié**
- 



# Intents implicites

---

- ▶ Intents implicites : Il ne définit pas une cible précise.

- ▶ Exemple 1:

- ▶ je veux faire un appel téléphonique, le système va chercher une application qui me permet de le faire.

```
String phone_number = «0606060606 »
```

```
Intent secondeActivite = new Intent(Intent.ACTION_DIAL, Uri.parse(phone_number));  
startActivity(secondeActivite);
```

- ▶ Action à réaliser

- ▶ constantes ACTION\_\* dans Intent pour les actions systèmes:  
exemple : *Intent.ACTION\_DIAL*

- ▶ Donnée sur laquelle l'action est réalisée

- ▶ Exemple : *tel:0606060606*



# Intents implicites

---

- ▶ Exemple 2,
  - ▶ voici comment ouvrir le navigateur sur un URL :

```
String url = "http://staff.univ-batna2.dz/drid_hamza/home"  
intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));  
startActivity(intent);
```

- ▶ L'actionVIEW avec un URI(généralisation d'un URL) est interprétée par Android, cela fait ouvrir automatiquement le navigateur.



---

## SEMAINE 7



# Barre d'action et menus



## Barre d'action (**Actions bar**)

- ▶ La barre d'action contient l'icône d'application (1), quelques items de menu (2) et un bouton . . . pour avoir les autres menus (3).



# Barre d'action (**Actions bar**)

## ► Déclarer notre menu

- Première étape, il faut déclarer nos actions (aussi nommées options), cela se fait dans des fichiers nommés « menu ».
- A la façon des layout, il faut les placer dans nos ressources. Pour cela, créez un dossier menu dans res (si il n'existe pas déjà).
- Puis créez un fichier res/menu/my\_menu.xml

menu\_test.xml

```
01 <menu xmlns:android="http://schemas.android.com/apk/res/android"
02     xmlns:app="http://schemas.android.com/apk/res-auto">
03
04     <item
05         android:id="@+id/action_save"
06         android:icon="@android:drawable/ic_menu_save"
07         android:title="@string/action_save"
08         app:showAsAction="ifRoom" />
09
10     <item
11         android:id="@+id/action_delete"
12         android:icon="@android:drawable/ic_menu_delete"
13         android:title="@string/action_delete"
14         app:showAsAction="ifRoom" />
15
16     <item
17         android:id="@+id/action_settings"
18         android:title="@string/action_settings"
19         app:showAsAction="never" />
20 </menu>
```

# Actions

---

- ▶ Les actions sont définis dans un tag nommé **menu**.
  - ▶ Chaque entrée est ensuite définit par un item, qui correspond à une action qui sera ajoutée à l'ActionBar.
  - ▶ On a utilisé les drawables fournis par le SDK android (@android:drawable/XXX) , mais il est aussi possible d'utiliser nos images personnalisées (@drawable/XXX).
- ▶ Un item possède
  - ▶ id : un identifiant unique
  - ▶ icon : une image (pointe vers un drawable)
  - ▶ title : une titre, utilisé quand aucun icon n'est définit, ou lorsque l'on effectue un long click sur l'item
  - ▶ showAsAction : le mode d'affichage de l'item





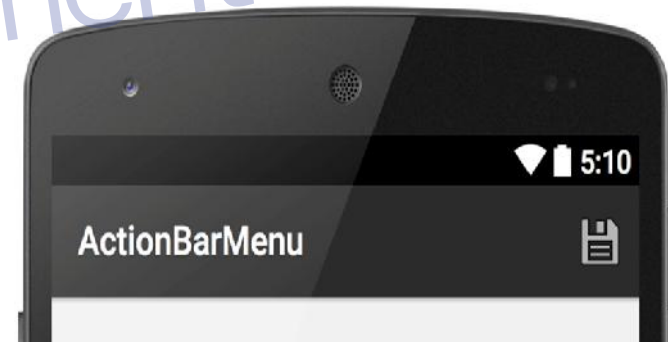
# showAsAction

## ► ifRoom

- Si vous définissez le showAsAction à **ifRoom**, l'item sera affiché directement dans l'ActionBar, en tant que « bouton clickable ».

Voici un exemple :

```
01 <menu xmlns:android="http://schemas.android.com/apk/res/android"
02     xmlns:app="http://schemas.android.com/apk/res-auto">
03
04     <item
05         android:id="@+id/action_save"
06         android:icon="@android:drawable/ic_menu_save"
07         android:title="@string/action_save"
08         app:showAsAction="ifRoom" />
09
10 </menu>
```

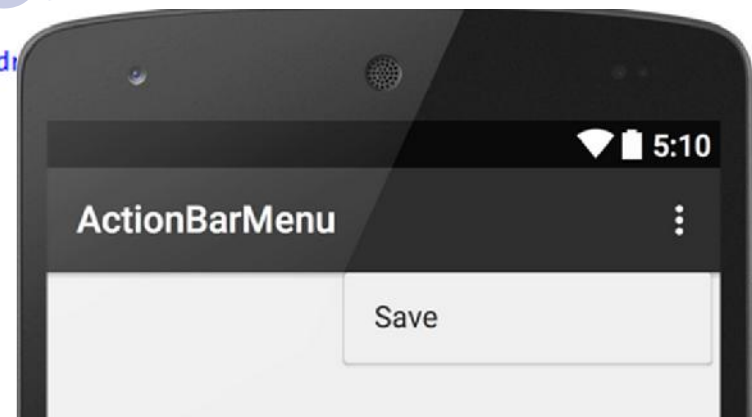


# showAsAction

## ► Never

- Si vous définissez le showAsAction à **never**, l'item sera affiché dans l'overflow (les 3 petits points qui indiquent que des actions sont disponibles). Après l'ouverture de cet overflow, l'item sera affiché en temps que texte clickable.

```
01 <menu xmlns:android="http://schemas.android.com/apk/res/android"
02     xmlns:app="http://schemas.android.com/apk/res-auto">
03
04     <item
05         android:id="@+id/action_save"
06         android:icon="@android:drawable/ic_menu_save"
07         android:title="@string/action_save"
08         app:showAsAction="never" />
09
10 </menu>
```



# Actions bar

## ► Utiliser depuis Activité

- Dans votre activité, afin d'utiliser **menu\_test.xml** en tant qu'icônes de votre ActionBar, il suffit de l'indiquer à votre menuInflater lors du **onCreateOptionsMenu** (fonction d'Activité).

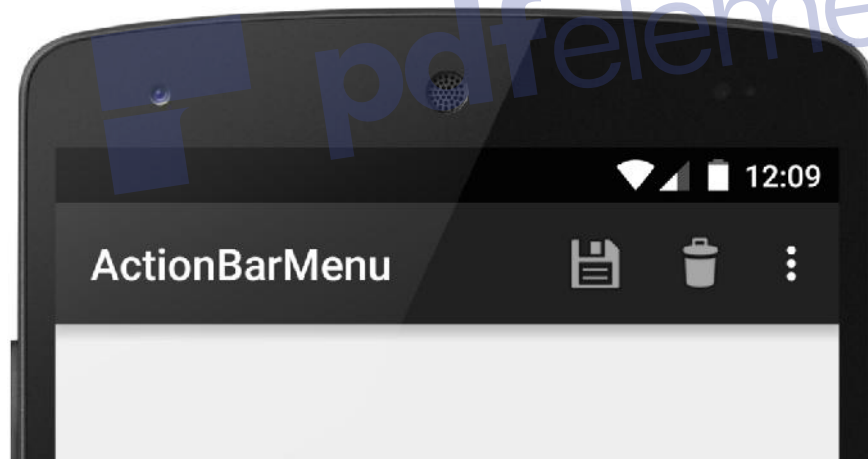
MainActivity.java

```
01 public class MainActivity extends AppCompatActivity {  
02  
03     @Override  
04     protected void onCreate(Bundle savedInstanceState) {  
05         super.onCreate(savedInstanceState);  
06         setContentView(R.layout.activity_main);  
07     }  
08  
09     @Override  
10     public boolean onCreateOptionsMenu(Menu menu) {  
11         //ajoute les entrées de menu_test à l'ActionBar  
12         getMenuInflater().inflate(R.menu.menu_test, menu);  
13         return true;  
14     }  
15 }
```

# Actions bar

---

- ▶ Ici, **getMenuInflater().inflate(R.menu.menu\_test, menu);** ajoutera nos 3 items.
- ▶ Si vous exécutez l'application, vous devriez voir ceci



# Actions bar

---

## ► Gérer le click

- Maintenant, pour récupérer le click de l'utilisateur sur une action, il faut surcharger la méthode d'activité **public boolean onOptionsItemSelected(MenuItem item)**
- En effectuant un switch sur l'identifiant de l'item, vous pouvez savoir lequel a été sélectionné.

```
//gère le click sur une action de l'ActionBar
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()){
        case R.id.action_save:
            save();
            return true;
        case R.id.action_delete:
            delete();
            return true;
        case R.id.action_settings:
            return true;
    }

    return super.onOptionsItemSelected(item);
}
```

# Actions bar (Exemple complet)

## MainActivity.java

```
01 public class MainActivity extends AppCompatActivity {
02
03     @Override
04     protected void onCreate(Bundle savedInstanceState) {
05         super.onCreate(savedInstanceState);
06         setContentView(R.layout.activity_main);
07     }
08
09     private void save(){
10         Toast.makeText(this,R.string.action_save,Toast.LENGTH_LONG).show();
11     }
12
13     private void delete(){
14         Toast.makeText(this,R.string.action_delete,Toast.LENGTH_LONG).show();
15     }
16
17     @Override
18     public boolean onCreateOptionsMenu(Menu menu) {
19         //ajoute les entrées de menu_test à l'ActionBar
20         getMenuInflater().inflate(R.menu.menu_test, menu);
21         return true;
22     }
23
24     //gère le click sur une action de l'ActionBar
25     @Override
26     public boolean onOptionsItemSelected(MenuItem item) {
27         switch (item.getItemId()){
28             case R.id.action_save:
29                 save();
30                 return true;
31             case R.id.action_delete:
32                 delete();
33                 return true;
34             case R.id.action_settings:
35                 return true;
36         }
37         return super.onOptionsItemSelected(item);
38     }
39 }
```

# LES LISTVIEWS



# LISTVIEWS

---

- ▶ Stockage d'une liste
- ▶ Affichage d'une liste, adaptateurs
- ▶ Consultation et édition d'un item





# LISTVIEWS

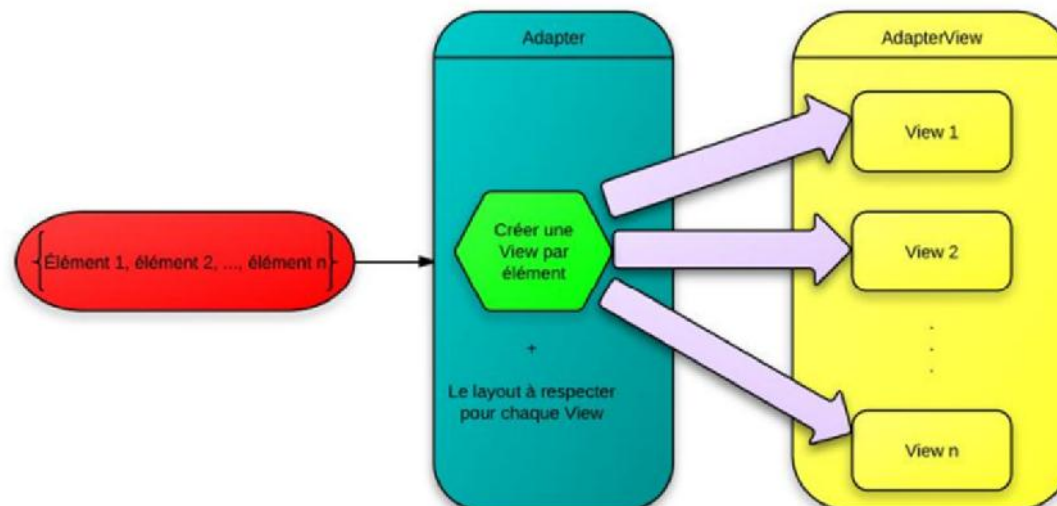
---

- ▶ L'objectif est d'afficher une liste d'un type d'objet particulier, des articles commerciaux par exemple.
- ▶ Il existe plusieurs paramètres à prendre en compte dans ce cas-là.
- ▶ Tout d'abord, quelle est l'information à afficher
  - ▶ pour chaque article?
  - ▶ Le nom?
  - ▶ Le prix?
  - ▶ La quantité ?
  - ▶ Et que faire quand on clique sur un élément de la liste ?



# LISTVIEWS

- ▶ Gestion des listes de données
  - ▶ Le comportement typique pour afficher une liste depuis un ensemble de données est celui-ci :
    - ▶ On donne à l'adaptateur une liste d'éléments à traiter et la manière dont ils doivent l'être,
    - ▶ On passe cet adaptateur à un AdapterView.
    - ▶ Dans l'AdapterView, l'adaptateur va créer un widget pour chaque élément en fonction des informations fournies en amont.



# LISTVIEWS

---

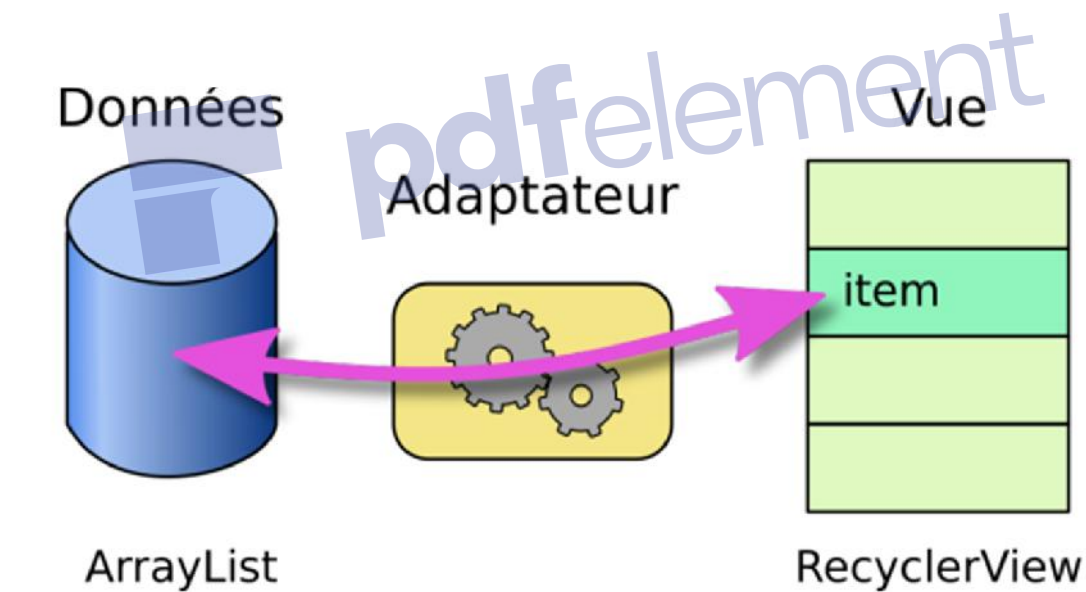
- ▶ La gestion des listes se divise en deux parties distinctes.
- ▶ Les **Adapter** qui sont les objets qui gèrent les données, mais pas leur affichage ou leur comportement en cas d'interaction avec l'utilisateur.
  - ▶ On peut considérer un adaptateur comme une intermédiaire entre les données et la vue qui représente ces données.
- ▶ Les **AdapterView**, qui vont gérer l'affichage et l'interaction avec l'utilisateur, mais sur lesquels on ne peut pas effectuer d'opération de modification des données.



# LISTVIEWS

## ► Schéma global

- Modèle MVC : le contrôleur entre les données et la vue s'appelle un adaptateur.



# LISTVIEWS

---

- Pour commencer, il faut représenter les données :

```
public class Planete {  
    public String nom;        // nom de la planète  
    public int distance;     // distance au soleil en Gm  
  
    Planete(String nom, int distance) {  
        this.nom = nom;  
        this.distance = distance;  
    }  
}
```

- Lui rajouter tous les accesseurs (getters) et modificateurs (setters) pour en faire un JavaBean : objet Java simple (POJO) composé de variables membres privées initialisées par le constructeur, et d'accesseurs



# LISTVIEWS

---

- ▶ Deux solutions pour initialiser la liste avec des valeurs prédéfinies :
  - ▶ Un tableau dans les ressources.
  - ▶ Un tableau constant Java comme ceci :

```
final Planete[] initdata = {  
    new Planete("Mercure", 58),  
    new Planete("Vénus", 108),  
    new Planete("Terre", 150),  
    ...  
};
```

- ▶ final signifie constant, initdata ne pourra pas être réaffecté (par contre, ses cases peuvent être réaffectées).



# LISTVIEWS

---

- L'étape suivante consiste à recopier les valeurs initiales dans un tableau dynamique de type ArrayList :

```
private List<Planete> liste;  
  
void onCreate(...)  
{  
    ...  
  
    // copie du tableau initdata dans le ArrayList  
    liste = new ArrayList<>(Arrays.asList(initdata));  
}
```

- NB: Arrays.asList crée une liste non modifiable, c'est pour ça qu'on la recopie dans un ArrayList.



# LISTVIEWS

---

- ▶ Quelques méthodes utiles de la classe abstraite List, héritées par ArrayList :
  - ▶ `liste.size()` : retourne le nombre d'éléments présents,
  - ▶ `liste.clear()` : supprime tous les éléments,
  - ▶ `liste.add(elem)` : ajoute cet élément à la liste,
  - ▶ `liste.remove(elem ou indice)` : retire cet élément
  - ▶ `liste.get(indice)` : retourne l'élément présent à cet indice,
  - ▶ `liste.contains(elem)` : true si elle contient cet élément,
  - ▶ `liste.indexOf(elem)` : indice de l'élément, s'il y est.





# LISTVIEWS

---

- ▶ Un tableau dans les ressources.
- ▶ On crée deux tableaux dans le fichier res/values/arrays.xml :

```
<resources>
  <string-array name="noms">
    <item>Mercure</item>
    <item>Venus</item>
    ...
  </string-array>
  <integer-array name="distances">
    <item>58</item>
    <item>108</item>
    ...
  </integer-array>
</resources>
```

- ▶ Intérêt : traduire les noms des planètes dans d'autres langues en créant des variantes, ex: res/values-en/arrays.xml



# LISTVIEWS

---

- ▶ Ensuite, on récupère ces ressources tableaux pour remplir le ArrayList :

```
// accès aux ressources
Resources res = getResources();
final String[] noms = res.getStringArray(R.array.noms);
final int[] distances = res.getIntArray(R.array.distances);

// recopie dans le ArrayList
liste = new ArrayList<>();
for (int i=0; i<noms.length; ++i) {
    liste.add(new Planete(noms[i], distances[i]));
}
```



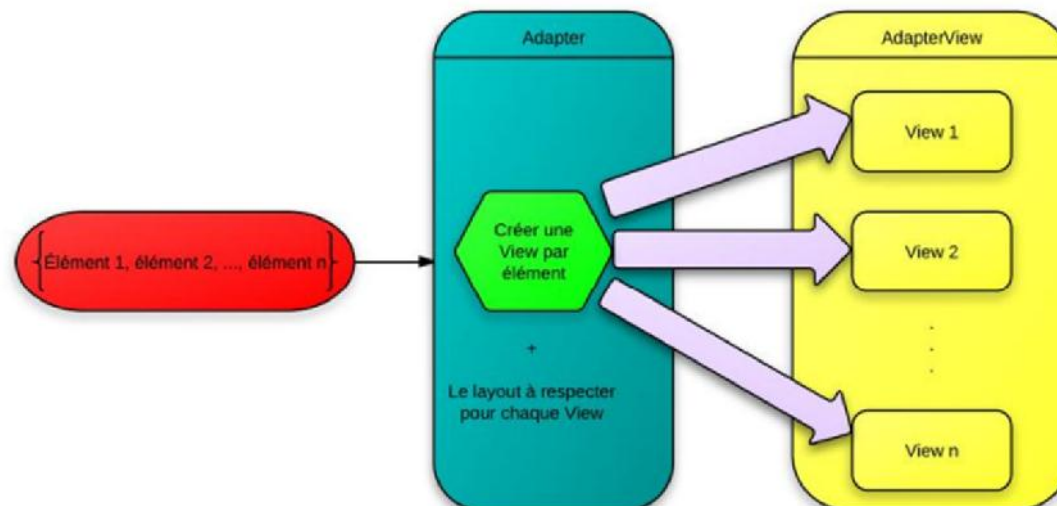
# Semaine 8



## pdfelement

# LISTVIEWS

- ▶ Gestion des listes de données
  - ▶ Le comportement typique pour afficher une liste depuis un ensemble de données est celui-ci :
    - ▶ On donne à l'adaptateur une liste d'éléments à traiter et la manière dont ils doivent l'être,
    - ▶ On passe cet adaptateur à un AdapterView.
    - ▶ Dans l'AdapterView, l'adaptateur va créer un widget pour chaque élément en fonction des informations fournies en amont.



# LISTVIEWS

---

- ▶ An adapter is a bridge between UI component and data source that helps us to fill data in UI component.
  - ▶ It holds the data and send the data to adapter view then view can takes the data from the adapter view and shows the data on different views like as list view, grid view, spinner etc.
- ▶ ListView is a subclass of AdapterView and it can be populated by binding to an Adapter
- ▶ **In android commonly used adapters are:**
  - ▶ Array Adapter
  - ▶ Base Adapter

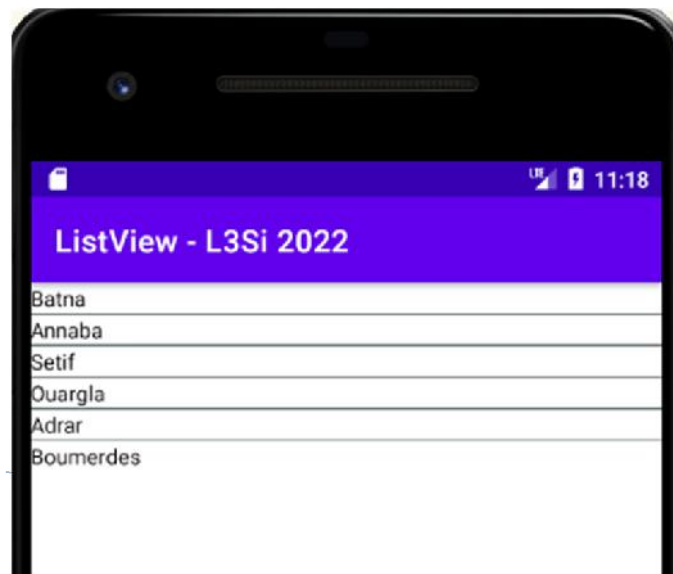


# LISTVIEWS

---

## ► **Array Adapter:**

- Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter.
- Configuration de l'affichage avec un exemple (exemple 1)
  - In this example, we display a list of Wilaya by using simple array adapter. Below is the final output we will create:



# LISTVIEWS

---

- Configuration de l'affichage (exemple 1)
  - **Step 1:** We create a ListView in LinearLayout. Below is the code of our activity\_main.xml

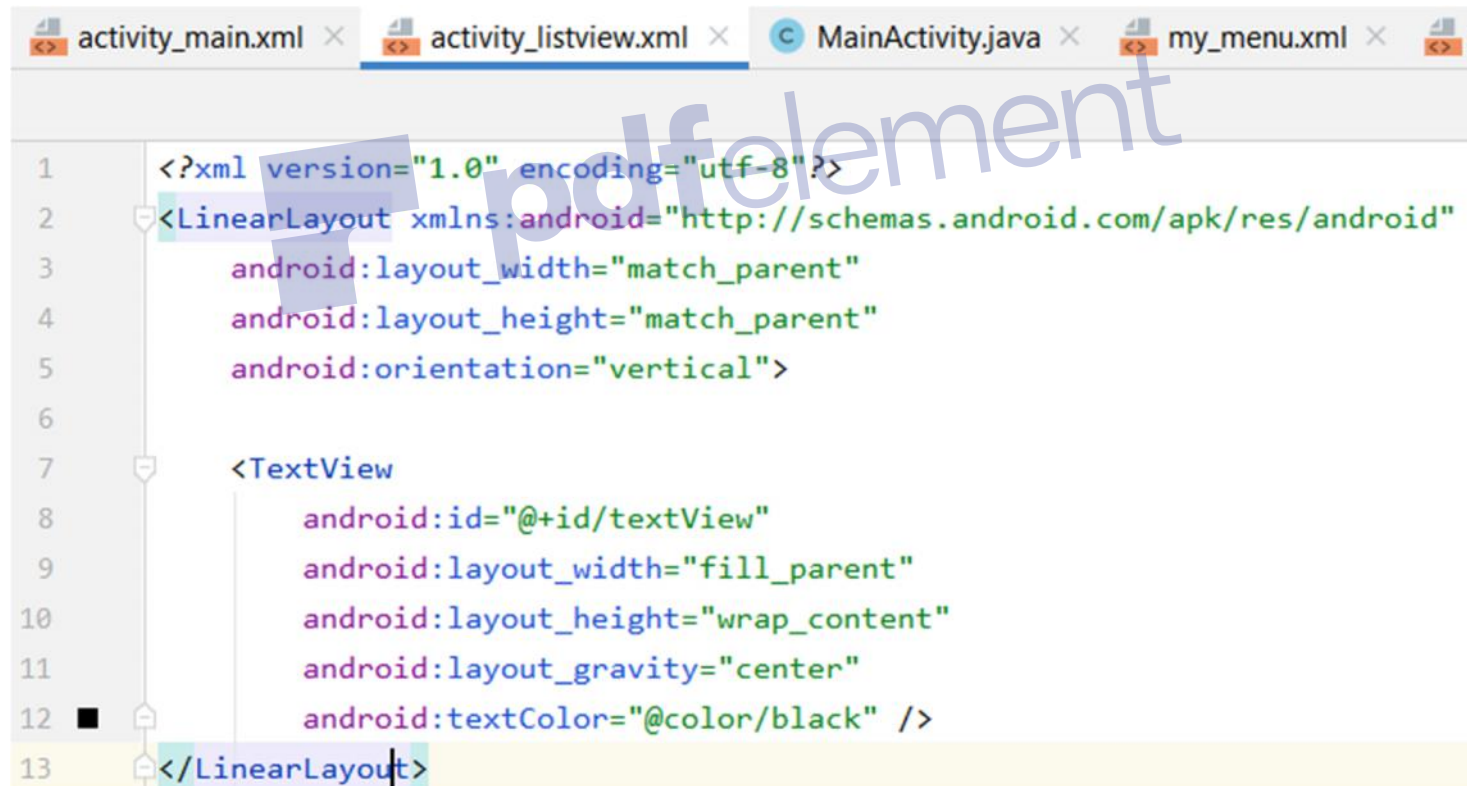
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ListView
        android:id="@+id/simpleListView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:divider="@color/material_blue_grey_800"
        android:dividerHeight="1dp" />

</LinearLayout>
```

# LISTVIEWS

- Configuration de l'affichage (exemple 1)
  - **Step 2:** Create a new layout, name Listview and below is the code of activity\_listview.xml



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="vertical">
6
7     <TextView
8         android:id="@+id/textView"
9         android:layout_width="fill_parent"
10        android:layout_height="wrap_content"
11        android:layout_gravity="center"
12        android:textColor="@color/black" />
13 </LinearLayout>
```



# LISTVIEWS

- Configuration de l'affichage (exemple 1)
  - **Step 3:** Now in this final step we will use ArrayAdapter to display the country names in UI. **Below is the code of MainActivity.java**

```
activity_main.xml x activity_listview.xml x MainActivity.java x my_menu.xml x strings.xml x
1 package com.example.xo;
2
3 import ...
8
9 public class MainActivity extends AppCompatActivity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15         ListView simpleList;
16         String countryList[] = {"India", "China", "australia", "Portugle", "America", "NewZealand"};
17         simpleList = (ListView)findViewById(R.id.simpleListView);
18         ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(context: this, R.layout.activity_listview, R.id.textView, countryList);
19         simpleList.setAdapter(arrayAdapter);
20
21
22
23
24     }
25 }
```

# LISTVIEWS

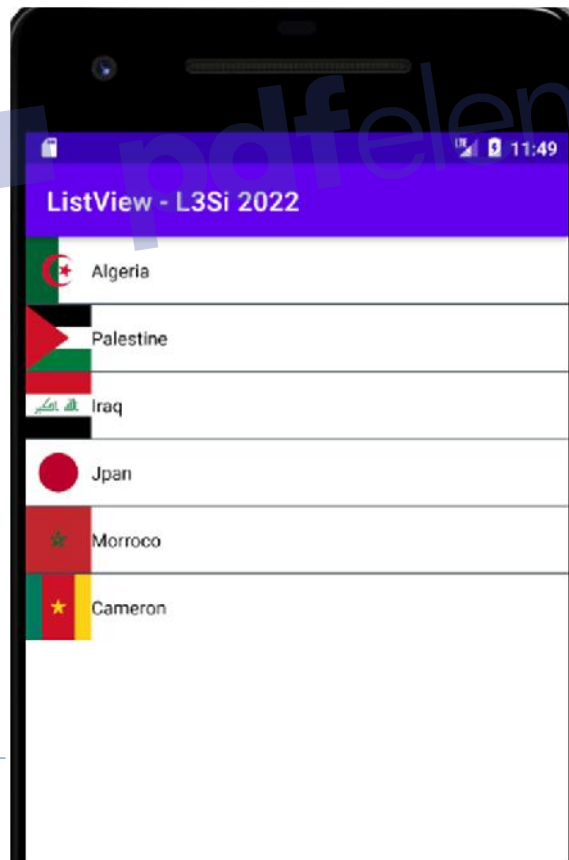
---

- ▶ Configuration de l'affichage (exemple 2)
  - ▶ **Base Adapter:**
    - ▶ Whenever you need a customized list you create your own adapter and extend base adapter in that.
    - ▶ Base Adapter can be extended to create a custom Adapter for displaying a custom list item. ArrayAdapter is also an implementation of BaseAdapter.



# LISTVIEWS

- ▶ Configuration de l'affichage (exemple 2)
  - ▶ **Example of list view using Custom adapter(Base adapter):**
    - In this example we display a list of countries with flags. For this, we have to use custom adapter as shown in example:



# LISTVIEWS

- Configuration de l'affichage (exemple 2)
  - **Step 1:** We create a ListView in LinearLayout. Below is the code of our activity\_main.xml



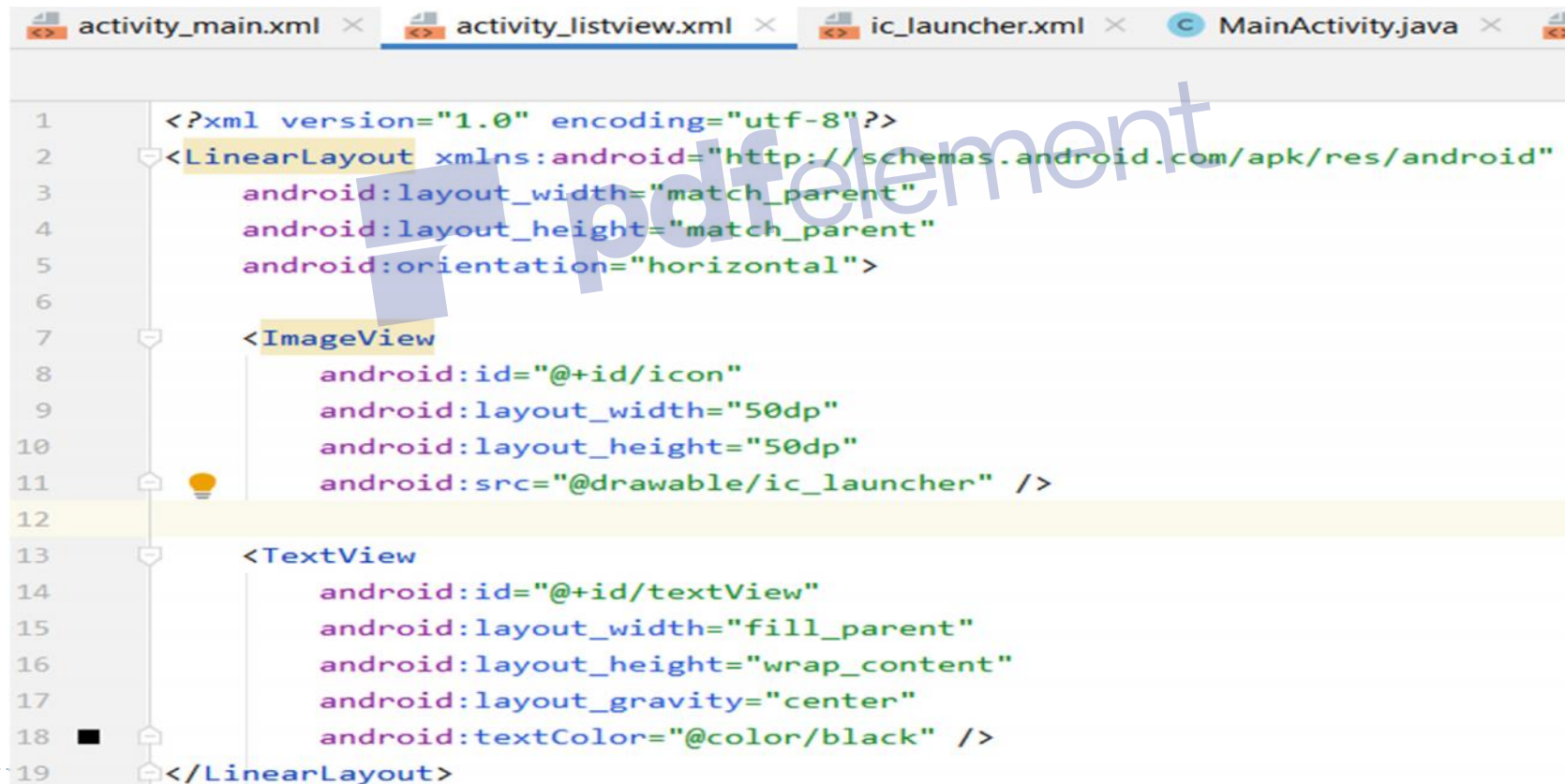
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ListView
        android:id="@+id/simpleListView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:divider="@color/material_blue_grey_800"
        android:dividerHeight="1dp" />

</LinearLayout>
```

# LISTVIEWS

- Configuration de l'affichage (exemple 2)
  - **Step 2:** Create a new layout, name Listview and below is the code of activity\_listview.xml



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="horizontal">
6
7      <ImageView
8          android:id="@+id/icon"
9          android:layout_width="50dp"
10         android:layout_height="50dp"
11         android:src="@drawable/ic_launcher" />
12
13     <TextView
14         android:id="@+id/textView"
15         android:layout_width="fill_parent"
16         android:layout_height="wrap_content"
17         android:layout_gravity="center"
18         android:textColor="@color/black" />
19 </LinearLayout>
```

# LISTVIEWS

- Configuration de l'affichage (exemple 2)
  - **Step 3:** In third step we will use custom adapter to display the country names in UI by coding MainActivity.java. **Below is the code of MainActivity.java**
    - **Important Note:** Make sure flag images are stored in drawable folder present inside res folder with correct naming.

```
public class MainActivity extends AppCompatActivity {  
  
    ListView simpleList;  
    String countryList[] = {"Algeria", "Palestine", "Iraq", "Jpan", "Morroco", "Cameron"};  
    int flags[] = {R.drawable.algeria, R.drawable.palestine, R.drawable.iraq, R.drawable.jpan, R.drawable.morocco, R.drawable.cameron}  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        simpleList = (ListView) findViewById(R.id.simpleListView);  
        CustomAdapter customAdapter = new CustomAdapter(getApplicationContext(), countryList, flags);  
        simpleList.setAdapter(customAdapter);  
    }  
}
```

# LISTVIEWS

---

- Configuration de l'affichage (exemple 2)
  - **Step 4:** Now create another class Custom Adapter which will extend BaseAdapter. Below is the code of CustomAdapter.[java](#)

```
public class CustomAdapter extends BaseAdapter {
    Context context;
    String countryList[];
    int flags[];
    LayoutInflater inflater;

    public CustomAdapter(Context applicationContext, String[] countryList, int[] flags) {
        this.context = context;
        this.countryList = countryList;
        this.flags = flags;
        inflater = (LayoutInflater.from(applicationContext));
    }

    @Override
    public int getCount() {
        return countryList.length;
    }

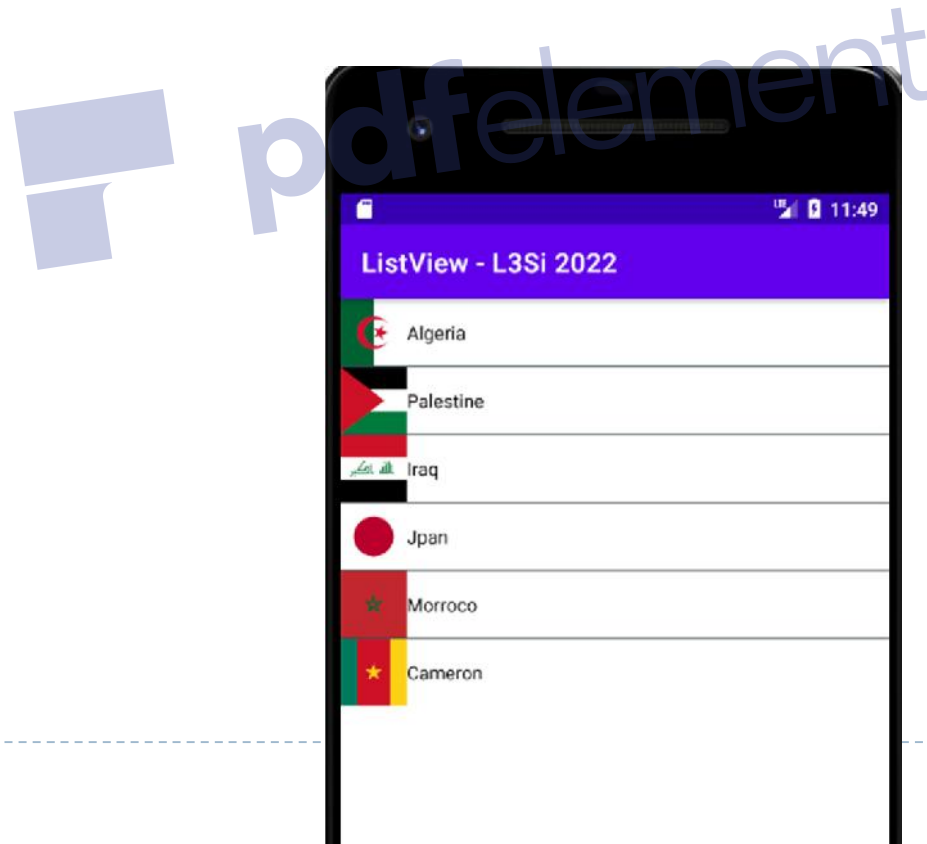
    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }
}
```



# LISTVIEWS

- ▶ Configuration de l'affichage (exemple 2)
  - ▶ **Output:**
    - ▶ Now run the App in Emulator and it will show you name of countries along with flags. Below is the output screen:





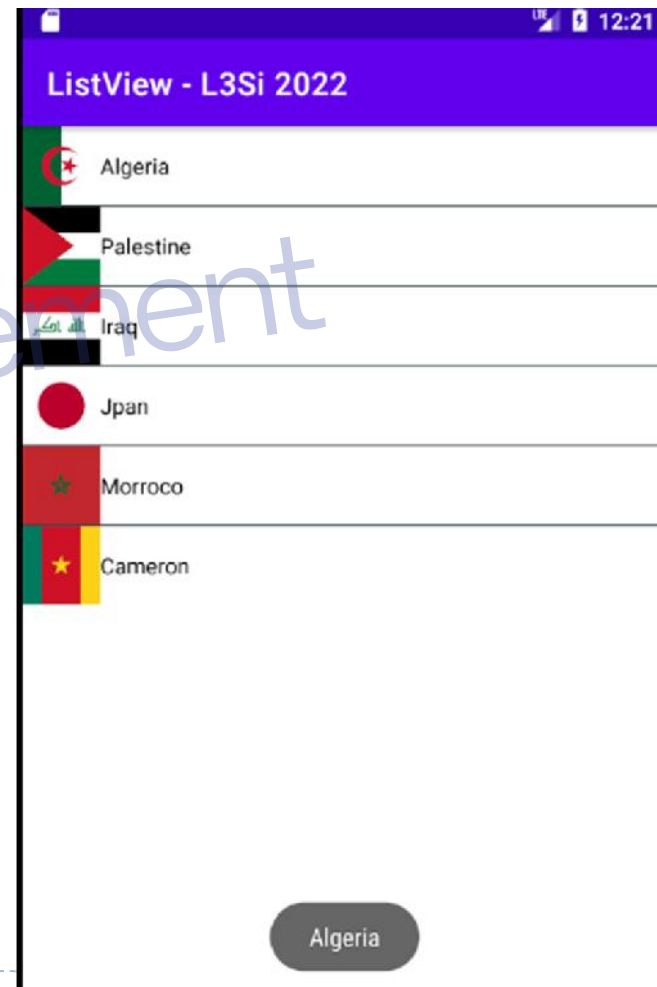
# LISTVIEWS

- Configuration de l'affichage (exemple 2)
  - Code for the OnItemClickListener

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    simpleList = (ListView) findViewById(R.id.simpleListView);  
    CustomAdapter customAdapter = new CustomAdapter(getApplicationContext(), countryList, flags);  
    simpleList.setAdapter(customAdapter);  
  
    simpleList.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
        @Override  
        public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {  
            Toast.makeText(getApplicationContext(), countryList[i], Toast.LENGTH_LONG).show(); // show  
        }  
    });  
}
```

# LISTVIEWS

- Configuration de l'affichage (exemple 2)
  - Code for the OnItemClickListener



# Semaine 9



# pdfelement

# Base de données Android SQLite