Programmation et interfaces Android

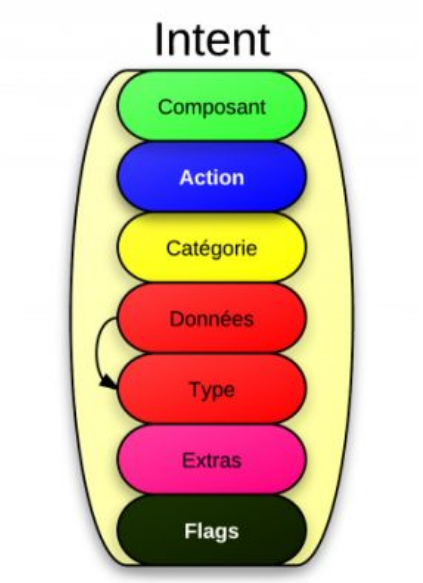
Manipulation de données et Organisation

Sommaire

1. Les Intents (Aspect technique)
2. Les intents (Explicites)
3. Les intents (Implicites)
4. Les intents (Résolution)
5. Les intents (Diffusion)
6. Sauvegarde de donnée in-app
7. Le MVC comme base

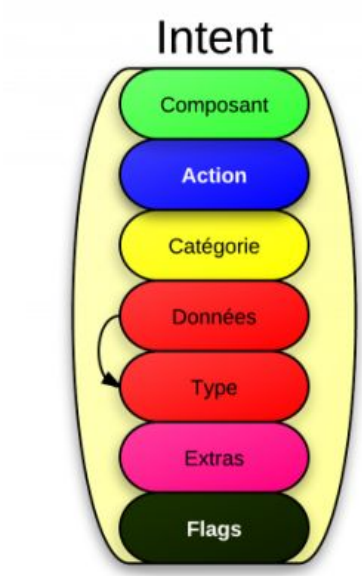
Les Intents (Aspect technique)

- Un intent est en fait un objet qui contient plusieurs champs



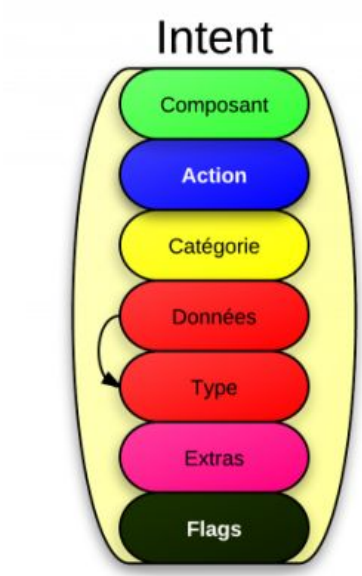
Les Intents (Aspect technique)

- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent



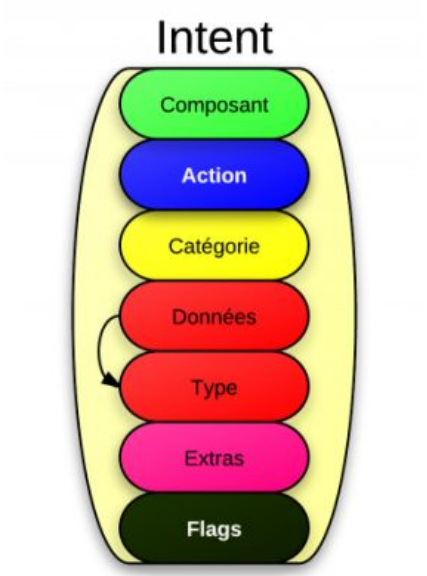
Les Intents (Aspect technique)

- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent
- Pour qu'un intent soit dit "**explicite**" il suffit que son champ **composant** soit renseigné



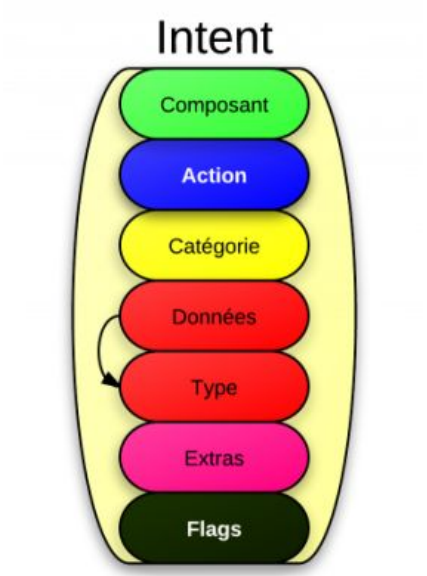
Les Intents (Aspect technique)

- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent
- Pour qu'un intent soit dit "**explicite**" il suffit que son champ **composant** soit renseigné
- Ce champ est constitué de deux informations : le package ou ce situe le composant de destination, ainsi que le nom du composant de destination (pour le retrouver de manière précise lors de l'exécution de l'intent)



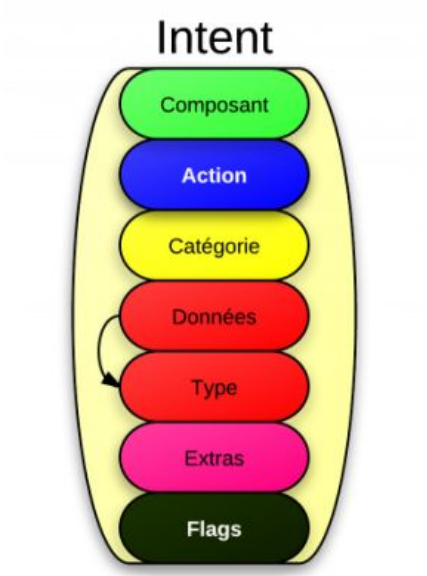
Les Intents (Aspect technique)

- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent
- à l'inverse un intent est dit "**implicites**" si on ne connais pas de manière précise le destinataire de l'intent



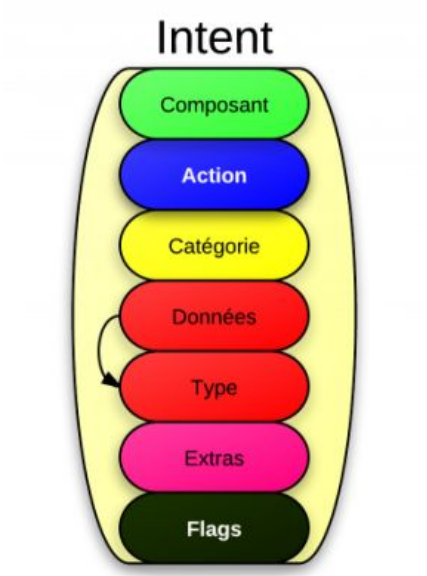
Les Intents (Aspect technique)

- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent
- à l'inverse un intent est dit "**implicites**" si on ne connais pas de manière précise le destinataire de l'intent
- c'est pourquoi on doit renseigner d'autres champs pour aider Android à déterminer quel composant est capable de réceptionner cet intent



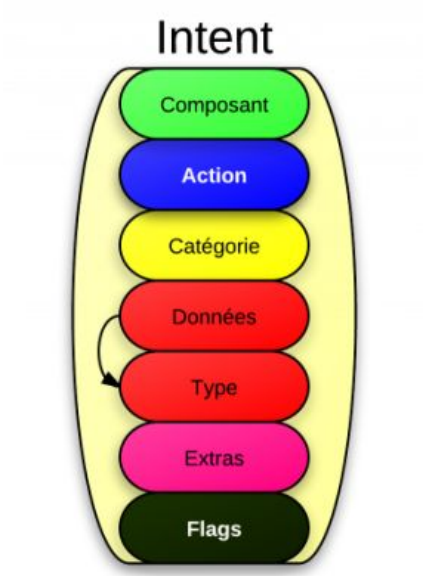
Les Intents (Aspect technique)

- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent
- à l'inverse un intent est dit "**implicites**" si on ne connais pas de manière précise le destinataire de l'intent
- c'est pourquoi on doit renseigner d'autres champs pour aider Android à déterminer quel composant est capable de réceptionner cet intent
- il faut donc au minimum 2 informations :
  - **Une action :** ce qu'on désire que le destinataire fasse.
  - **Un ensemble de données :** sur quelles données le destinataire doit effectuer son action.



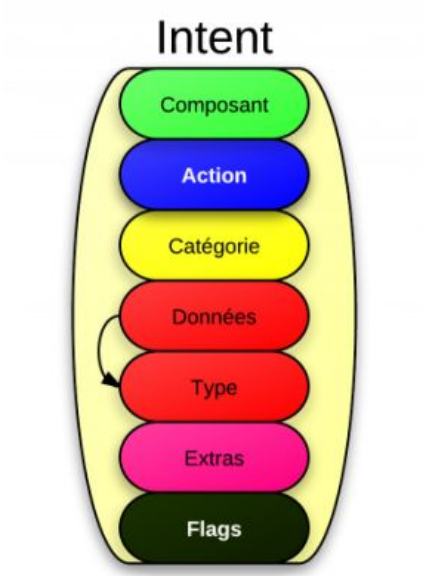
Les Intents (Aspect technique)

- Il existe aussi d'autres informations, pas forcément obligatoires, mais qui ont aussi leur utilité propre le moment venu :
  - **La catégorie :** permet d'apporter des informations supplémentaires sur l'action à exécuter et le type de composant qui devra gérer l'intent.
  - **Le type :** pour indiquer quel type de données incluses. Normalement ce type est contenu dans les données
  - **Les extras :** pour ajouter du contenu à vos intents afin de les faire circuler entre les composants
  - **Les flags :** permettent de modifier le comportement de l'intent



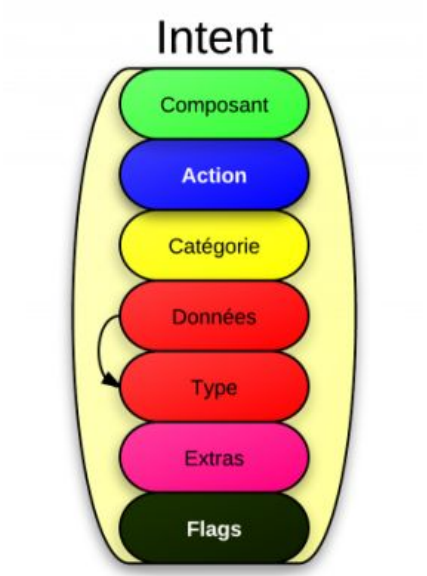
Les Intents (Aspect technique)

- Nous avons vu que les intents avaient un champ “extra” qui permet de contenir des données à véhiculer entre les composants. Pour cela il suffit d’utiliser la méthode : **putExtrat(String key, X value)** avec **key** la clé de l’extra et **value** la valeur associée



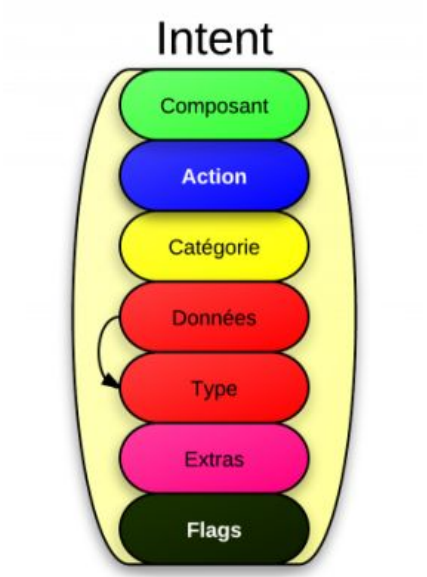
Les Intents (Aspect technique)

- Nous avons vu que les intents avaient un champ “extra” qui permet de contenir des données à véhiculer entre les composants. Pour cela il suffit d’utiliser la méthode : **putExtras(String key, X value)** avec **key** la clé de l’extra et **value** la valeur associée
- Puis on récupère l’extra d’un intent à l’aide la méthode : **getExtras()**



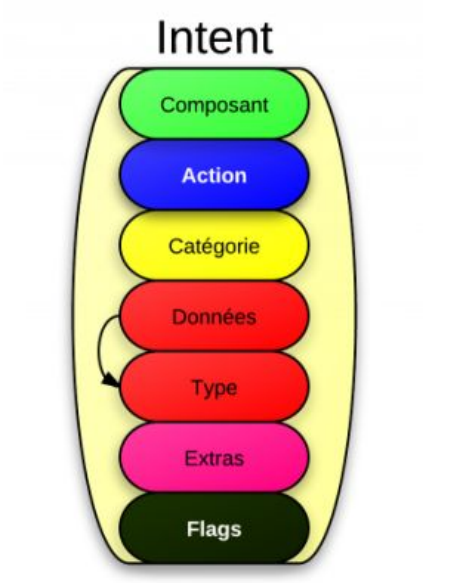
Les Intents (Aspect technique)

- Cependant, la méthode précédente ne peut pas prendre tous les objets, pour les objets autres que les primitive (Int, Float, String, etc) il faut qu'ils soient sérialisables. Sauf qu'en Android pour qu'un élément soit sérialisable il faut qu'il implémente correctement l'interface **Parcelable**



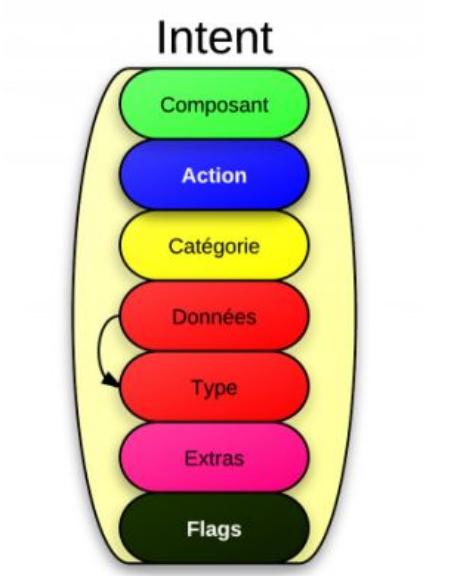
Les Intents (Aspect technique)

- Cependant, la méthode précédente ne peut pas prendre tous les objets, pour les objets autres que les primitive (Int, Float, String, etc) il faut qu'ils soient sérialisables. Sauf qu'en Android pour qu'un élément soit sérialisable il faut qu'il implémente correctement l'interface **Parcelable**
- Pour cela il faut que les objets qui implémente **Parcelable** définissent les deux méthodes suivantes :



Les Intents (Aspect technique)

- Cependant, la méthode précédente ne peut pas prendre tous les objets, pour les objets autres que les primitive (Int, Float, String, etc) il faut qu'ils soient sérialisables. Sauf qu'en Android pour qu'un élément soit sérialisable il faut qu'il implémente correctement l'interface **Parcelable**
- Pour cela il faut que les objets qui implémente **Parcelable** définissent les deux méthodes suivantes :
  - **describeContents() :** permet de définir si vous avez des paramètres spéciaux dans votre Parcelable
  - **writeToParcel(Parcel dest, int flags) : dest** le **Parcel** dans lequel nous allons insérer les attributs de notre objet et **flags** un entier qui vaut la plupart du temps 0, c'est ici que nous allons écrire dans le **Parcel** de destination
    Les attributs sont à inserer dans le Parcel dans l'ordre dans lequel ils sont déclarés dans la classe !



Les Intents (Aspect technique)

- Voici un exemple d'objet parcelable :

```
public class Contact implements Parcelable{
  private String mNom;
  private String mPrenom;
  private int mNumero;

  public Contact(String pNom, String pPrenom, int pNumero) {
    mNom = pNom;
    mPrenom = pPrenom;
    mNumero = pNumero;
  }

  @Override
  public int describeContents() {
    //On renvoie 0, car notre classe ne contient pas de FileDescriptor
    return 0;
  }

  @Override
  public void writeToParcel(Parcel dest, int flags) {
    // On ajoute les objets dans l'ordre dans lequel on les a déclarés
    dest.writeString(mNom);
    dest.writeString(mPrenom);
    dest.writeInt(mNumero);
  }
}
```

Les Intents (Aspect technique)

- Il faut ensuite ajouter un champ statique de type : Parcelable.Creator et qui s'appellera impérativement "CREATOR" :

```
public static final Parcelable.Creator<Contact> CREATOR = new Parcelable.Creator<Contact>() {
  @Override
  public Contact createFromParcel(Parcel source) {
    return new Contact(source);
  }

  @Override
  public Contact[] newArray(int size) {
    return new Contact[size];
  }
};

public Contact(Parcel in) {
  mNom = in.readString();
  mPrenom = in.readString();
  mNumero = in.readInt();
}
```

Les Intents (Aspect technique)

- Il n’y a plus qu'à envoyer notre classe dans l’intent :)

```
Intent i = new Intent();
Contact c = new Contact("Dupont", "Dupond", 06);
i.putExtra("sdz.chapitreTrois.intent.examples.CONTACT", c);
```

Les Intents (Explicites)

- Créer un intent explicite est très simple, il suffit de donner un Context qui appartient au package de destination :

```
Intent intent = new Intent(Activite_de_depart.this, Activite_de_destination.class);
```

Les Intents (Explicites)

- Il est possible d'envoyer un Intent sans retour (c'est à dire sans réponse de retour), pour cela il suffit simplement de démarrer la nouvelle activity par exemple avec la méthode : **startActivity(Intent intent)** et de lui passer votre nouvel intent fraichement crée

Les Intents (Explicites)

- Il est aussi possible d'envoyer un intent avec un feedback, pour cela on utilisera la méthode : **startActivityForResult(Intent intent, int requestCode)**, avec **requestCode** qui est une valeur qui permet d'identifier l'intent de manière unique

Les Intents (Explicites)

- Il est aussi possible d'envoyer un intent avec un feedback, pour cela on utilisera la méthode : **startActivityForResult(Intent intent, int requestCode)**, avec **requestCode** qui est une valeur qui permet d'identifier l'intent de manière unique
- Ce code doit être supérieur ou égal à 0

Les Intents (Explicites)

- Il est aussi possible d'envoyer un intent avec un feedback, pour cela on utilisera la méthode : **startActivityForResult(Intent intent, int requestCode)**, avec **requestCode** qui est une valeur qui permet d'identifier l'intent de manière unique
- Ce code doit être supérieur ou égal à 0
- Lorsque l'activité appelé ce terminera l'activité précédente entrera dans la méthode du cycle de vie suivante : **onActivityResult(int requestCode, int resultCode, Intent data)** le **requestCode** sera le même code que celui passé dans le startActivity et permet de repérer quel composant a provoqué l'appel de l'activité, **resultCode** est quant à lui un code renvoyé qui indique comment c'est terminé celle ci (Activity.RESULT_OK, Activity.RESULT_CANCELED, ...) et enfin un intent qui contient éventuellement des données de retour

Les Intents (Explicites)

- Voila un exemple :

  Activity appelante

```
startActivityForResult(secondeActivite, CHOOSE_BUTTON_REQUEST);
```

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  // On vérifie tout d'abord à quel intent on fait référence ici à l'aide de notre
dentifiant
  if (requestCode == CHOOSE_BUTTON_REQUEST) {
    // On vérifie aussi que l'opération s'est bien déroulée
    if (resultCode == RESULT_OK) {
      // On affiche le bouton qui a été choisi
      Toast.makeText(this, "Vous avez choisi le bouton " + data.getStringExtra(BUTTONS),
oast.LENGTH_SHORT).show();
    }
  }
}
```

Les Intents (Explicites)

- Voila un exemple :

  Activity appelé

```
Intent result = new Intent();
result.putExtra(MainActivity.BUTTONS, "2");
setResult(RESULT_OK, result);
finish();
```

Les Intents (Implicites)

- Ici on fera en sorte d'envoyer une requête à un destinataire sans savoir qui il est, et d'ailleurs on s'en fiche. Du coup les destinataires peuvent être des applications fournis par Android ou téléchargées sur le Play Store

Les Intents (Implicites)

- Ici on fera en sorte d'envoyer une requête à un destinataire sans savoir qui il est, et d'ailleurs on s'en fiche. Du coup les destinataires peuvent être des applications fournis par Android ou téléchargées sur le Play Store
- Il y'a différents type de donnée :
  - L'URI : ce présente comme ceci : **<schéma> : <information> { ? < requête } { # <fragment> },**
    - le schéma décrit quelle est la nature de l'information : tel, http, etc
    - l'information est la donnée en tant que telle : 0663583849, 123.45,-12.34
    - la requête permet de fournir une précision par rapport à l'information
    - le fragment permet d'accéder à une sous partie de l'inforamtion

  L'uri s'appelle grâce à la méthode : Uri.parse(String uri) , exemple pour envoyer un sms à une personne :

```
Uri sms = Uri.parse("sms:0606060606");
```

Les Intents (Implicites)

- Ici on fera en sorte d'envoyer une requête à un destinataire sans savoir qui il est, et d'ailleurs on s'en fiche. Du coup les destinataires peuvent être des applications fournis par Android ou téléchargées sur le Play Store
- Il y'a différents type de donnée :
  - Type MIME (non pas la formation :p) est un identifiant pour les formats de fichier (text, audio, video, ...) et aussi des sous types (audio/mp3, audio/wav, ...)
    - Vous pouvez crée votre propre type MIME comme ceci : vnd.votre_package.le_type
      Exemple : vnd.tuto.monPackage.contact_telephonique
    - Voici un lien de tout les types MIME standard : https://fr.wikipedia.org/wiki/Type_de_m%C3%A9dias

Les Intents (Implicites)

- Ici on fera en sorte d'envoyer une requête à un destinataire sans savoir qui il est, et d'ailleurs on s'en fiche. Du coup les destinataires peuvent être des applications fournis par Android ou téléchargées sur le Play Store
- Il y'a différents type de donnée :
    - L'Action est une constante qui se trouve dans la classe Intent et qui commence toujours par ACTION, si vous utilisez ACTION_VIEW sur un numéro de téléphone alors le numéro de téléphone s'affichera dans le composeur de numéro de tel
        - Vous pouvez aussi crée vos propres actions : package.intent.action.NOM_DE_L_ACTION

Les Intents (Implicites)

- Pour l'Action voici les différentes Action possible :

`ACTION_SENDTO`	Envoyer un message à quelqu'un	La personne à qui envoyer le message	/
`ACTION_VIEW`	Permet de visionner une donnée	Un peu tout. Une adresse e-mail sera visionnée dans l'application pour les e-mails, un numéro de téléphone dans le composeur, une adresse internet dans le navigateur, etc.	/
ACTION_WEB_SEARCH	Effectuer une recherche sur internet	S'il s'agit d'un texte qui commence par « http », le site s'affichera directement, sinon c'est une recherche dans Google qui se fera	/

Intitulé	Action	Entrée attendue	Sortie attendue
`ACTION_MAIN`	Pour indiquer qu'il s'agit du point d'entrée dans l'application	/	/
`ACTION_DIAL`	Pour ouvrir le composeur de numéros téléphoniques	Un numéro de téléphone semble une bonne idée :-p	/
`ACTION_DELETE*`	Supprimer des données	Un URI vers les données à supprimer	/
`ACTION_EDIT*`	Ouvrir un éditeur adapté pour modifier les données fournies	Un URI vers les données à éditer	/
`ACTION_INSERT*`	Insérer des données	L'URI du répertoire où insérer les données	L'URI des nouvelles données créées
`ACTION_PICK*`	Sélectionner un élément dans un ensemble de données	L'URI qui contient un répertoire de données à partir duquel l'élément sera sélectionné	L'URI de l'élément qui a été sélectionné
`ACTION_SEARCH`	Effectuer une recherche	Le texte à rechercher	/

Les Intents (Implicites)

- Et par exemple pour créer un intent qui va ouvrir le composeur téléphonique avec le numéro de téléphone : 0606060606 j'adapte mon code comme ceci :

```
public void onClick(View v) {
  Uri telephone = Uri.parse("tel:0606060606");
  Intent secondeActivite = new Intent(Intent.ACTION_DIAL, telephone);
  startActivity(secondeActivite);
}
```

Les Intents (Résolution)

- Pour résumer rapidement : un Intent Explicite permet de fournir des données et d'appeler un élément spécifique, tandis qu'un Intent Implicite lance un appel sur tout le téléphone à une applications ou un morceau dans l'application qui peut traiter les données fournis

Les Intents (Résolution)

- Comment Android détermine qui doit répondre à un intent ?

Les Intents (Résolution)

- Comment Android détermine qui doit répondre à un intent ? C’est simple il va comparer l’intent à des filtres que nous allons déclarer dans le Manifest et qui signalent que les composants de nos applications peuvent gérer certains intents.

Les Intents (Résolution)

- Comment Android détermine qui doit répondre à un intent ? C'est simple il va comparer l'intent à des filtres que nous allons déclarer dans le Manifest et qui signalent que les composants de nos applications peuvent gérer certains intents.
- Ces filtres sont les noeuds **<intent-filter>**, Un composant d'application doit avoir autant de filtres que de capacités de traitement : si il peut gérer deux types d'intent il doit donc avoir deux filtres

Les Intents (Résolution)

- Pour qu'un intent match avec un filter il faut remplir 3 critères

Les Intents (Résolution)

- Pour qu'un intent match avec un filter il faut remplir 3 critères :
  - l'Action : Permet de filtrer en fonction du champ d'action (un ou plusieurs), si vous ne mettez pas de filtre tout les intents seront recalés à l'entrée, inversement si vous ne mettez pas d'action dans un intent il sera automatiquement accepté pour ce test, exemple ci dessous

```
<activity>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.SENDTO" />
  </intent-filter>
</activity>
```

Les Intents (Résolution)

- Pour qu'un intent match avec un filter il faut remplir 3 critères :
  - la Catégorie : il n'est pas indispensable d'avoir une catégorie pour un Intent, mais si il y'en a une ou plusieurs alors pour passer le filtre il faut que les catégories correspondent

```
<activity>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.SEARCH" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="com.sdz.intent.category.DESEMBROUILLEUR" />
  </intent-filter>
</activity>
```

Les Intents (Résolution)

- Pour qu'un intent match avec un filter il faut remplir 3 critères :
  - la Catégorie : il n'est pas indispensable d'avoir une catégorie pour un Intent, mais si il y'en a une ou plusieurs alors pour passer le filtre il faut que les catégories correspondent

`CATEGORY_DEFAULT`	Indique qu'il faut effectuer le traitement par défaut sur les données correspondantes. Concrètement, on l'utilise pour déclarer qu'on accepte que ce composant soit utilisé par des intents implicites.
`CATEGORY_BROWSABLE`	Utilisé pour indiquer qu'une activité peut être appelée sans risque depuis un navigateur web. Ainsi, si un utilisateur clique sur un lien dans votre application, vous promettez que rien de dangereux ne se passera à la suite de l'activation de cet intent.
`CATEGORY_TAB`	Utilisé pour les activités qu'on retrouve dans des onglets.
`CATEGORY_ALTERNATIVE`	Permet de définir une activité comme un traitement alternatif dans le visionnage d'éléments. C'est par exemple intéressant dans les menus, si vous souhaitez proposer à votre utilisateur de regarder telles données de la manière proposée par votre application ou d'une manière que propose une autre application.
`CATEGORY_SELECTED_ALTERNATIVE`	Comme ci-dessus, mais pour des éléments qui ont été sélectionnés, pas seulement pour les voir.
`CATEGORY_LAUNCHER`	Indique que c'est ce composant qui doit s'afficher dans le lanceur d'applications.
`CATEGORY_HOME`	Permet d'indiquer que c'est cette activité qui doit se trouver sur l'écran d'accueil d'Android.
`CATEGORY_PREFERENCE`	Utilisé pour identifier les `PreferenceActivity` (dont nous parlerons au chapitre suivant).

Les Intents (Résolution)

- Pour qu'un intent match avec un filter il faut remplir 3 critères :
  - les Données : il est possible de préciser plusieurs informations sur les données qu'une activité peut traiter. Principalement on peut préciser le schéma qu'on veut avec android:scheme on peut aussi préciser le type MIME avec android:mimeType. Exemple si on traite des fichiers textes qui proviennent d'internet on aura besoin du type "texte" et du schéma "internet"

```
<data android:mimeType="text/plain" android:scheme="http" />
<data android:mimeType="text/plain" android:scheme="https" />
```

Les Intents (Résolution)

- Enfin il est possible de vérifier si un intent à bien porter ses fruits grâce au **PackageManager**

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("tel:0606060606"));

PackageManager manager = getPackageManager();

ComponentName component = intent.resolveActivity(manager);
// On vérifie que component n'est pas null
if(component != null)
  //Alors c'est qu'il y a une activité qui va gérer l'intent
```

Les Intents (Diffusion)

- Jusque la on à vue comment dire “Je veux que vous traitiez cela, alors que quelqu’un le fasse pour moi”. Ici on va voir comment dire “Cet évènement vient de se dérouler, je préviens juste”. En gros au lieu d’attendre un traitement on va notifier d’un évènement sur lequel il y’a un potentiel traitement à faire

Les Intents (Diffusion)

- Jusque la on à vue comment dire “Je veux que vous traitiez cela, alors que quelqu’un le fasse pour moi”. Ici on va voir comment dire “Cet évènement vient de se dérouler, je préviens juste”. En gros au lieu d’attendre un traitement on va notifier d’un évènement sur lequel il y’a un potentiel traitement à faire
- C’est le principe d’intent anonyme et broadcasté à tout le système. (Broadcast intents), on va encore une fois utilisé un filtrage pour savoir qui peut traiter cet intent mais c’est la façon de le réceptionner et de l’envoyer qui change un peu

Les Intents (Diffusion)

- La création de broadcast intents est similaire à celle des intents classiques, sauf que la vous les envoyer avec la méthode **sendBroadcast(Intent intent).** Ainsi l'intent ne sera reçu que par les classes qui dérivent de la classe **BroadcastReceiver** de plus dans le Manifest lors de la déclaration d'un composant il faudra annoncez qu'il s'agit d'un broadcast receiver

Les Intents (Diffusion)

- La création de broadcast intents est similaire à celle des intents classiques, sauf que la vous les envoyer avec la méthode **sendBroadcast(Intent intent).** Ainsi l'intent ne sera reçu que par les classes qui dérivent de la classe **BroadcastReceiver** de plus dans le Manifest lors de la déclaration d'un composant il faudra annoncez qu'il s'agit d'un broadcast receiver

```
<receiver android:name="CoucouReceiver">
  <intent-filter>
    <action android:name="sdz.chapitreTrois.intent.action.coucou" />
  </intent-filter>
</receiver>
```

Les Intents (Diffusion)

- La création de broadcast intents est similaire à celle des intents classiques, sauf que la vous les envoyer avec la méthode **sendBroadcast(Intent intent).** Ainsi l'intent ne sera reçu que par les classes qui dérivent de la classe **BroadcastReceiver** de plus dans le Manifest lors de la déclaration d'un composant il faudra annoncez qu'il s'agit d'un broadcast receiver

```
public class CoucouReceiver extends BroadcastReceiver {
  private static final String NOM_USER = "sdz.chapitreTrois.intent.extra.NOM";

  // Déclenché dès qu'on reçoit un broadcast intent qui réponde aux filtres déclarés dans le
Manifest
  @Override
  public void onReceive(Context context, Intent intent) {
    // On vérifie qu'il s'agit du bon intent
    if(intent.getAction().equals("ACTION_COUCOU")) {
      // On récupère le nom de l'utilisateur
      String nom = intent.getExtra(NOM_USER);
      Toast.makeText(context, "Coucou " + nom + " !", Toast.LENGTH_LONG).show();
    }
  }
}
```

Les Intents (Diffusion)

- Un broadcast receiver déclaré de cette manière sera disponible tout le temps, même quand l'application n'est pas lancée ! mais ne sera viable que pendant la durée d'exécution de la méthode onReceive. Ne vous attendez donc pas à ce que cela fonctionne si vous lancez un thread, une dialog ou un autre composant d'application à partir de lui
- Mais il est possible de déclarer un broadcast receiver de manière dynamique, directement dans le code, cette technique est utilisé pour gérer les évènement de l'interface graphique

Les Intents (Diffusion)

- Un broadcast receiver déclaré de cette manière sera disponible tout le temps, même quand l'application n'est pas lancée ! mais ne sera viable que pendant la durée d'exécution de la méthode onReceive. Ne vous attendez donc pas à ce que cela fonctionne si vous lancez un thread, une dialog ou un autre composant d'application à partir de lui
- Mais il est possible de déclarer un broadcast receiver de manière dynamique, directement dans le code, cette technique est utilisé pour gérer les évènement de l'interface graphique
- Pour cela il faut crée uen classe qui dérive de BroadcastReceiver (sans l'enregistrer dans le Manifest) puis lui rajouter de lois de filtrage avec la classe IntentFilter et l'enregistrer/désenregistré d'une activité avec les méthodes : registerReceiver(BroadcastReceiver receiver, IntentFilter filter) et unregisterReceiver(BroadcastReceiver receiver)

Les Intents (Diffusion)

- Ensuite dans notre activity on fait :

```
import android.app.Activity;
import android.content.IntentFilter;
import android.os.Bundle;

public class CoucouActivity extends Activity {
  private static final String COUCOU = "sdz.chapitreTrois.intent.action.coucou";
  private IntentFilter filtre = null;
  private CoucouReceiver receiver = null;

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

    filtre = new IntentFilter(COUCOU);
    receiver = new CoucouReceiver();
  }

  @Override
  public void onResume() {
    super.onResume();
    registerReceiver(receiver, filtre);
  }

  /** Si vous déclarez votre receiver dans le onResume, n'oubliez pas qu'il faut l'arrêter
dans le onPause **/
  @Override
  public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
  }
}
```

Les Intents (Diffusion)

- Maintenant qu’on à vue ça on va parler un peu sécurité !

Les Intents (Diffusion)

- Et oui car n’importe quelle application peut envoyer des broadcast intents à votre receiver, ce qui est une faiblesse de sécurité ! Voila comment la corriger

Les Intents (Diffusion)

- Et oui car n'importe quelle application peut envoyer des broadcast intents à votre receiver, ce qui est une faiblesse de sécurité ! Voila comment la corriger
- On ajoute android:exported="false" qui permet de faire en sorte que le receiver ne soit accessible que dans votre app

```
<receiver android:name="CoucouReceiver"
  android:exported="false">
  <intent-filter>
    <action android:name="sdz.chapitreTrois.intent.action.coucou" />
  </intent-filter>
</receiver>
```

Les Intents (Diffusion)

- Et aussi lors de l'envoie toutes les applications peuvent recevoir le broadcast intent ! Afin de déterminer qui peut recevoir un broadcast intent il suffit de lui ajouter une permission à l'aide la méthode : **sendBroadcast(Intent intent, String receiverPermission)** avec **receiverPermission** une permission que vous aurez déterminée.

Les Intents (Diffusion)

- Et aussi lors de l'envoie toutes les applications peuvent recevoir le broadcast intent ! Afin de déterminer qui peut recevoir un broadcast intent il suffit de lui ajouter une permission à l'aide la méthode : **sendBroadcast(Intent intent, String receiverPermission)** avec **receiverPermission** une permission que vous aurez déterminée.

```
private String COUCOU_BROADCAST = "sdz.chapitreTrois.permission.COUCOU_BROADCAST";

…

sendBroadcast(i, COUCOU_BROADCAST);
```

```
<uses-permission android:name="sdz.chapitreTrois.permission.COUCOU_BROADCAST"/>
```

Sauvegarde de donnée (In-app)

- Dans de grand nombre d’applications il est utile voire indispensable de sauvegarder des données utilisateurs dans le téléphone afin de proposer des expérience personnalisé pour chacun.

Sauvegarde de donnée (In-app)

- Dans de grand nombre d’applications il est utile voire indispensable de sauvegarder des données utilisateurs dans le téléphone afin de proposer des expérience personnalisé pour chacun.
- Le point de départ de ce genre de sauvegarde est la classe **SharedPreferences.** Elle possède des méthodes qui permettent d’enregistrer et de récupérer des donnée via des identifiant. Ce qui permet bien évidemment des conserver des données même lors de l’arrêt de l’application

Sauvegarde de donnée (In-app)

- Dans de grand nombre d’applications il est utile voire indispensable de sauvegarder des données utilisateurs dans le téléphone afin de proposer des expérience personnalisé pour chacun.
- Le point de départ de ce genre de sauvegarde est la classe **SharedPreferences.** Elle possède des méthodes qui permettent d’enregistrer et de récupérer des donnée via des identifiant. Ce qui permet bien évidemment des conserver des données même lors de l’arrêt de l’application

Sauvegarde de donnée (In-app)

- Pour faire cela c'est très simple il suffit d'enregistrer une donnée comme ceci : (par exemple une couleur favorite)

```
public final static String FAVORITE_COLOR = "fav color";

…

SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);
SharedPreferences.Editor editor = preferences.edit();
editor.putString(FAVORITE_COLOR, "FFABB4");
editor.commit();
```

Sauvegarde de donnée (In-app)

- Et de la récupérer ailleurs comme ceci

```
// On veut la chaîne de caractères d'identifiant FAVORITE_COLOR
// Si on ne trouve pas cette valeur, on veut rendre "FFFFFF"
String couleur = preferences.getString(FAVORITE_COLOR, "FFFFFF");
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
  - Tout d'abord il faut ajouter les dépendance suivantes :

```
// Kotlin
compile "android.arch.persistence.room:runtime:1.0.0"
annotationProcessor "android.arch.persistence.room:compiler:1.0.0"

// Gson
compile "com.google.code.gson:gson:2.8.0"
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
    - Ensuite on va crée un modèle de donnée :

```
abstract class User(var name: String)

class Player (name: String, var position: String) : User(name) {
    lateinit var avatar: String
}

class Coach(name: String, var experience: Int) : User(name)
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
    - Il est possible de stipuler des annotation SQL qui vont permettre la traduction de la classe en objet SQL

```
@Entity(tableName="players")
class Player (name: String, var position: String) : User(name) {
    @Ignore
    lateinit var avatar: String
}


@Entity
class Coach(name: String,
            @ColumnInfo(name = "xp")
            var experience: Int
           ) : User(name)
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
    - Il est possible de stipuler des annotation SQL qui vont permettre la traduction de la classe en objet SQL

```
abstract class User(var name: String) {
    @PrimaryKey(autoGenerate = true)
    var id: Long = 0
}
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
    - Il est possible aussi de stipuler une relation d'objet à objet : One-to-many

```
class TeamAllPlayers {

    @Embedded
    var team: Team? = null

    @Relation(parentColumn = "id", entityColumn = "teamId")
    var players: List<Player>? = null
}
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
  - Il est possible aussi de stipuler une relation d'objet à objet : Many-to-many

```
@Entity(tableName = "team_player_join",
        primaryKeys = { "teamId", "playerId" },
        foreignKeys = {
                ForeignKey(entity = Team.class,
                           parentColumns = "id",
                           childColumns = "teamId"),
                ForeignKey(entity = Player.class,
                           parentColumns = "id",
                           childColumns = "playerId")
                })
class TeamPlayerJoin(val teamId: Long, val playerId: Long)
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
    - Il faut ensuite crée un converter qui va s'occuper de convertir un json en objet

```
class PlayersConverter {

    @TypeConverter
    fun stringToPlayers(value: String): List<Long> {
        val listPlayers = object : TypeToken<Long>() {}.type
        return Gson().fromJson(value, listPlayers)
    }

    @TypeConverter
    fun playersToString(list: List<Long>): String {
        val gson = Gson()
        return gson.toJson(list)
    }
}
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
  - Puis il faut crée un DAO qui servira de listes d'actions possible sur notre objet

```
@Dao
interface UserDao {

    /* PLAYER */

    @Insert
    fun insertPlayer(player: Player) : Long

    @Insert
    fun insertPlayers(players: List<Player>) : List<Long>

    @Insert
    fun insertPlayers(vararg players: Player)
    @Update
    fun updatePlayer(player: Player)
    ...

    @Delete
    fun deletePlayer(player: Player)
    ...

    @Query("SELECT * FROM Player")
    fun getAllPlayer(): List<Player>

    @Query("SELECT * FROM Player WHERE id=:playerId")
    fun getPlayer(playerId: Long) : Player

    /* Coach */
    ...
}
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
    - Ensuite il faut crée l'objet qui représentera la base de donnée

```
@Database(entities = arrayOf(
            Player::class, Coach::class, Team::class, Match::class, Score::c
            version = 1)
@TypeConverters(Converter::class)
abstract class MyDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
```

Sauvegarde de donnée (In-app)

- Il est aussi possible grâce au nouveauté d'android d'utiliser les room, elles permettent de stocket des données sous forme SQL dans notre applications (SQLite), voici comment procéder :
  - Pour finir voila comment crée notre BDD in app :

    `MyApp.database = Room.databaseBuilder(this, MyDatabase::class.java, "championship-db").build()`
  - Et un exemple d'utilisation (création d'un player)

```
fun createPlayer(name: String, position: String, avatar: String) : Player {
    var player = Player(name, position)
    player.avatar = avatar
    player.id = MyApp.database?.userDao()?.insertPlayer(player)!!
    return player
}
```

Le MVC comme base

- Au cours de notre avancé en Android on à crée plein de classes, d’objets en tout genre mais comment pouvons nous organiser nos fichier afin de rendre ça plus “lisible” et plus “logique” ?

Le MVC comme base

- Au cours de notre avancé en Android on à crée plein de classes, d'objets en tout genre mais comment pouvons nous organiser nos fichier afin de rendre ça plus "lisible" et plus "logique" ?
- Il existe aujourd'hui 3 architectures utilisé en Android :
    - Le MVC : Modèle Vue Controlleur
    - Le MVP : Modèle Vue Presenter
    - Le MVVM : Modèle Vue Vue-Modèle

Le MVC comme base

- Au cours de notre avancé en Android on à crée plein de classes, d'objets en tout genre mais comment pouvons nous organiser nos fichier afin de rendre ça plus "lisible" et plus "logique" ?
- Il existe aujourd'hui 3 architectures utilisé en Android :
    - Le MVC : Modèle Vue Controlleur
    - Le MVP : Modèle Vue Presenter
    - Le MVVM : Modèle Vue Vue-Modèle
- Le MVVM est aujourd'hui le plus demandé sur le marché du travail mais aussi le plus complexe !

Le MVC comme base

- Au cours de notre avancé en Android on à crée plein de classes, d'objets en tout genre mais comment pouvons nous organiser nos fichier afin de rendre ça plus "lisible" et plus "logique" ?
- Il existe aujourd'hui 3 architectures utilisé en Android :
    - Le MVC : Modèle Vue Controlleur
    - Le MVP : Modèle Vue Presenter
    - Le MVVM : Modèle Vue Vue-Modèle
- Le MVVM est aujourd'hui le plus demandé sur le marché du travail mais aussi le plus complexe !
- Le MVP est encore un peu utilisé mais il à surtout servi de transition au MVVM

Le MVC comme base

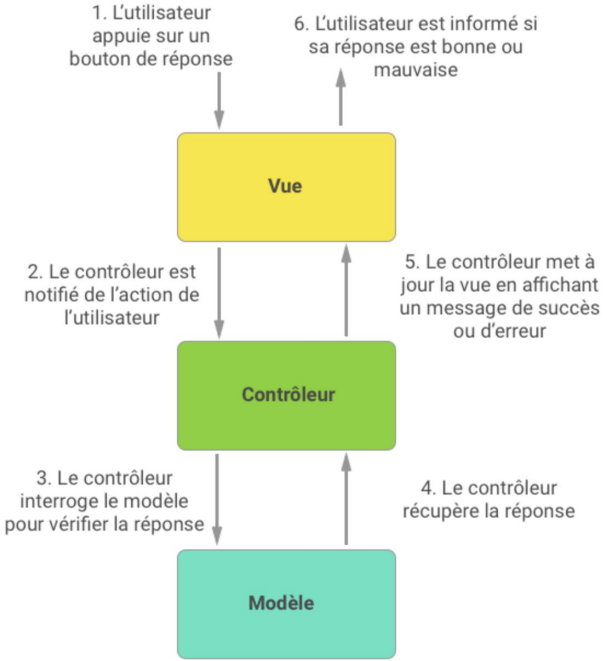
- Au cours de notre avancé en Android on à crée plein de classes, d'objets en tout genre mais comment pouvons nous organiser nos fichier afin de rendre ça plus "lisible" et plus "logique" ?
- Il existe aujourd'hui 3 architectures utilisé en Android :
    - Le MVC : Modèle Vue Controlleur
    - Le MVP : Modèle Vue Presenter
    - Le MVVM : Modèle Vue Vue-Modèle
- Le MVVM est aujourd'hui le plus demandé sur le marché du travail mais aussi le plus complexe !
- Le MVP est encore un peu utilisé mais il à surtout servi de transition au MVVM
- Et le MVC est l'architecture qui à été la plus utilisé pendant des dizaines d'année (toujours présente sur d'autres technologie), et c'est la plus simple à comprendre, Nous allons donc nous attaquez à celle la pour vous permettre d'avoir une première base en organisation de code

Le MVC comme base

- L’architecture MVC consiste à découper son code pour qu’il appartienne à l’une des trois composantes du MVC
  - Modèle : Contient les données de l’application et la logique métier. c’est lui va contenir les objet : User, Pictures, Etc... que vous pourrez crée et utilisez dans vos app
  - Vue : Contient tout ce qui est visible à l’écran et qui propose une interaction utilisateur (en gros vos fichiers layout.xml)
  - Contrôleur : c’est la colle entre la vue et le modèle qui gère aussi la logique de l’application c’est vos fichier : Activity, Fragment etc ..

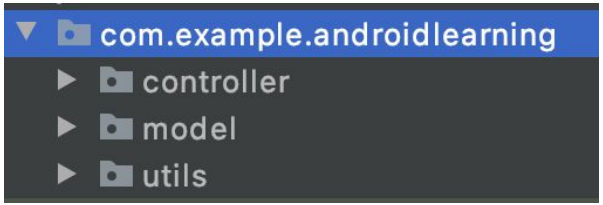
Le MVC comme base

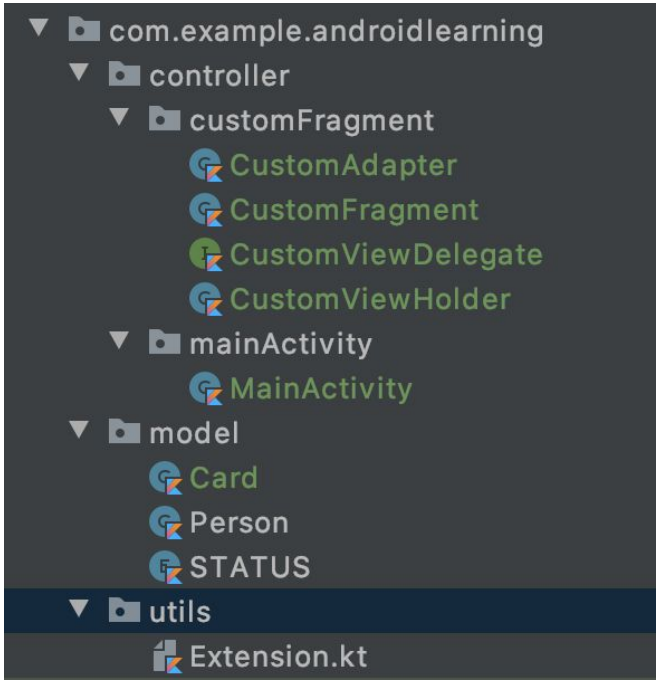
- Voila comment ce présente le MVC :



Le MVC comme base

- Voila dans le code :





Le MVC comme base

- Dans le prochain cours nous verrons comment implémenter le MVVM qui même si il est complexe est utile sur le marché du travail, nous verrons donc dans ce cours une façon basique et simple de l'implémenter