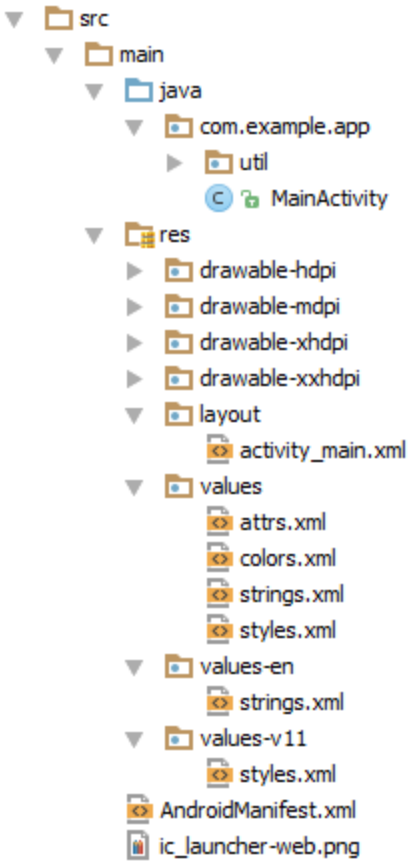
Android Cheat Sheet #1

Android Cheat Sheet #1

Organisation des répertoires d'un projet Android :



- **Java** : Répertoire des classes Java
- **Res** : Répertoire des ressources
- **drawable** : Répertoire des images (ldpi: Basse définition, mdpi: Moyenne, hdpi: haute, xhdpi: très haute, etc...)
- **layout** : Répertoire des structures visuelles de l'IHM (interface homme machine)
- **values** : Répertoire des ressources textuelles autres
- **attrs.xml** : Permet la définition d'attributs personnalisés pour les layouts
- **colors.xml** : Permet la définition de couleurs personnalisées (Exemple : couleur du menu)
- **strings.xml** : Permet la définition des différents textes utilisés dans l'application
- **styles.xml** : Permet la définition de styles personnalisés pour les views (éléments) des layouts
- **values-en (values-it, values-de, etc...)** : Permet la traduction de l'application dans les diverses langues
- **AndroidManifest.xml** : Paramètres de l'application
- **ic_launcher-web.png** : Icone pour publier dans le play store en taille 512x512

Resources :

Les ressources présentes dans le dossier res ne doivent pas contenir de majuscules, d'accents, de caractères spéciaux mis à part les « _ ».

Layouts :

Définit la structure visuelle d'une vue.
Peut contenir :

- EditText (équivalent des textbox/input type=text)
- ImageView
- LinearLayout (groupement de vues qui aligne tous ses enfants de manière ordrée horizontalement ou verticalement) Voir +...
- ListView (groupement de vues en liste spécifique qui demande d'avoir ses enfants dans un Adapter (page 2, Voir +...)
- Button
- TextView
- CheckBox
- RelativeLayout (groupement de vues qui permet de placer ses enfants à des positions définies et donc qui permet de les superposer) Voir +...
- GridView (groupement de vues similaire à la ListView avec une notion de colonnes, Voir +...)
- Etc...

Manière d'utiliser les Layouts explicitée dans la partie Activity et dans « Communiquer avec un layout »

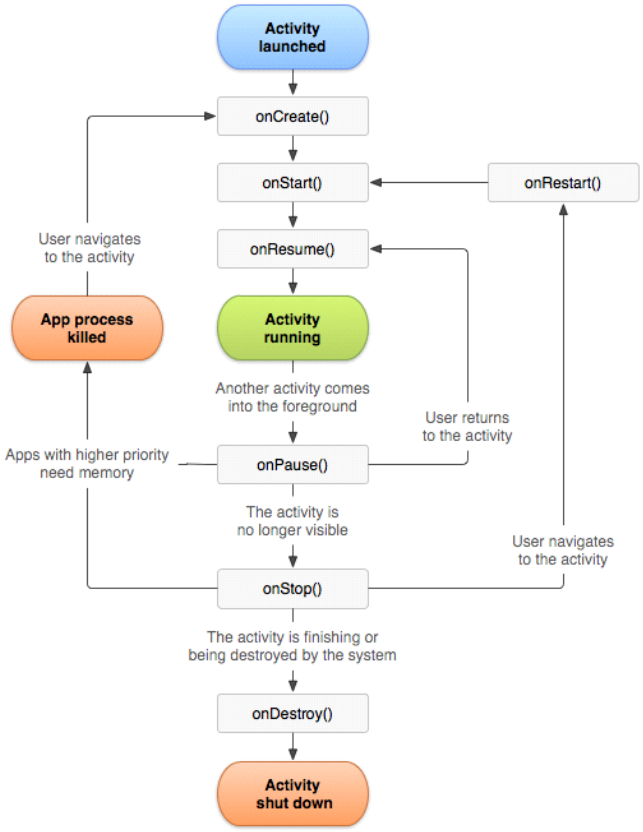
Android Cheat Sheet #1

Activity :

Composant qui permet d'afficher un écran de l'application

Cycle de vie d'une activité

- onCreate() : Se produit lors du lancement principal de l'activité ou lors d'une bascule entre applications
- onStart() : Se produit à chaque fois que l'activité est affichée
- onStop() : Se produit à chaque fois que l'activité est cachée
- Autres redéfinitions peu utilisées



Exemple d'activité avec redéfinition de la méthode onCreate :

```
public class MonActivite extends Activity {

        @Override
        public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //Permet la création d'une vue avec le contenu du layout
nommé main.xml dans les ressources (/res/layout/main.xml)
        }
}
```

Déclaration correspondante dans l'AndroidManifest.xml si l'activité n'est pas la première de l'application :

```
<activity
    android:name="nom.du.package.MonActivite"
    android:label="@string/app_name" >
</activity>
```

Destruction d'une activity : finish()

Déclaration correspondante dans l'AndroidManifest.xml si l'activité est la première de l'application :

```
<activity
    android:name="nom.du.package.MonActivite"
    android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

Sauvegarde de l'état de l'activité lors de la bascule sur une autre activité

Bundle : Conteneur de données basiques similaire à un dictionnaire mais pouvant stocker différents types (boolean, boolean[], byte, byte[],char,char[],String, double, int, long, etc…)
onSaveInstanceState s'exécute automatiquement lorsque les données de l'activité vont être remise à zéro
Exemple onSaveInstanceState :

```
protected void onSaveInstanceState(Bundle sauvegardeBundle) {
 super.onSaveInstanceState(sauvegardeBundle);
 sauvegardeBundle.putString("contenuDeMonEditTextNom", edNom.getText().toString());
}
```

Exemple réutilisation dans onCreate :

```
public void onCreate(Bundle sauvegardeBundle) {
 if (sauvegardeBundle!= null){
   edNom.setText(sauvegardeBundle.getString("contenuDeMonEditTextNom "));
 }
}
```

Voir +…

Les intents :

Utilisation nécessaire de la classe Intent, qui traduit littéralement voudrai dire « Intention de faire une action ».

Les intents peuvent servir à :

- Démarrer une activité depuis une autre activité
- Utiliser une fonctionnalité d'une application tierce

Démarrage d'une activité avec un intent depuis une autre activité:

```
Intent myIntent = new Intent(getApplicationContext(),MonActivite.class);
startActivity(myIntent);
```

Dans cet exemple, le code démarre une activité MonActivite avec le context ApplicationContext. getApplicationContext() est une méthode provenant des activités.

Démarrage d'une activité avec un intent avec récupération du résultat de la fermeture de l'activité :

Dans l'activité qui va démarrer MonActivite :

```
private static final int MON_ACTIVITE_REQUEST_CODE = 1;
private void startMonActivite(){
   Intent myIntent = new Intent(getApplicationContext(),MonActivite.class);
   startActivityForResult(myIntent,MON_ACTIVITE_REQUEST_CODE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   if (requestCode == MON_ACTIVITE_REQUEST_CODE)
   {
      if (resultCode==RESULT_OK)
      {
         String s = data.getStringExtra("etat");
      }
   }
}
```

Dans l'activité MonActivite :

```
private void activiteTerminee(boolean resultat, boolean etatHyperactif){
   if (resultat){
      Intent fermetureMonActivite = new Intent();
      if (etatHyperactif){
         fermetureMonActivite.putExtra("etat","hyperactif");
      }else{
         fermetureMonActivite.putExtra("etat","calme");
      }
      setResult(RESULT_OK,fermetureMonActivite);
   }else{
      setResult(RESULT_CANCELED);
   }
   finish();
}
```

L'activité de base démarre MonActivite en attendant un résultat pour le code de requete MON_ACTIVITE_REQUEST_CODE. L'activité MonActivite lorsque l'on exécute la méthode activiteTerminee sauvegarde le résultat via setResult. Ce résultat est alors retransmis lors du finish() à l'activité de base via l'exécution de onActivity résult.

AndroidManifest.xml :

Descriptif général de l'application :

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
   package="com.example.app"
   android:versionCode="5"
   android:versionName="MyHoneycomb" >
```

xmlns :android : descriptif de la grammaire utilisée pour le préfixe android
package : Package de vos sources (classes java)
versionCode : numéro de version de l'application
versionName : nom complet de la version

Dans <manifest></manifest> :

```
<uses-sdk android:minSdkVersion="11" android:targetSdkVersion="11" />
```

minSdkVersion : SDK minimal pour installer l'application
targetSdkVersion : SDK optimal pour l'application

```
<uses-feature android:name="android.hardware.camera" />
```

Permet la déclaration pour le play store des conditions que doit remplir l'appareil android pour installer l'application (dans ce cas, nécessité d'avoir un appareil photo)

Gestion des permissions :

A placer dans <manifest></manifest>.

Permissions diverses utilisées :

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>                    <!—Permission web -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>      <!—Permission GPS Approximatif -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>        <!—Permission GPS Précis -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />                      <!—Permission écriture/lecture
SDCARD -->
<uses-permission android:name="android.permission.CAMERA" />                                      <!—Permission utilisation camera --
>
< !—Etc… -->
```

Si les permissions ne sont pas ajoutées, cela provoque des exceptions

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name" >
```

icon : Définit l'icône de l'application dans le menu de l'appareil android (va rechercher dans les drawable un fichier nommé ic_launcher.png)
label : Définit le nom de l'application dans le menu de l'appareil android (va rechercher dans le fichier strings.xml la string correspondant à app_name)

Dans <application></application> :

Activity : Voir dans la page 2 Activity
Services : Voir + …

Communiquer avec un layout :

```
public class MonActivite extends Activity implements
View.OnClickListener {
    private Button btnValider ;

            @Override
            public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            btnValider = (Button) findViewById(R.id.btnValiderID);
            btnValider.setOnClickListener(this) ;
            }
            @Override
            public void onClick(View v) {
            Toast.makeText(getApplicationContext(),"Bouton
cliqué",Toast.LENGTH_SHORT).show();
            }
}
```

L'application assigne dans ce cas l'instance du bouton avec l'id btnValiderID du layout main dans la variable btnValider. On écoute ensuite l'évènement du bouton « valider » sur l'instance de l'activité.
L'activité est capable d'écouter les évènements du bouton car elle implémente l'interface. View.OnClickListener. (L'implémentation de l'interface OnClickListener étant la méthode overridée onClick)
Toast est une classe permettant l'affichage de petits messages en popup disparaissant automatiquement.

L'effet de ce bout de code fait donc que lorsque l'on appuie sur le btnValiderID, on affiche une petite popup ayant le texte « Bouton cliqué ».

Layout associé :

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <Button
      android:id="@+id/btnValiderID"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@color/ black "
      android:text="@string/valider" >
   </Button>
</LinearLayout>
```

On définit un LinearLayout qui fait que tous les éléments vont être affichés les uns en dessous des autres.
Le LinearLayout a pour largeur et pour hauteur la taille de son parent (fill_parent).
Le Button défini a l'id btnValiderID, une largeur et une hauteur qui s'adapte à son contenu (wrap_content), comme contenu la valeur dans strings.xml de « valider », la couleur « black» comme background.

strings.xml :

Fichier défini dans tous les répertoires values de l'application (values, values-en, values-it, etc…)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">Mon premier projet</string>
   <string name="valider">Je valide le formulaire</string>
   <string name="action_settings">Paramètres</string>
</resources>
```

Chaque string notée a sa clef qui lui permet d'être recherchée dans un layout ou dans votre code Java (programmatically) via la méthode getResources().getString(R.string.valider) d'une activité.

Voir +…

colors.xml :

Permet de définir des couleurs communes à l'application accessible via R.color.(*) ou @color/(*)

```
<resources>
   <color name="black ">#66000000</color>
</resources>
```

Pour définir la couleur noire dans un layout :
android:background="@color/ black "
Programmatically :
btn.setBackgroundColor(R.color.black);

Voir +…

Utiliser les ListView :

```
public class MainActivity extends Activity implements AdapterView.OnItemClickListener{
   private ListView lvMyListView;
   private String[] listItems = {"item 1", "item 2 ", "list", "android", "item 3", "foobar", "bar", };
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      lvMyListView = (ListView)findViewById(R.id.lvMyListView);
      lvMyListView.setAdapter(new ArrayAdapter(getApplicationContext(),
android.R.layout.simple_list_item_1, listItems));
      lvMyListView.setOnItemClickListener(this);
   }

   @Override
   public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
      if (position == 1)
      {
         Toast.makeText(getApplicationContext(),"J'ai sélectionné l'item
1",Toast.LENGTH_SHORT).show();
         Log.d("monApplication","J'ai sélectionné l'item 1");
      }
   }
}
```

Les listview fonctionnent sur le même système que les autres composants des layout sauf pour une certaine particularité :
Les adapter.
Une ListView, pour afficher des éléments doit être rattachée à un adapter. C'est l'adapter qui servira à afficher chaque ligne de la liste et qui contiendra donc les données de la liste.
L'adapter doit implémenter l'interface ListAdapter.
Un exemple simple est l'ArrayAdapter qui avec comme paramètre le contexte, la manière d'afficher la liste (sous forme d'id de ressource) et un simple tableau de String peut afficher les éléments.
Pour personnaliser l'ArrayAdapter, il est alors possible de changer la manière d'afficher les lignes.

Il est possible de personnaliser une liste en redéfinissant un Adapter (soit en héritant d'un Adapter existant, soit en implémentant l'interface ListAdapter).

Exemple d'adapter affichant une icône d'iPhone si le texte de la ligne commence par « iPhone » autrement un icône de Windows :

(Les icônes sont recherchées dans les ressources)

```
public class MySimpleArrayAdapter extends ArrayAdapter<String> {
   private final Context context;
   private final String[] values;
   public MySimpleArrayAdapter(Context context, String[] values) {
      super(context, R.layout.rowlayout, values);
      this.context = context;
      this.values = values;
   }
   @Override
   public View getView(int position, View convertView, ViewGroup parent) {
      LayoutInflater inflater = (LayoutInflater) context
      .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
      View rowView = inflater.inflate(R.layout.rowlayout, parent, false);
      TextView textView = (TextView) rowView.findViewById(R.id.label);
      ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);
      textView.setText(values[position]);
      String s = values[position];
      if (s.startsWith("iPhone")) {
         imageView.setImageResource(R.drawable.no);
      } else {
         imageView.setImageResource(R.drawable.ok);
      }
      return rowView;
   }
}
```

Redéfinir la méthode View getView(…) permet la modification de la vue de la ligne.

La classe LayoutInflater permet de créer une vue grâce aux layouts des ressources.
Dans le layout rowlayout, on récupère donc la TextView label et l'ImageView. On insère le texte dans la TextView via setText() et l'image correspondant à la ressource R.drawable.no dans l'ImageView via setImageResource().

Comme le demande la redéfinition de la méthode View, on renvoie à la fin la vue que l'on souhaite afficher.

Pour utiliser ensuite cet Adapter, il suffit de réutiliser la méthode :

```
lvMyListView.setAdapter(new ArrayAdapter(getApplicationContext(),  android.R.layout.simple_list_item_1, listItems));
```

AsyncTask

Classe à hériter permettant la création de tâches asynchrones, c'est-à-dire de taches s'effectuant en même temps que le fonctionnement de l'application normal. (Thread différent)
Plusieurs opérations permettent de récupérer le Thread principal (UI Thread), + de détails ci-dessous.
Les tâches asynchrones sont nécessaires pour ne pas faire attendre l'utilisateur lorsque des longues opérations se passent (calcul long, récupération de données sur internet, opération sur bases de données, etc…)

Exemple d'AsyncTask :

La fonction suivante permet de faire la somme des paramètres envoyés de manière asynchrone puis l'affiche avec une petite popup.

```
public class ExempleSommeTacheAsynchrone extends AsyncTask<Long,Integer, Double> {
   private Context myContext;
   public ExempleSommeTacheAsynchrone(Context myContext)
   {
      this.myContext = myContext;
   }
   @Override
   protected void onPreExecute() {
      super.onPreExecute();
      Toast.makeText(myContext, "Début tache", Toast.LENGTH_SHORT).show();
   }
   @Override
   protected void onProgressUpdate(Integer... values){
      super.onProgressUpdate(values);
      Toast.makeText(myContext, "Avancement : "+values.toString()+"/100",
Toast.LENGTH_SHORT).show();
   }
   @Override
   protected Double doInBackground(Long... arg0)  {
      double retour = 0;
      for(int i=0;i<arg0.length;i++){
         retour += arg0[i];
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {}
         publishProgress(new Integer((i*100)/arg0.length));
      }
      return retour;
   }
   @Override
   protected void onPostExecute(Double result) {
      Toast.makeText(myContext, "Somme = "+result.toString(),
Toast.LENGTH_SHORT).show();
   }
}
```

Les types définis dans l'héritage se retrouvent à plusieurs endroits de la définition de la classe.

- Long, paramètres d'entrée de l'exécution de la tâche asynchrone, visible dans doInBackground(Long…)
- Integer, type du paramètre utilisé pour l'avancement, visible dans la méthode publishProgress(Integer) et dans la redéfinition de la méthode onProgressUpdate.
- Double, type du paramètre reçu en fin d'exécution de tache asynchrone, visible pour le type de retour de doInBackground et dans la redéfinition de la méthode onPostExecute.

onPreExecute, onProgressUpdate, onPostExecute s'exécutent sur le thread principal.
doInBackground s'exécute sur le thread créé par la tâche.

L'appel de cette tâche asynchrone se fera via le code suivant :

New ExempleSommeTacheAsynchrone(getApplicationContext()).execute(5121L,8454L,68L) ;

Cet appel affichera donc les popups suivantes :

Début tâche, Avancement : 1/100, etc…, Somme = 13643

Appels de webservices

2 méthodes généralement utilisées : GET ou POST

Requêtes POST permettent d'ajouter autant de paramètres que l'on souhaite

Requêtes GET assez souvent limitées par le serveur à cause de la longueur d'une URL

Méthode GET

```
HttpURLConnection urlConnection = null;
try {
   URL url = new URL("http://www.android.com/");
   urlConnection = (HttpURLConnection) url.openConnection();
   InputStream in = new
BufferedInputStream(urlConnection.getInputStream());
   ByteArrayOutputStream bo = new ByteArrayOutputStream();
   int i = in.read();
   while(i != -1) {
      bo.write(i);
      i = in.read();
   }
   String result = bo.toString();
} catch (IOException e) {
   //TODO: Handle error
} finally {
   if(urlConnection!=null){
      urlConnection.disconnect();
   }
}
```

Retourne la réponse de l'URL provenant de la String « url » dans la String « content ».

Méthode POST

```
public String  performPostCall(String requestURL, HashMap<String, String>
postDataParams) {
   URL url;
   String response = "";
   try {
      url = new URL(requestURL);
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      conn.setReadTimeout(15000);
      conn.setConnectTimeout(15000);
      conn.setRequestMethod("POST");
      conn.setDoInput(true);
      conn.setDoOutput(true);
      OutputStream os = conn.getOutputStream();
      BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os, "UTF-8"));
      writer.write(getPostDataString(postDataParams));
      writer.flush();
      writer.close();
      os.close();
      int responseCode=conn.getResponseCode();
      if (responseCode == HttpsURLConnection.HTTP_OK) {
         String line;
         BufferedReader br=new BufferedReader(new
InputStreamReader(conn.getInputStream()));
         while ((line=br.readLine()) != null) {
            response+=line;
         }
      }
      else {
         response="";
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
```

Retourne la réponse de l'URL provenant de la String « url » avec les paramètres mis dans nameValuePairs dans la String « content ».

```
    return response;
}

private String getPostDataString(HashMap<String, String> params) throws
UnsupportedEncodingException {
    StringBuilder result = new StringBuilder();
    boolean first = true;
    for(Map.Entry<String, String> entry : params.entrySet()){
        if (first) first = false;
        else result.append("&");

        result.append(URLEncoder.encode(entry.getKey(), "UTF-8"));
        result.append("=");
        result.append(URLEncoder.encode(entry.getValue(), "UTF-8"));
    }
    return result.toString();
}
```

Desérialisation/sérialisation de JSON

Utilisation de la librairie GSON (Lien) (Equivalent en XML : Xstream (Lien))

Fonctionnement avec des types primitifs

Importer la librairie dans le dossier lib, faire un clic droit -> Add as library

```
class Personne {
  private int age = 24;
  private String nom= "bischof";
  Personne() {
    // Constructeur sans arguments
  }
}
```

Classe à sérialiser/désérialiser

```
{"age":24",nom":"bischof"}
```

Résultante JSON

```
Personne obj = new Personne();
Gson gson = new Gson();
String json = gson.toJson(obj);
```

Sérialisation

```
Personne obj2 = gson.fromJson(json, Personne.class);
```

Déserialisation

Utilisation de types complexes

Exemple de sérialisation/désérialisation d'une date avec des Adapters afin de définir le comportement de GSON lorsqu'il rencontre ces types. Permet de s'adapter à la sérialisation des types complexes provenant des API.

```
private class DateSerializer implements JsonSerializer<Date> {
  public JsonElement serialize(Date src, Type typeOfSrc, JsonSerializationContext
context) {
    return src == null ? null : new JsonPrimitive(src.getTime());
  }
}
```

Création de l'Adapter permettant de sérialiser une date avec un formatage définit via « src.getTime() ».

```
private class DateDeserializer implements JsonDeserializer<Date> {
  public Date deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context)
      throws JsonParseException {
   return json == null ? null : new Date(json.getAsLong());
  }
}
```

Création de l'Adapter permettant de désérialiser une date utilisant le formatage utilisé pour la sérialisation dans l'autre sens.

```
GsonBuilder gsonBuilder = new GsonBuilder();
gsonBuilder.registerTypeAdapter(Date.class, new DateDeserializer());
gsonBuilder.registerTypeAdapter(Date.class, new DateSerializer());
Gson gson =gsonBuilder.create() ;
```

Création de l'objet GSON pour pouvoir l'utiliser comme avec les types primitifs par la suite.

Utilisation d'images stockées en local

```
File imgFile = new  File("/sdcard/Images/test_image.jpg");
if(imgFile.exists()){
    Bitmap myBitmap =
BitmapFactory.decodeFile(imgFile.getAbsolutePath());
    ImageView myImage = (ImageView)
findViewById(R.id.imageviewTest);
    myImage.setImageBitmap(myBitmap);
}
```

Utilisation de BitmapFactory pour récupérer les données de l'image puis utilisation de setImageBitmap pour mettre ces données dans une ImageView.
La méthode setImageBitmap est utilisée dans d'autres composants comme l'ImageButton.

Les différents stockages

4 types de stockages :

- SharedPreferences, utilisé pour les paramètres
- Stockage interne, utilisé pour de petits fichiers
- Stockage externe, utilisé pour les grands fichiers
- Bases de données (via système de fichiers externe en général)

(Voir +…)

Stockage des préférences

Permet un stockage simple des données primitives, basé sur un fonctionnement par clef/valeurs.

```
public static final String PREFS_NAME = "MyPrefsFile";
….
    SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
    SharedPreferences.Editor editor = settings.edit();
    editor.putBoolean("silentMode", mSilentMode);

    // Commit the edits!
    editor.commit();
```

Insertion de données
PREFS_NAME correspond au nom du fichier de préférence, 0 correspond au mode de lecture/écriture
(+ d'infos…)

```
SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
boolean silent = settings.getBoolean("silentMode", false);
```

Lecture du boolean silentMode dans les sharedPreferences, si ce booléen n'existe pas, silent vaudra false par défaut

Stockage dans l'internal storage

Fichiers privés à l'application, supprimés lors de la désinstallation de l'application. Espace disponible faible en taille, pour les gros fichiers, préférer l'external storage.

L'URI à utiliser pour écrire un fichier sur l'internal storage est une URI relative.

Ecriture d'un fichier texte

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME,
Context.MODE_PRIVATE);
PrintWriter pw = new PrintWriter(fos);
pw.println(string);
pw.flush();
pw.close();
fos.close();
```

Lecture d'un fichier texte

```
String FILENAME = "hello_file";
FileInputStream in = openFileInput(FILENAME); InputStreamReader
inputStreamReader = new InputStreamReader(in);
BufferedReader bufferedReader = new
BufferedReader(inputStreamReader);
StringBuilder sb = new StringBuilder();
String line;
while ((line = bufferedReader.readLine()) != null) { sb.append(line);
}
```

Stockage dans l'external storage

Il est nécessaire pour une application utilisant l'external storage de déclarer son utilisation dans les autorisations de l'AndroidManifest.xml via :

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Il est important de vérifier la disponibilité du stockage externe. En effet, la carte de stockage peut être enlevée ou le portable connecté en mode clef USB sur un pc, ce qui empêche l'accès aux données du stockage externe.

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
   String state = Environment.getExternalStorageState();
   if (Environment.MEDIA_MOUNTED.equals(state)) {
      return true;
   }
   return false;
}
```

Deux manières de stocker les fichiers sur l'external storage :

- Stockage dans les dossiers privés prévus pour l'application
- Stockage dans un répertoire choisis par le développeur

Fichiers privés

Remarque :

- Permet de ne pas indexer ces fichiers dans le MediaStore (Galerie photo, lecteur de musique, etc…)
- Supprimés à la désinstallation de l'application

Lecture du fichier « myData.txt » dans le dossier privé Downloads/test

```
File dir = new File (getExternalFilesDir
(Environment.DIRECTORY_DOWNLOADS)+"/test/");
   dir.mkdirs();
   File file = new File(dir, "myData.txt");
   FileOutputStream f = null;
   try {
      f = new FileOutputStream(file);
      PrintWriter pw = new PrintWriter(f);
      pw.println("Hi , How are you");
      pw.println("Hello");
      pw.flush();
```

```
      pw.close();
      f.close();
    } catch (FileNotFoundException e) {
      //TODO HANDLER
    } catch (IOException e) {
      //TODO HANDLER
    }
```

Fichiers publics

Accessible depuis les autres applications, automatiquement indexé via le MediaStore.

```
File dir = new File (Environment.getExternalStorageDirectory()+"/test/");
   dir.mkdirs();
   File file = new File(dir, "myData.txt");
   FileOutputStream f = null;
   try {
      f = new FileOutputStream(file);
      PrintWriter pw = new PrintWriter(f);
      pw.println("Hi , How are you");
      pw.println("Hello");
      pw.flush();
      pw.close();
      f.close();
   } catch (FileNotFoundException e) {
      //TODO HANDLER
   } catch (IOException e) {
      //TODO HANDLER
   }
```

Environment.getExternalStorageDirectory() permet de récupérer le répertoire source du stockage externe.
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES) permet de récupérer le répertoire public utilisé pour le stockage des images.

Récupération de fichiers sur internet

```
public void downloadFromUrl(String fileURL, String fileName) {
      try {
         URL url = new URL( fileURL);
         File file = new File(fileName);
         file.createNewFile();
           /* Open a connection to that URL. */
         URLConnection ucon = url.openConnection();
           /* Define InputStreams to read from the URLConnection.*/
         InputStream is = ucon.getInputStream();
           /* Read bytes to the Buffer until there is nothing more to
read(-1) and write on the fly in the file.*/
         FileOutputStream fos = new FileOutputStream(file);
         final int BUFFER_SIZE = 23 * 1024;
         BufferedInputStream bis = new BufferedInputStream(is,
BUFFER_SIZE);
         byte[] baf = new byte[BUFFER_SIZE];
         int actual = 0;
         while (actual != -1) {
            fos.write(baf, 0, actual);
            actual = bis.read(baf, 0, BUFFER_SIZE);
         }
         fos.close();
      } catch (IOException e) {
        //TODO HANDLER
      }
   }
```

Téléchargement d'un fichier fileURL dans le répertoire et avec le nom de fichier fileName
4 étapes :

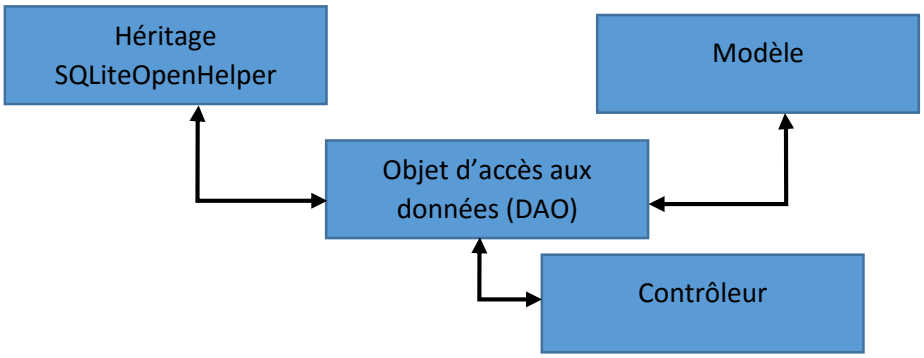
- Création de l'URLConnection
- Récupération de son stream de données
- Création d'un stream de sortie qui écrira sur le système de fichiers
- Utilisation d'un BufferedInputStream pour écrire au fur et à mesure du téléchargement le contenu dans le fichier de destination

Stockage en base de données

Base de données SQLite :

- Lightweight
- Serverless
- Possibilité d'utilisation des triggers
- Zero-configuration
- Limité dans ses types de données (Voir +…)

Organisation des classes pour l'utilisation d'une BDD sur Android :



La classe héritée de SQLiteOpenHelper s'occupe de créer/gérer la structure de la base de données et son emplacement dans le système de fichiers.

L'objet d'accès aux données permet de faire les opérations de CRUD (Create,Read,Update,Delete) sur la base et s'occupe de créer des objets correspondants au modèle qu'il transfère au contrôleur.

Exemple de base de données :

```
public class ExempleDB extends SQLiteOpenHelper {
   private static final int DATABASE_VERSION = 2;
   private static final String DATABASE_NAME = "MyDB.db";
   public static final String HOMME_TABLE_NAME = "Homme";
   public static final String KEY_ID = "id";
   public static final String KEY_NAME = "name";
   public static final String KEY_AGE = "age";
   private static final String HOMME_TABLE_CREATE = "CREATE TABLE " + HOMME_TABLE_NAME + "
(" + KEY_ID + " TEXT, " +
            KEY_NAME + " TEXT, " + KEY_AGE + " TEXT);";
   public ExempleDB(Context context) {
      super(context, Environment.getExternalStorageDirectory()+"/"+DATABASE_NAME, null,
DATABASE_VERSION);
   }
   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL(HOMME_TABLE_CREATE);
   }
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {db.execSQL("DROP
TABLE IF EXISTS " + HOMME_TABLE_NAME);
      db.execSQL("DROP TABLE IF EXISTS " + HOMME_TABLE_NAME);
      onCreate(db);
   }
}
```

La méthode onCreate permet de définir la structure de la base de données via db.execSQL(String structure)

La méthode onUpgrade permet de définir lors de la mise à jour d'une application, le comportement de la base de données à sa réouverture. Dans ce cas, on supprime tout et recrée tout.

```
public class Homme {
   private UUID id;
   private String nom;
   private int age;
/*Getters and setters…*/ }
```

Définition d'une classe simple Homme

```
public class HommeDataSource {
   // Database fields
   private SQLiteDatabase database;
   private ExempleDB dbHelper;
   private String[] allColumns = { ExempleDB.KEY_ID,ExempleDB.KEY_NAME,
         ExempleDB.KEY_AGE };
   public HommeDataSource(Context context) {
      dbHelper = new ExempleDB(context);
   }
   public void open() throws SQLException {
      database = dbHelper.getWritableDatabase();
   }
   public void close() {
      dbHelper.close();
   }
public Homme createHomme(String nom,int age) {
   ContentValues values = new ContentValues();
   values.put(ExempleDB.KEY_NAME, nom);
   values.put(ExempleDB.KEY_AGE, age);
   UUID newID = UUID.randomUUID();
   values.put(ExempleDB.KEY_ID, newID.toString());
   database.insert(ExempleDB.HOMME_TABLE_NAME, null,
         values);
   Cursor cursor = database.query(ExempleDB.HOMME_TABLE_NAME, allColumns,
ExempleDB.KEY_ID + " = \"" + newID.toString()+"\"", null,
         null, null, null);
   cursor.moveToFirst();
   Homme newHomme = cursorToHomme(cursor);
   cursor.close();
   return newHomme;
}
public List<Homme> getAllHommes() {
      List<Homme> lesHommes = new ArrayList<Homme>();
      Cursor cursor = database.query(ExempleDB.HOMME_TABLE_NAME,
            allColumns, null, null, null, null, null);
      cursor.moveToFirst();
      while (!cursor.isAfterLast()) {
         Homme unHomme = cursorToHomme(cursor);
         lesHommes.add(unHomme);
         cursor.moveToNext();
      }
      // make sure to close the cursor
      cursor.close();
      return lesHommes;
   }
   private Homme cursorToHomme(Cursor cursor) {
      Homme comment = new Homme();
      String result = cursor.getString(0);
      comment.setId(UUID.fromString(result));
      comment.setNom(cursor.getString(1));
      comment.setAge(cursor.getInt(2));
      return comment;
   }
public void deleteHomme(Homme unHomme) {
      UUID id = unHomme.getId();
      database.delete(ExempleDB.HOMME_TABLE_NAME, ExempleDB.KEY_ID
            + " = \"" + id.toString()+"\"", null);
   }
}
```

La méthode open permet de récupérer une instance de la base de données dans database. La méthode close permet sa libération

La méthode createHomme permet de d'insérer un homme en bdd à partir des paramètres puis de récupérer cette ligne nouvellement créée.

La méthode insert d'une SQLiteDatabase permet l'insertion d'une ligne grâce aux paramètres String-> nom de la table et aux ContentValues (Voir +…)
La méthode query d'une SQLiteDatabase que l'on retrouve dans createHomme et getAllHommes permet de sélectionner des données dans la bases. Les paramètres sont :

- Nom de la table
- Colonnes demandées
- Arguments de sélection (clause where)
- Group by
- Having
- Order by
- Limit

(Voir +…)
Cette méthode retourne un curseur dans lequel il faut boucler pour récupérer les infos. La récupération des infos est centralisée dans « cursorToHomme ».

La suppression d'une ligne se fait via SQLiteDatabase.delete avec comme paramètres :

- Nom de la table
- Clause where
- Arguments de la clause where

Retour un int correspondant au nombre de lignes supprimées

Les Intents extérieurs

Les Intents, en + de permettre la création d'Activity, permettent aussi d'utiliser de nombreuses fonctionnalités des applications tierces d'un appareil Android. Ci-dessous, une liste avec exemple des différentes utilisations possibles :

Utilisation de l'appareil photo d'un Android :

```
private static final int INTENT_PHOTO = 0;
private void takePhoto(_photoFile){
        Uri _fileUri = Uri.fromFile(_photoFile);
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE
);
        intent.putExtra( MediaStore.EXTRA_OUTPUT, _fileUri);
        startActivityForResult(intent, INTENT_PHOTO);
}
protected void onActivityResult(int requestCode, int resultCode, Intent
intent) {
    switch (requestCode) {
      case INTENT_PHOTO:
            if (resultCode == RESULT_OK)
            {
              resizeFile(_photoFile);
              String[] projection = { MediaStore.Images.ImageColumns.SIZE,
                MediaStore.Images.ImageColumns.DISPLAY_NAME,
                MediaStore.Images.ImageColumns.DATA, BaseColumns._ID, };
              Cursor c = null;
              Uri u = MediaStore.Images.Media.EXTERNAL_CONTENT_URI;
              try {
                if (u != null) {
                  c = managedQuery(u, projection, null, null, null);
                }
                if ((c != null) && (c.moveToLast())) {
                  ContentResolver cr = getContentResolver();
                  cr.delete(
MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
                    BaseColumns._ID + "=" + c.getString(3), null);
                }
              } finally {
                if (c != null) {
                  c.close();
                }
              }
            }
            break;
   }
}
```

On appelle l'Intent via la méthode takePhoto qui permet la création d' Intent MediaStore.ACTION_IMAGE_CAPTURE. On spécifie à cet intent l'endroit où sera stocké l'image.

Via l'activity result, on récupère cette image pour la redimensionner (l'image renvoyée par l'appareil photo est en grande qualité, il est conseillé de la redimensionner, surtout si elle doit être par la suite uploadée)

Sur certains appareils android, lorsqu'un Intent photo est lancé, les photos prisent se retrouvent stockées à la fois à l'URI _photoFile et dans la galerie. Ce qui n'est pas forcement souhaitable. Afin de parer à cela, les lignes suivant le redimensionnement de l'image suppriment de la galerie l'image qui vient d'être prise.

La permission pour écrire sur la sdcard est nécessaire pour la prise de photo :

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

Utilisation de la fonction Share :

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

Sur le même principe que l'Intent photo, sans result cette fois-ci.

Utilisation de la fonction appel :

```
String uri = "tel:" + pt.getTel().trim() ;
Intent intent = new Intent(Intent.ACTION_CALL);
intent.setData(Uri.parse(uri));
startActivity(intent);
```

Nécessite :

```
<uses-permission android:name="android.permission.CALL_PHONE"
/>
```

Les événements

```
public interface OnNewsUpdateListener
{
   public void onNewsUpdate(String news);
}
```

Création du listener

```
public class NewsProvider{
      private ArrayList<onNewsUpdateListener> listeners = new
ArrayList<onNewsUpdateListener> ();
   public void setOnNewsUpdateListener (onNewsUpdateListener
listener)
      {
      // Store the listener object
      this.listeners.add(listener);
      }
   private void newsDownloadCompleted(String myNews){
         for (onNewsUpdateListener oneListener : listeners)
         {
            oneListener.onNewsUpdate(myNews);
         }
      }
}
```

Création de la classe levant les événements

La méthode setOnNewsUpdateListener permet d'ajouter un listener à l'événement

Lorsque l'on veut lever l'événement, on appelle pour chaque listener la méthode onNewsUpdate.

```
public class NewsActivity extends Activity {
   TextView tvNews;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      /* ...*/
      NewsProvider np = new NewsProvider();
      np.setOnNewsUpdateListener(new onNewsUpdateListener() {
         @Override
         public void onNewsUpdate(String news) {
            /* Do some stuff... */
         }
      });
   }
}
```

Exemple de classe utilisant l'écouteur et donc écoutant son événement

Les layout inflater

Il arrive que l'on ait besoin de créer une vue dans le code Java (programmatically). Pour pouvoir créer cette vue à partir d'un layout, il faut faire appel aux LayoutInflater.

Exemple :

```
LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
View rowView = inflater.inflate(R.layout.rowlayout, parent, false);
```

Ces deux lignes permettent de retourner une vue à partir du layout rowlayout.

Les librairies

Plusieurs librairies sont très utiles en Android. Réinventer la roue n'est pas le but du développement.

La librairie GSON

ci-dessus

La librairie MintExpress

Librairie de BugTracking. Permet de recevoir des rapports d'erreurs en cas de crash de l'application sur les appareils l'ayant installé. Les rapports sont en général assez détaillés et permettent de retrouver assez facilement le problème. L'utilisation du portail MintExpress est gratuit pour un certains nombres d'utilisateurs (<1k).
Le SDK fonctionne grâce à un Singleton, il faut donc faire attention car celui-ci peut être désinstancié si l'utilisateur va sur une autre application qui demande beaucoup de ressources et retourne après sur votre application. Il faut donc penser à le réinstancier au retour sur l'appli.

Site web : https://mint.splunk.com

La librairie Google Analytics

Librairie de statistiques. Permet de savoir quels sont les vues affichées sur votre application et de trouver donc les points d'intérêts pour les utilisateurs dans votre application. Cette librairie comporte aussi un système de BugTracking mais beaucoup moins abouti que BugSense.

Ce SDK est toujours en béta mais comme pour Gmail, les béta durent longtemps chez Google….

Site web : https://developers.google.com/analytics