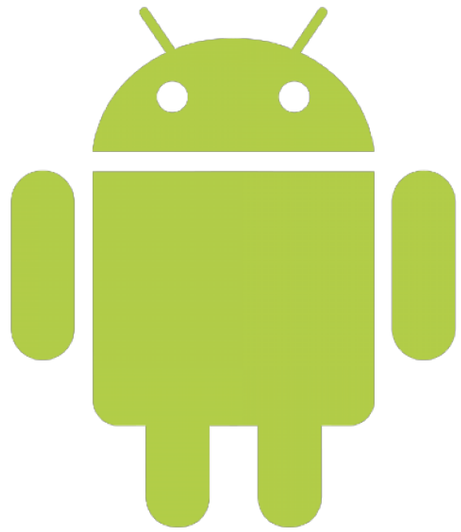
Android
Lire et enregistrer des données

Jean-Marc Lecarpentier
Université de Caen

Au programme

- Quelles solutions pour enregistrer les données d'une application
- Stockage simple : Preferences
- Fichiers internes à une App
- Utilisation de base de données : SQLite



Stockage de données

- Préférences : données clé-valeur
- Mémoire interne : système de fichiers
- Mémoire externe : système de fichiers éjectable
- Base de données : SQLite
- Réseau : serveur dédié, Web Service, Cloud, etc

Preferences vs. *settings*

⚠ *preferences* ≠ *settings* mais *settings* ⊂ *preferences*

- *Preferences* : stocker un ensemble de paires clé/valeur
- *Settings* : modifiables par l'utilisateur et possède son UI
- Données accessibles par tous les composants de l'application ou non
- Données NON partagées avec les autres apps
- ex. : choix du thème pour l'app ⇒ *setting*

  sauvegarder le dernier "high score" ⇒ *preference*

Gestion des préférences

- Objet SharedPreferences gère les paires clé,valeur
- Une application peut avoir N “fichiers” de paires partagés par tous ses composants
- Toute Activity peut avoir un fichier dédié
- Valeurs de type primitif (int, long, float, bool, string)

Lire les preferences

- Obtenir un objet SharedPreferences :
    - getSharedPreferences(fichier, 0) : charger les valeurs de fichier en mode PRIVATE (cas général)
    - getPreferences() : charger les valeurs de l'Activity
- lire une valeur : get<type>(clé, défaut)
  ex. getString("nom", "Michel"); getInt("score", 0);
- Méthode contains(cle) pour tester la présence d'une clé

Modifier les préférences

- Utiliser l'éditeur de l'objet SharedPreferences

```
SharedPreferences mesPrefs = getSharedPreferences("toto", 0);
SharedPreferences.Editor editeur = mesPrefs.edit();
```

- Méthodes put<type>(clé, valeur)

```
editeur.putBoolean("silentMode", false);
 editeur.putString("nom", "Marie");
```

- Enregistrer les changements avec commit

```
 editeur.commit();
```

- Capter un changement dans les préférences : registerOnSharedPreferenceChangeListener()

Android et Fichiers

- Fichiers internes à l'application (read only)
- Mémoire interne
    - Fichiers accessibles seulement par l'application
    - Supprimés si l'app est désinstallée
- Support externe
    - Fichiers "world readable"
    - Aucune garantie de disponibilité

Fichiers de l’application

- Stockés dans res/raw/
- Disponibles avec l’id R.raw.<nom_fichier>

```
Resources myResources = getResources();
InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```

- Read-only : non modifiables par l’application
- Utile pour stocker des ressources volumineuses pour l’application (dictionnaire, etc)

Mémoire interne

- Lire un fichier : openFileInput(fichier)

```
FileInputStream fis = openFileInput("toto.txt");
while ((content = fis.read()) != -1) {
    System.out.print((char) content);
}
fis.close();
```

- Écrire un fichier : openFileOutput(fichier, mode)

```
String string = "Bonjour Android !";
try {
    FileOutputStream fos = openFileOutput("toto.txt", MODE_PRIVATE);
    fos.write(string.getBytes());
    fos.close();
} catch …
// modes disponibles
MODE_PRIVATE, MODE_APPEND, MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE
```

Mémoire interne

- Méthodes utiles : getFilesDir(), getDir(), deleteFile(), fileList(), createTempFile(), getFreeSpace(), getTotalSpace()

- Gestion de la mémoire cache de l'application : getCacheDir()

- Fichiers cache à "nettoyer" régulièrement

Lister les fichiers de l'app

- Utiliser l'exécutable adb pour se connecter sur l'émulateur ou le *device*
- Commande adb -s *emulator-name* shell pour lancer un terminal
- Commande adb devices pour voir la liste des émulateurs ou *devices*
- Utiliser run-as *packagename* pour accéder au dossier de l'app

- Exemple sur l'émulateur

```
$ adb devices
List of devices attached
emulator-5554      device
$ adb -s emulator-5554 shell
generic_x86:/ $ run-as fr.greyc.users.lecarpentier.demostorage
generic_x86:/data/data/fr.greyc.users.lecarpentier.demostorage $ ls -l
total 40
drwxrwx--x 2 u0_a74 u0_a74 4096 2017-01-09 15:54 cache
drwxrwx--x 2 u0_a74 u0_a74 4096 2017-01-09 15:41 code_cache
drwxrwx--x 2 u0_a74 u0_a74 4096 2017-01-09 15:54 databases
drwx------ 3 u0_a74 u0_a74 4096 2017-01-09 15:42 files
drwxrwx--x 2 u0_a74 u0_a74 4096 2017-01-09 15:54 shared_prefs
```

Mémoire externe

- Méthodes getExternalStorageState(), isExternalStorageWritable(), isExternalStorageReadable()

- Déclarer la permission dans le Manifest

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

- Méthodes getExternalStoragePublicDirectory() pour avoir le dossier Musique/, Videos/, etc.

```
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
```

Android et BD

- Android utilise SQLite (http://www.sqlite.org)
- SQLite : BD indépendante, sans serveur, utilisable par les programmes sans configuration spécifique
- Toute application peut créer des bases SQLite
- Bases accessibles par tous ses composants
- Non accessibles en dehors de l'application

Utiliser une BD

- Créer une classe pour la création et l'ouverture de la BD et une autre pour les opérations BD (requêtes, etc)

- BD avec n° de version pour prévoir les *upgrades*

- Recommandé : tables avec ID auto_increment
  ⇒ implémenter BaseColumns

- Ne PAS stocker de fichiers (images, etc) en BD

Connexion et création de BD

- Connexion BD : classe dérivant de SQLiteOpenHelper
- Méthodes close(), getDatabaseName()
- Méthodes callback onCreate(), onUpgrade(), onDowngrade(), onOpen(), onConfigure(), etc
- Ouvrir la base en lecture ou écriture : getWritableDatabase () et getReadableDatabase() renvoient un objet SQLiteDatabase
- Aucune opération faite avant getWritableDatabase()

Connexion et création de BD

```
// Gestion de l'ouverture de la BD
public class MaBaseOpenHelper extends SQLiteOpenHelper {
    // SQL Statement to create a new database.
    private static final String BD_CREATE = "create table ...";

    // constructeur
    public MaBaseOpenHelper(Context context, String name, CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(BD_CREATE);
    }

    // Callback appelé si versions de BD différentes
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // gestion sans états d'âmes : supprimer la table et la recréer
        db.execSQL("DROP TABLE ...");
        onCreate(db);
    }
}
```

Lecture de la BD

- Ouvrir en lecture avec getReadableDatabase()
- Utiliser la méthode query() de SQLiteDatabase
- Requêtes complexes : utiliser SQLiteQueryBuilder
- Retourne un objet Cursor

Lecture de la BD

```
public class MaBaseGestionnaire {
     // Constantes de BD
     private static final String BD_NAME = "mabase.db";
     private static final int BD_VERSION = 1;
     private static final String BD_TABLE_PERS = "personnes";
     private static final String BD_TABLE_SCORE = "scores";
     public static final String COL_PERS_ID = "_id";
     public static final String COL_PERS_NOM = "nom";
     public static final String COL_PERS_PRENOM = "prenom";
     // etc

     // Mon SQLiteOpenHelper
     private MaBaseOpenHelper MaBaseOpenHelper;
     public MaBase(Context context) {
          MaBaseOpenHelper = new MaBaseOpenHelper(context, BD_NAME, null, BD_VERSION);
     }

     // Ajouter méthodes pour requêtes etc
     public Cursor getNoms(String like) {
          // colonnes à retourner
          String[] colonnes = new String[] { COL_PERS_ID, COL_PERS_NOM };
          // where
          String where = COL_PERS_NOM + " LIKE \"%?%\"";
          String whereArgs[] = { like };
          String groupBy = null;
          String having = null;
          String order = COL_PERS_NOM + " ASC";
          String order = null;
          SQLiteDatabase db = MaBaseOpenHelper.getWritableDatabase();
          Cursor cursor = db.query(BD_TABLE_PERS, colonnes, where,
                                   whereArgs, groupBy, having, order);
          return cursor;
     }
}
```

Parcourir les résultats

- Objet Cursor
- Méthodes getCount, moveToFirst, moveToNext, moveToPrevious, getCount, moveToPosition, getPosition, etc
- Méthodes getColumnIndexOrThrow, getColumnName, getColumnNames
- Méthodes get<type>(index_colonne)
- Boucle pour parcourir les résultats

Insérer des données en BD

- Utiliser la classe ContentValues et méthode put
- Méthode insert() de SQLiteDatabase

```
// Insérer un nom, prénom
public void ajouter(String nom, String prenom) {
    ContentValues valeurs = new ContentValues();
    valeurs.put(COL_PERS_NOM, nom);
    valeurs.put(COL_PERS_PRENOM, prenom);

    // Insérer les données
    SQLiteDatabase db = MaBaseOpenHelper.getWritableDatabase();
    // insert() renvoie l'id généré ou -1 en cas d'erreur
    // TO DO gestion erreur
    db.insert(BD_TABLE_PERS, null, valeurs);
}
```

Modifier des données en BD

- Utiliser la classe ContentValues et méthode put
- Méthode update() de SQLiteDatabase

```
// Modifier une entrée
public void modifier(int id, String nom, String prenom) {
    ContentValues valeurs = new ContentValues();
    valeurs.put(COL_PERS_NOM, nom);
    valeurs.put(COL_PERS_PRENOM, prenom);

    String where = COL_PERS_ID + " = ?";
    String whereArgs[] = {  Integer.toString(id) };

    // Modifier
    SQLiteDatabase db = MaBaseOpenHelper.getWritableDatabase();
    // update() renvoie le nombre de lignes modifiées
    db.update(BD_TABLE_PERS, valeurs, where, whereArgs);
}
```

Supprimer des données

- Méthode delete() de SQLiteDatabase

```
// Supprimer une entrée
public void supprimer(Int id) {
    String where = COL_PERS_ID + " = ?";
    String whereArgs[] = {  Integer.toString(id) };

    SQLiteDatabase db = MaBaseOpenHelper.getWritableDatabase();
    // delete() renvoie le nombre de lignes supprimées s'il y a une clause WHERE
    // renvoie 0 sinon. Passer une clause vraie pour avoir le nombre de suppressions
    db.delete(BD_TABLE_PERS, where, whereArgs);
}
```

Inspecter une base

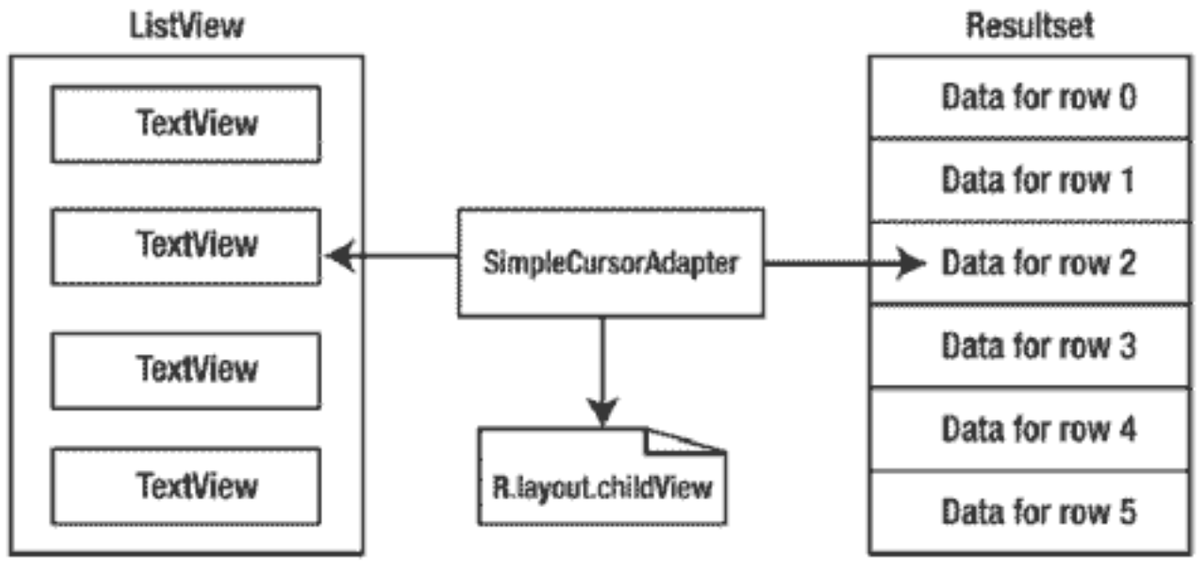
- Utiliser adb (Android Debug Bridge) pour lancer un terminal sur l'émulateur (identifié par son n°)

```
$ adb -s emulator-5554 shell
$ run-as fr.jml.demo
$ cd databases
$ sqlite3 mabase.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
personnes scores
sqlite> .schema personnes
CREATE TABLE personnes (_id INTEGER PRIMARY KEY,nom VARCHAR(255), …)
 sqlite> select * from personnes;
1|Lecarpentier|Jean-Marc
2|Safi|Waseem
5|Lambert|Jean-Luc
```

Application : ListView & BD

- Utiliser un ListView dans un layout
- Utilisation de SimpleCursorAdapter
- Gestion des évènements sur un item

SimpleCursorAdapter



- Établir correspondance entre colonnes de résultat BD et id° des éléments de vue d'un item

- Gestion du clic sur un item

ListActivity

- Permet d'avoir une liste de type ListView
- Sans avoir à créer un Layout ⇒ objet View créé automatiquement avec un ListView plein écran
- À combiner avec les layouts prédéfinis pour les items de liste
- Charger une liste avec un adapteur ⇒ utiliser setListAdapter()
- Gère en natif les clics sur un item

RecyclerView

- Android 5
- Gestion plus fine des listes, indépendante du layout
- Performances optimisées avec ViewHolder pour recycler les items non visibles
- https://guides.codepath.com/android/Using-the-RecyclerView

En bref

- Gestion des données
- Preferences
- Base de données

- Listes simples de données
- À voir :
  - RecyclerView
  - Room Persistence Library : couche d’abstraction pour les bases de données SQLite (Entités & DAO)