

Par NIDHAL JELASSI
jelassi.nidhal@gmail.com

DÉVELOPPEMENT MOBILE

PROG. ANDROID



Chapitre 7 : Persistance des données

TYPES DE STOCKAGE DE DONNÉES

- SharedPreferences
- Stockage externe
- Stockage interne
- Base de données SQLite

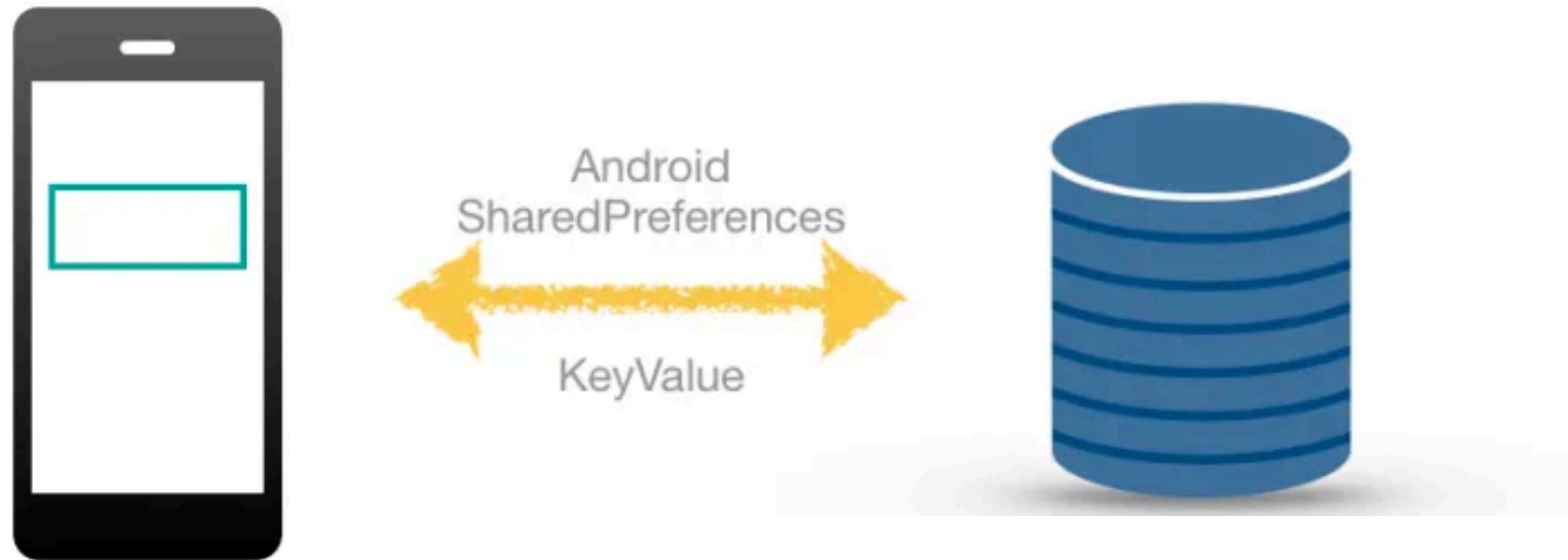
LE TYPE IDÉAL ?

- ▶ C'est une question dont la réponse dépend du contexte de l'application et de ses besoins spécifiques.
- ▶ Choisir un type pour persister ses données dépend, en effet, de plusieurs facteurs:
 - ▶ **Espace** mémoire requis
 - ▶ **Structuration** des données
 - ▶ **Accessibilité** des données (privés, utilisables par d'autres applications, etc..)

SHARED PREFERENCES

SHARED PREFERENCES

- ▶ Ce type de persistance consiste en un framework général qui permet de sauvegarder et extraire des paires clef-valeur persistantes de types primitifs



- ▶ Il est généralement utilisé pour sauvegarder les préférences utilisateur (d'où le nom) du genre : la sonnerie préférée, le fond d'écran, certains autres réglages, etc...

VS.SAVEDINSTANCESTATE

Persiste à toutes les sessions utilisateurs (app supprimé, redemarrage terminal,etc.)	Persiste sur une seule et même session
Données qui doivent être mémorisées d'une session à l'autre, telles que les paramètres préférés de l'utilisateur ou son score de jeu.	Données qui ne doivent pas être mémorisées d'une session à une autre, telles que l'onglet actuellement sélectionné
Les données sont privées pour l'application.	Les données sont privées pour l'application.
Petit nombre de paires clé / valeur.	Petit nombre de paires clé / valeur.

SHARED PREFERENCES

- ▶ Une fois qu'on a utilisé objet **SharedPreferences** pour sauvegarder des données, plusieurs méthodes sont proposées pour lire cet objet. Parmi les plus connues, il y a :
- ▶ `getSharedPreferences`: pour utiliser plusieurs fichiers de préférences identifiables par nom
- ▶ `getPreferences`: pour utiliser un seul fichier de préférences

SHARED PREFERENCES

- ▶ A partir d'un objet `SharedPreferences`, nous pouvons ajouter des valeurs, de la manière suivante :
- ▶ Invoquer `edit()` pour obtenir un objet `SharedPreferences.Editor`.
- ▶ Ajouter des valeurs avec des méthodes, tel que `putBoolean()` et `putString()`.
- ▶ Valider les nouvelles valeurs avec :
 - ▶ `apply()` (sauvegarde asynchrone)
 - ▶ `commit()` (sauvegarde synchrone)

EXAMPLE

```
public void save(View v) {  
    SharedPreferences.Editor editor = getSharedPreferences(MY_PREFS_NAME, MODE_PRIVATE).edit();  
    editor.putString("name", nameInput.getText().toString());  
    editor.putInt("age", Integer.valueOf(ageInput.getText().toString()));  
    editor.commit();  
  
    SharedPreferences prefs = getSharedPreferences(MY_PREFS_NAME, MODE_PRIVATE);  
    String name = prefs.getString("name", "No name defined");  
    int age = prefs.getInt("age", 0);  
  
    nameText.setText("Name: " + name);  
    ageText.setText("Age: " + age);  
}
```

STOCKAGE EXTERNE

STOCKAGE EXTERNE

- ▶ Contrairement aux terminaux Apple, par exemple, tous les appareils tournant sous Android supportent un espace de stockage externe qui peut être :
 - ▶ Une carte SD ;
 - ▶ Un support interne non-amovible
- ▶ Les fichiers sauvegardés dans un cet espace de stockage externe sont accessibles à toutes les applications en lecture


STOCKAGE EXTERNE

- Pour effectuer les opérations de lecture et d'écriture sur les fichiers stockés de manière externe, il est indispensable d'avoir les permissions requises dans le fichier « `AndroidManifest.xml` ».

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nidhal.external">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
```

STOCKAGE EXTERNE

- ▶ En plus des permissions à ajouter au fichier `AndroidManifest`, il est impératif de :
- ▶ Vérifier la disponibilité du support de stockage grâce à la méthode `getExternalStorageState`



```
public boolean isWritable(){
    String st = Environment.getExternalStorageState();
    if(Environment.MEDIA_MOUNTED.equals(st)){
        return true;
    }
    return false;
}
```

- ▶ Sauvegarder les fichiers créés.

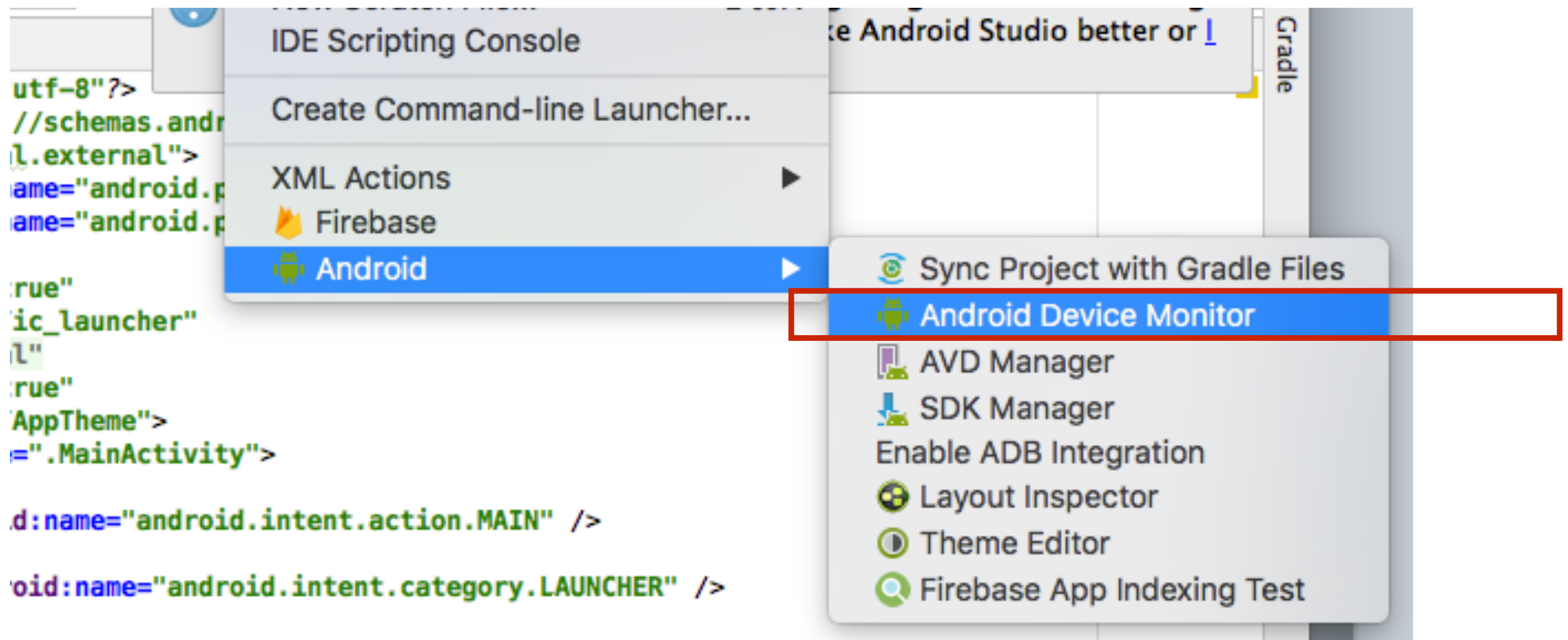
EXTERNE (EXEMPLE)

- Pour Ecrire dans un fichier :

```
public void ExternalSave(View v) throws IOException {  
    if(isWirtable())  
    {  
        File dir = Environment.getExternalStorageDirectory();  
        File f = new File(dir+"/"+externe.txt);  
        FileOutputStream o = new FileOutputStream(f);  
        String msg = "Je suis "+nom.getText()+" "+prenom.getText();  
        o.write(msg.getBytes());  
        o.close();  
    }  
}
```

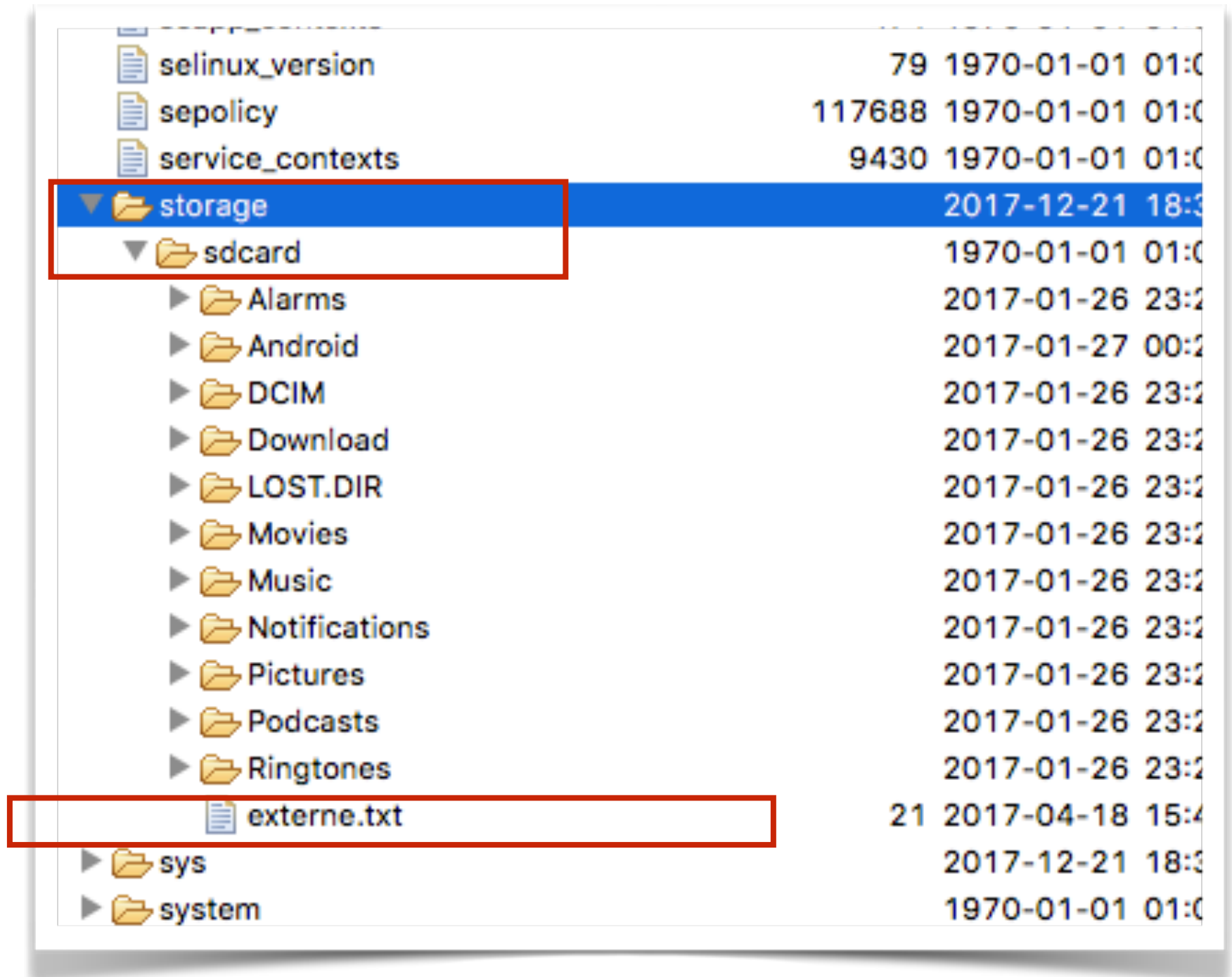
VISUALISATION DES FICHIERS

- Pour visualiser les fichiers que vous avez créé pour votre application, rendez-vous vers le menu Android Device Monitor et cliquer sur l'onglet File Explorer.



VISUALISATION DES FICHIERS

- ▶ Vos fichiers « externes » se trouvent sous le dossier storage/sdcard



selinux_version	79	1970-01-01	01:0
sepolicy	117688	1970-01-01	01:0
service_contexts	9430	1970-01-01	01:0
storage		2017-12-21	18:3
sdcard		1970-01-01	01:0
Alarms		2017-01-26	23:2
Android		2017-01-27	00:2
DCIM		2017-01-26	23:2
Download		2017-01-26	23:2
LOST.DIR		2017-01-26	23:2
Movies		2017-01-26	23:2
Music		2017-01-26	23:2
Notifications		2017-01-26	23:2
Pictures		2017-01-26	23:2
Podcasts		2017-01-26	23:2
Ringtones		2017-01-26	23:2
externe.txt	21	2017-04-18	15:4
sys		2017-12-21	18:3
system		1970-01-01	01:0

**STOCKAGE
INTERNE**

STOCKAGE INTERNE

- ▶ **Principe** : Utiliser la mémoire interne de votre téléphone pour stocker des fichiers contenant vos données.
 - ▶ **Inconvénient** : Le ou les fichiers sauvegardés de cette manière sont exclusivement liés à l'application qui les a créés. Ils sont donc inaccessibles depuis les autres applications.
- ➡ En désinstallant l'application, les fichiers qui lui sont liés sont automatiquement supprimés

STOCKAGE INTERNE

- ▶ Les modes opératoires sont:
- ▶ `MODE_PRIVATE` : Le fichier n'est accessible que par l'application qui l'a créé.
- ▶ `MODE_WORLD_READABLE` : Le fichier est accessible en lecture par les autres applications.
- ▶ `MODE_WORLD_WRITEABLE` : Le fichier est accessible en écriture par les autres applications.
- ▶ `MODE_APPEND` : Si le fichier existe déjà, les données seront ajoutées à la fin.

STOCKAGE INTERNE : ECRIRE

- ▶ Les opérations d'écriture et de lecture à partir d'un fichier enregistré dans la mémoire interne se fait de la manière suivante :
- ▶ Invoquer la méthode `openFileOutput` qui prend en paramètres le nom du fichier et le mode d'ouverture dudit fichier
- ▶ Utiliser la méthode `write()` pour écrire.
- ▶ Fermer le flux avec `close()` (Très important)

STOCKAGE INTERNE : EXEMPLE

- Ecriture dans un fichier en stockage interne :

```
public void Save(View v) throws IOException {  
    FileOutputStream o = openFileOutput("isie.txt", MODE_PRIVATE);  
    String msg = "Je suis "+nom.getText()+" "+prenom.getText();  
    o.write(msg.getBytes());  
    Toast.makeText(this, "Données enregistrés", Toast.LENGTH_LONG).show();  
    o.close();  
}
```

STOCKAGE INTERNE : LECTURE

- ▶ Les opérations d'écriture et de lecture à partir d'un fichier enregistré dans la mémoire interne se fait de la manière suivante :
- ▶ Invoquer la méthode `openFileInput` qui prend en paramètres le nom du fichier.
- ▶ Utiliser la méthode `read()` pour écrire.
- ▶ Fermer le flux avec `close()`.

STOCKAGE INTERNE : EXEMPLE

- Lecture à partir d'un fichier en stockage interne :

```
public void Read(View v) throws IOException {  
    FileInputStream i = openFileInput("isie.txt");  
    StringBuffer buffer = new StringBuffer();  
    int ligne;  
    while((ligne = i.read()) != -1){  
        buffer.append((char) ligne);  
    }  
    Toast.makeText(this, buffer, Toast.LENGTH_LONG).show();  
    i.close();  
}
```

VISUALISATION DES FICHIERS

- ▶ Vos fichiers « internes » se trouvent sous le dossier `data/AppName/files/`

▶	com.android.speechrecorder	2017-01-27 00:27	drwxr-x--x
▶	com.android.systemui	2017-01-27 00:27	drwxr-x--x
▶	com.android.vending	2017-01-27 00:27	drwxr-x--x
▶	com.android.vpndialogs	2017-01-27 00:27	drwxr-x--x
▶	com.android.wallpaper.livepicker	2017-01-27 00:27	drwxr-x--x
▶	com.android.webview	2017-01-27 00:27	drwxr-x--x
▶	com.android.widgetpreview	2017-01-27 00:27	drwxr-x--x
▶	com.example.android.apis	2017-01-27 00:27	drwxr-x--x
▶	com.example.android.livecubes	2017-01-27 00:27	drwxr-x--x
▶	com.example.android.softkeyboard	2017-01-27 00:27	drwxr-x--x
▶	com.example.nidhal.compteur	2017-03-09 08:56	drwxr-x--x
▼	com.example.nidhal.external	2017-04-18 15:19	drwxr-x--x
▶	cache	2017-04-18 15:21	drwxrwx--x
▼	files	2017-04-18 15:20	drwx-----
▶	instant-run	2017-04-18 15:40	drwx-----
	isie.txt	22 2017-04-18 15:21	-rw-rw----
	lib	2017-04-18 15:19	lrwxrwxrwx
▶	com.example.nidhal.firstapp	2017-01-27 00:28	drwxr-x--x
▶	com.example.nidhal.moneyconverter	2017-02-10 12:44	drwxr-x--x
▶	com.example.nidhal.moyenne	2017-02-21 15:41	drwxr-x--x
▶	com.example.nidhal.passage1	2017-04-04 23:28	drwxr-x--x
▶	com.example.nidhal.tp3	2017-02-27 14:46	drwxr-x--x

BASE DE DONNÉES

SQLITE

SQLITE

- ▶ SQLite est une bibliothèque logicielle qui implémente un moteur de base de données SQL léger et sans dépendances externes
- ▶ Il est simplifié pour s'adapter aux terminaux mobiles.
- ▶ Cette simplification consiste en l'absence de :
 - ▶ Gestion des utilisateurs
 - ▶ Réglages
 - ▶ Diversité des types de données

SQLITE

- ▶ Absence de serveur.
- ▶ Toutes les bases de données créées dans une application seront accessibles par nom à travers toute application « interne », mais pas d'une application extérieure.
- ▶ Pour manipuler une base de données SQLite, il est nécessaire de se passer via ces 3 classes :
 - ▶ SQLiteOpenHelper
 - ▶ SQLiteDatabase
 - ▶ Cursor

SQLITEOPENHELPER

- ▶ Sert à créer une base de données de manière simple et efficace.
- ▶ Pour y parvenir, il suffit de suivre la procédure suivante :
 - ▶ Créer une classe qui hérite de SQLiteOpenHelper
 - ▶ Créer la base de données et les tables nécessaires
 - ▶ Implémenter les méthodes suivantes : constructeur, onCreate et onUpgrade.

```
public class MyDBHandler extends SQLiteOpenHelper {
```

```
    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "productDB.db";
    private static final String TABLE_PRODUCTS = "products";
```

```
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_PRODUCTNAME = "productname";
    public static final String COLUMN_QUANTITY = "quantity";
```

```
    public MyDBHandler(Context context, String name,
        SQLiteDatabase.CursorFactory factory, int version) {
        super(context, DATABASE_NAME, factory, DATABASE_VERSION);
    }
```

```
    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_PRODUCTS_TABLE = "CREATE TABLE " +
            TABLE_PRODUCTS + "("
            + COLUMN_ID + " INTEGER PRIMARY KEY," + COLUMN_PRODUCTNAME
            + " TEXT," + COLUMN_QUANTITY + " INTEGER" + ")";
        db.execSQL(CREATE_PRODUCTS_TABLE);
    }
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
        int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_PRODUCTS);
        onCreate(db);
    }
```

Constructeur



OnCreate



OnUpgrade



SQLITEDATABASE

- ▶ Cette classe représente une base de données
- ▶ Elle propose les méthodes nécessaires pour l'exécution des requêtes comme :
- ▶ `execSQL` pour exécuter des requêtes qui ne retournent pas de données. Elle prend en argument une chaîne de caractères.
- ▶ `rawQuery` réservée aux requêtes qui retournent des n-uplets. Sa valeur de retour est un objet de type `Cursor`

CURSOR

- ▶ Représente un n-uplet où sont stockés les résultats d'une requête.
- ▶ Exemple d'utilisation de Cursor :

```
public void ExempleRequete(SQLiteDatabase bdd) {  
    Cursor cursor = bdd.rawQuery("SELECT * FROM NidhaUsers", null);  
  
    try {  
        if (cursor.moveToFirst()) {  
            while (!cursor.isAfterLast()) {  
                cursor.moveToNext();  
            }  
        }  
    } finally {  
        if (cursor != null) cursor.close();  
    }  
}
```



JELASSI.NIDHAL@GMAIL.COM

NIDHAL JELASSI