Persistance et partage de données

Plan

- Introduction
- SharedPreferences
- Stockage interne
- Stockage externe
- Stockage en base de données
- ContentProvider
- Partager vos données avec d’autres applications
- Recevoir des données depuis d’autres applications
- Récupérer des données stockées en ligne

Introduction

- Le framework Android permet de stocker les données nécessaires au bon fonctionnement d'une application de plusieurs manières :
  - SharedPreferences : permet de stocker des données sous forme de couples clé/valeur
  - Fichiers : permet de stocker des données dans des fichiers (créés sur la mémoire interne ou externe de l'appareil)
  - Stockage en base de données : permet de créer une base de données SQLite afin de stocker des données nécessaires au bon fonctionnement d'une application.

SharedPreferences

- La classe SharedPreferences fournit un ensemble de méthodes permettant de stocker et récupérer très facilement un couple clé/valeur avec la particularité de ne contenir que des valeurs primitives (float, int, string, …)
- Ces données restent persistantes jusqu'à la désinstallation de l'application

Récupération d'une instance de SharedPreferences

- Deux méthodes permettent de récupérer une instance de SharedPreferences à partir d'un context:
  - getPreferences(int mode)
    - Cette méthode doit être utilisée si on a besoin d'un seul et unique fichier de préférences dans toute l'application. Il ne permet pas de spécifier le nom du fichier où seront stockées les données SharedPreferences
    - SharedPreferences myPref = getPreferences(Context.MODE_PRIVATE);
    - L'argument mode peut prendre l'une des valeurs suivantes :
      - MODE_PRIVATE : le fichier est privé, donc réservé à l'application
      - MODE_MULTI_PROCESS : permet au fichier d'être utilisé simultanément par plusieurs processus
  - getSharedPreferences(String name, int mode) : à utiliser cette méthode en cas de besoin de plusieurs fichiers de préférences qu'on peut identifier par leur noms

Ajouter des données à un SharedPreferences

- Il faut récupérer une instance de la classe Editor (classe permettant de modifier un SharedPreferences) en utilisant la méthode edit de l'instance de SharedPreferences
  - SharedPreferences.Editor editor = myPref.edit();
- Ensuite, on utilise les méthodes putString(key, value), putInt(key, value) pour ajouter des données
  - editor.putString("cle_String", "valeurString");
  - editor.putInt("cle_int", 5);
- À la fin, on utilise la méthode apply pour sauvegarder les données
  - myPref.apply();

Récupérer des données d'un SharedPreferences

La récupération d'un élémént persisté se fait avec les méthodes **.getString(KEY,DEFAULT_VALUE)**, **.getInt(KEY,DEFAULT_VALUE)**, etc.

Vous pouvez vérifier la présence d'un élément avec **.containsKey(KEY)**

```
1  int myInteger = sharedPreferences.getInt("cle_integer", 0);
2  String myString = sharedPreferences.getString("cle_string", null)
```

Supprimer une valeur d’une collection SharedPreferences

- Pour supprimer une valeur d’une collection SharedPreferences, utiliser la méthode remove de la classe Editor.

Exemple complet

```
public class MainActivity extends AppCompatActivity {
    private static final String PREFS = "PREFS";
    private static final String PREFS_AGE = "PREFS_AGE";
    private static final String PREFS_NAME = "PREFS_NAME";
    SharedPreferences sharedPreferences;

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

        sharedPreferences = getBaseContext().getSharedPreferences(PREFS, MODE_PRIVATE);

        //objectif : sauvegarder 1 seule fois le nom et l'age de l'utilisateur

        //pour cela, on commence par regarder si on a déjà des éléments sauvegardés
        if (sharedPreferences.contains(PREFS_AGE) && sharedPreferences.contains(PREFS_NAME)) {

            int age = sharedPreferences.getInt(PREFS_AGE, 0);
            String name = sharedPreferences.getString(PREFS_NAME, null);

            Toast.makeText(this, "Age: " + age + " name: " + name, Toast.LENGTH_SHORT).show();

        } else {
            //si aucun utilisateur n'est sauvegardé, on ajouter [24,florent]
            SharedPreferences.Editor editor = sharedPreferences.edit();
            editor.putInt(PREFS_AGE, 24);
            editor.putString(PREFS_NAME, "florent");
            editor.apply();
            Toast.makeText(this, "Sauvegardé, relancez l'application pour voir le résultat", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Exercice 1

- Basez-vous sur l'exemple de la diapositive précédente pour écrire une application Android qui affiche deux EditText (nom et age) et un bouton (Enregistrer)
- Quand on clique sur le bouton, l'application enregistre ou modifie dans les préférences le nom et l'âge de l'utilisateur
- Quand on lance l'application, l'application vérifie dans les préférences si le nom et l'âge ont été enregistrés et les affiche dans les EditText appropriés

Le stockage interne

- On peut sauvegarder des fichiers directement sur la mémoire intene du téléphone.
- Par défaut, les fichiers sauvegardés par une application ne sont pas visibles par les autres applications
- Si l’utilisateur désinstalle une application, les fichiers correspondants sur le stockage interne seront supprimés

Exercice 2 – Écriture dans un fichier – Stockage interne

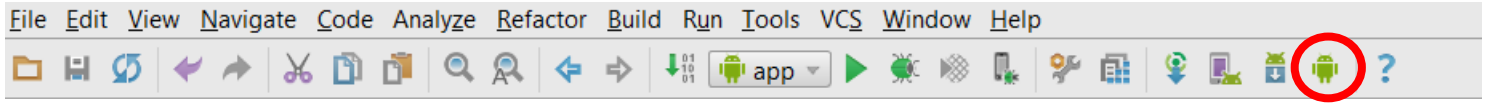
- Taper ce code pour créer un fichier texte en stockage interne et utiliser l'outil Android Device Monitor de l'émulateur pour aller chercher ce fichier

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String fileName = "monFichier.txt";
        String str = "Ceci est un exemple de stockage interne";
        FileOutputStream fos = null;
        try{
            fos = openFileOutput(fileName, Context.MODE_PRIVATE) ;
            fos.write(str.getBytes());
            Toast.makeText(this, "Ecriture dans le fichier terminé", Toast.LENGTH_LONG).show();
            fos.close();
        }
        catch(IOException ex){
            ex.printStackTrace();
        }
    }
}
```

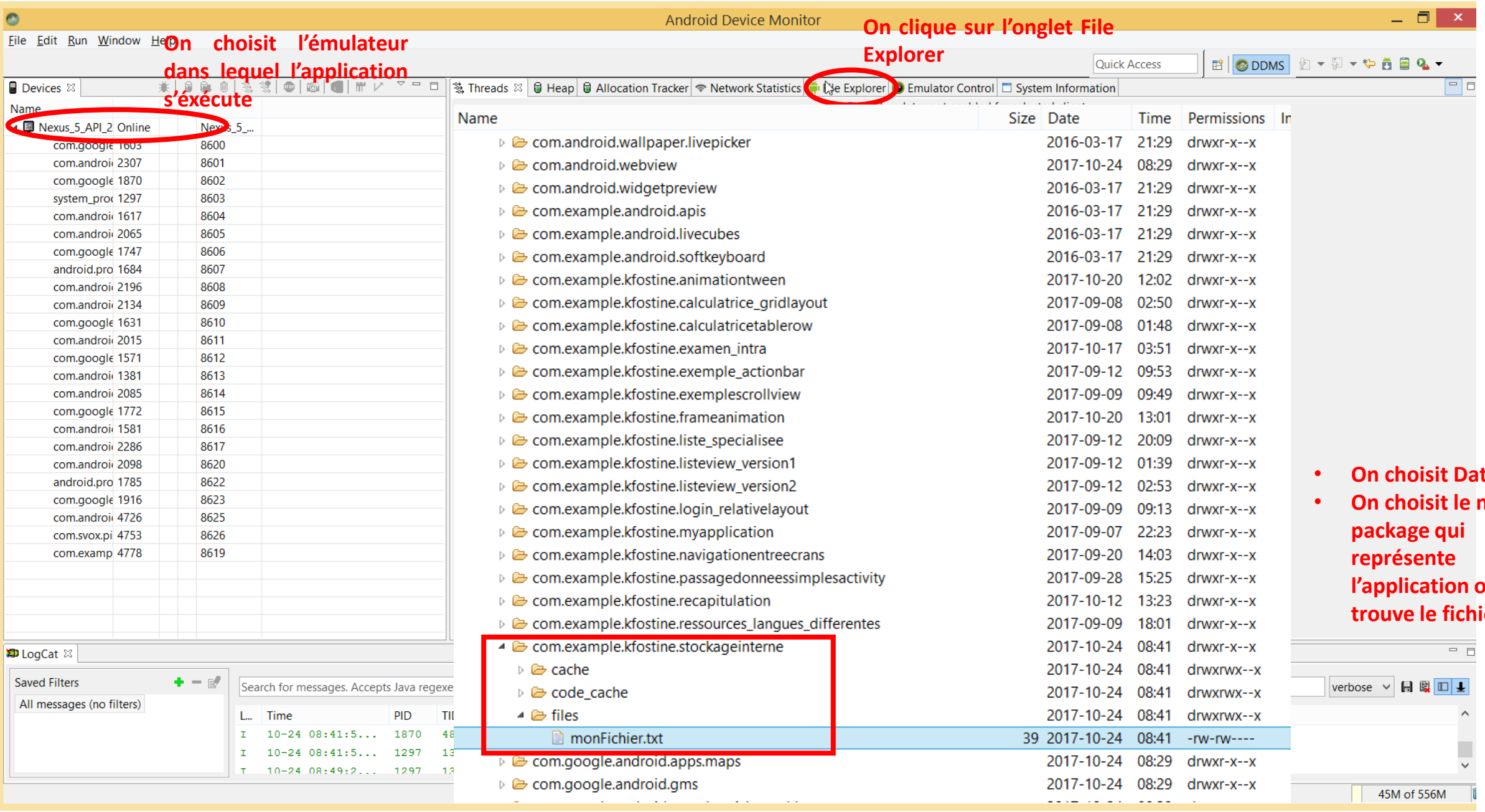
Solution Exercice 2 – Aller chercher le fichier dans l'émulateur

- Cliquer sur la commande Android Device Monitor dans la barre d'outils de Android Studio :



- Qui ouvre la fenêtre de la diapositive suivante :

Android Device Monitor



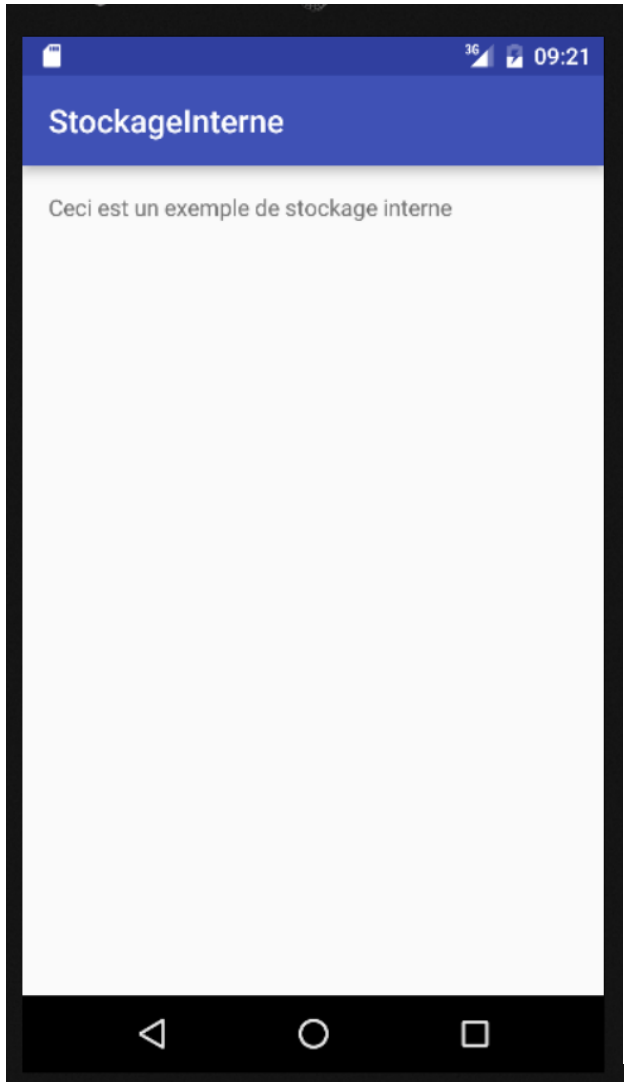
- On choisit Data/Data
- On choisit le nom du package qui représente l'application ou se trouve le fichier

Exercice 3

- Écrire un programme Android qui crée un objet Employe (matricule, nom, prenom, poste, salaire) et qui affiche une activité avec des EditText(nom, prenom, salaire) et des boutons radios pour le poste(Gérant, vendeur, caissier) et un bouton enregistrer
- Quand on clique sur le bouton Enregistrer, le programme ajoute l'employé dans le fichier employes.txt (qui ont les informations des employés sous format csv)
  - Matricule;nom;prenom;poste;salaire
- L'application doit réinitialiser les champs de saisie après avoir enregistrer un employé pour permettre à l'utilisateur de saisir un autre employé
- Le fichier doit être ouvert en mode append, de manière à ce que plusieurs employés puissent être ajoutés dans le fichier

Exercice 4 – Lecture dans un fichier interne

- Écrire un programme qui lit le fichier « monFichier.txt » créé précédemment et qui l'affiche dans un TextView




```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    String fileName = "monFichier.txt";
    TextView textView = (TextView) findViewById(R.id.contenuFichier);
    FileInputStream fis;
    try{
        fis = openFileInput(fileName) ;
        byte[] buffer = new byte[2024];
        StringBuilder content = new StringBuilder();
        while(fis.read(buffer)!= -1){
            content.append(new String(buffer));
        }
        textView.setText(content);
        Toast.makeText(this, "Lecture de fichier complétée", Toast.LENGTH_LONG).show();
        fis.close();
    }
    catch(FileNotFoundException ex){
        ex.printStackTrace();
    }
    catch(IOException ex){
        ex.printStackTrace();
    }
}
```

Exercice 5

- Reproduire avec Android les Ecrans de l’image exercice5.jpg
- À enregistrer les informations sous format CSV dans un fichier interne
- À récupérer les informations et les afficher dans la liste

Utilisation de fichiers de cache

- On peut aussi stocker des données en cache sans pour pour autant les sauvegarder dans des fichiers persistants.
- On peut utiliser la méthode getChacheDir() pour ouvir un dossier de cache getCacheDir();
- Présent dans la mémoire du téléphone, ce dossier représente l'emplacement dédié à la sauvegarde des fichiers caches de l'application
- **Si un appareil rencontre des problèmes d'espaces, Android pourra supprimer ces fichiers pour récupérer de l'espace**
- **Les fichiers de cache doivent prendre une place limitée dans la mémoire du téléphone (1 Mo maximum par exemple)**
- **Lors de la désinstallation de l'application, le cache est supprimé**

Exercice 6

- Refaire l'exercice 5 en sauvegardant les données en cache. Utiliser Android Device Monitor pour récupérer et visualiser le contenu du fichier
  - Vous pouvez vous baser sur cet exemple

```
String content = "hello world";
File file;
FileOutputStream outputStream;
try {
    // file = File.createTempFile("MyCache.txt", null, getCacheDir());
    file = new File(getCacheDir(), "MyCache.txt");

    outputStream = new FileOutputStream(file);
    outputStream.write(content.getBytes());
    outputStream.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Stockage externe

- Pour éviter les problèmes de mémoire disponible, on peut stocker les fichiers d'une application sur la mémoire externe de l'appareil
- Il existe deux types de mémoire externe
  - Stockage externe démontable (Carte SD par exemple)
  - Stockage externe non démontable (présent sur la mémoire physique du téléphone)
- Les fichiers créés dans le stockage externe sont disponibles en lecture pour l'utilisateur et les autres applications
- Les utilisateurs peuvent supprimer ces fichiers sans désinstaller l'application

Stockage externe

- Les utilisateurs peuvent utiliser sa mémoire externe pour stocker ses musiques, photos et vidéos.
- Cette mémoire est susceptible d'être non disponible(lecture/écriture) lors de l'utilisation d'une application
- Il est nécessaire de vérifier la disponibilité du stockage externe pour n'importe quelle requête en lecture ou en écriture
- La méthode getExternalStorageState permet de vérifier cette disponibilité

Vérifier l'état du stockage externe

```
final String storageState = Environment.getExternalStorageState();
if(storageState.equals(Environment.MEDIA_MOUNTED)){
    //On peut lire et écrire sur le stockage externe
}
else if(storageState.equals(Environment.MEDIA_MOUNTED_READ_ONLY)){
    //On peut lire seulement lire dans le stockage externe
}
else if(storageState.equals(Environment.MEDIA_REMOVED)){
    //Le stockage externe n'est pas disponible
}
```

Les différents états possibles

- MEDIA_BAD_REMOVAL : stockage débranché avant d'être démonté
- MEDIA_CHECKING : stockage en cours de vérification
- MEDIA_MOUNTED : stockage disponible en lecture/écriture
- MEDIA_MOUNTED_READ_ONLY : stockage disponible en lecture seule
- MEDIA_NOFS : stockage disponible mais utilise un formatage non pris en charge
- MEDIA_REMOVED : stockage non présent
- MEDIA_SHARED : stockage utisé en USB (Connecté au PC)
- MEDIA_UNMOUNTABLE : stockage présent mais ne peut pas être monté
- MEDIA_UNMOUNTED : stockage présent mais pas monté

Permission requise

- L'écriture sur le stockage externe nécessite la permission WRITE_EXTERNAL_STORAGE dans le fichier manifest

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

- La lecture du stockage interne nécessite la permission

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

Depuis la version Jelly Bean, le stockage externe en lecture nécessite la permission READ_EXTERNAL_STORAGE

Exercice 7

- Refaire l'exercice 5 en sauvegardant les données en stockage externe. Utiliser Android Device Monitor pour récupérer et visualiser le contenu du fichier
  - Vous pouvez vous baser sur cet exemple tiré du site : http://vogella.developpez.com/tutoriels/android/persitance-preferences-fichiers/

```
private void readFileFromSDCard() {
  File directory = Environment.getExternalStorageDirectory();
  // suppose qu'un fichier article.rss soit disponible sur la carte SD
  File file = new File(directory + "/article.rss");
  if (!file.exists()) {
    throw new RuntimeException("File not found");
  }
  Log.e("Testing", "Starting to read");
  BufferedReader reader = null;
  try {
    reader = new BufferedReader(new FileReader(file));
    StringBuilder builder = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
      builder.append(line);
    }
  } catch (Exception e) {
    e.printStackTrace();
  } finally {
    if (reader != null) {
      try {
        reader.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

À vérifier l'état du stockage externe avant d'y enregistrer des données

Éxécuter une action

- Une application Android peut vous proposer une liste d'application en fonction du type d'action à exécuter
- Exemple
  - On aimerait ouvrir un fichier PDF avec une application installée sur l'appareil
  - On ne connait pas le nom de l'application, ni l'identifiant du package
  - Pour cela, il faut utiliser une autre spécificité des intents (**Une action**)

```
File file = new File("/sdcard/fichier.pdf");
Uri path = Uri.fromFile(file);
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setDataAndType(path, "application/pdf");
PackageManager  pm = getPackageManager();
ComponentName component = intent.resolveActivity(pm);

if(component == null){
    Toast toast = Toast.makeText(MainActivity.this,
            "Aucune application disponible pour ouvrir un fichier pdf", Toast.LENGTH_LONG);
    toast.show();
}
else{
    startActivity(intent);
}
```

Exécuter une action

- **ACTION_VIEW** : permet d'afficher des données à l'utilisateur afin qu'il puisse faire une sélection
- On a spécifié le type de données ciblées et le fichier à ouvrir avec **setDataAndType**
- Pour savoir si au moins une application pouvant répondre à l'intent est disponible sur l'appareil, on utilise **resolveActivity**

Exercice 8

- Placez un fichier pdf dans votre stockage externe et trouvez une application disponible sur le téléphone pour le lire. Inspirez-vous du code de la diapo 26.

Accéder aux fichiers d'une application

- Vous pouvez accéder aux fichiers sauvegardés par votre application par le stockage externe de l'appareil
- Ces fichiers sont désinstallés lors de la désinstallation de l'application
- Il faut utiliser la méthode getExternalFileDir(String type) qui permet d'obtenir une instance de la classe File portant sur le répertoire (non visible de l'utilisateur comme des médias)
  - **File file = Environment.getExternalFileDir(Environment.DIRECTORY_DCIM);**
- Les types possibles :
- DIRECTORY_ALARMS: emplacement des sons pouvant être utilisés comme alarme
- DIRECTORY_DCIM : emplacement des images et photos
- DIRECTORY_DOWNLOADS : emplacement des fichiers téléchargés

Les types possibles (Suite)

- DIRECTORY_MOVIES : emplacement des films, séries et autres vidéos
- DIRECTORY_MUSIC : emplacement des musiques
- DIRECTORY_NOTIFICATION : emplacement des sons utilisés pour les notifications
- DIRECTORY_PICTURE : emplacement des images accessibles par l'utilisateur
- DIRECTORY_PODCAST : emplacement des fichiers audio représentant des podcasts
- DIRECTORY_RINGTONES : emplacement des sons utilisés comme sonneries
- Null : retourne le dossier racine de l'application

Exercice 9

- Écrire une application qui utilise la caméra pour prendre des photos et de sauvegarder les photos dans le bon dossier de la mémoire externe.
- L'application doit afficher dans un ListView toutes les photos sauvegardés dans ce dossier.
- Voir les directives dans l'exemple de Multimedia.pdf

Exercice 10

- Écrire une application qui utilise la caméra pour prendre des vidéos et de sauvegarder ces vidéos dans le bon dossier de la mémoire externe.
- L’application doit afficher dans un ListView toutes les vidéos sauvegardés dans ce dossier.
- Voir les directives dans l’exemple de Multimedia.pdf

Accéder aux fichiers partagés

- Les fichiers partagés sont sauvegardés sur un répertoire accessible par l'utilisateur et toutes les autres applications
- Ces fichiers ne seront pas supprimés lors de la désinstallation de l'application
- Il faut utiliser la méthode getExternalStoragePublicDirectory(String type)
- Le type peut avoir les mêmes valeurs que ceux présentés pour la méthode getExternalFileDir
  - **File sharedFile = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DCIM);**

Exercice 11

- Refaire les exercices 9 et 10 en utilisant des fichiers partagés utilisant le stockage externe.

Stockage en base de données

- Chaque appareil Android embarque système de Gestion de Base de Données SQLite qu'on peut utiliser au sein d'une application afin de stocker différents types de données
- SQLite est un SGBD relationnel et open source qui ne nécessite qu'une petite quantité de mémoire à l'exécution.
- La syntaxe du SQLite ressemble à celle du SQL et les types utilisés sont les mêmes(INTEGER, TEXT, …)
- Une base de données SQLite est privée et ne peut être accessible directement que par l'application l'ayant créée.
  - Elle se trouve à un endroit spécifique de l'appareil :
    - *data/data/package_de_lapplication/databases*

Création d’une base de données

- Lors de la création d’une base de données, on aura besoin de plusieurs briques :
  - SQLiteOpenHelper : utilisé pour faciliter la création et la gestion des différentes versions d’une base de données
  - SQLiteDatabase : sert à exposer les méthodes nécessaires à l’interaction avec une base de données (création, suppression, mise à jour, …)
  - Cursor : représente le résultat d’une requête qui peut contenir 0 ou plusieurs éléments

Exercice 12

- Créer une base de données contenant les différents chapitres de livre avec une description de chaque chapitre et qui affiche dans un listView les chapitres insérés dans la base de données.
  - Id (auto_incrément, géré par la BD)
  - Nom
  - Description

Solution Exercice 12 – Classe Chapitre

```java
public class Chapitre {
    private int id;
    private String name;
    private String description;

    public Chapitre(){}


    public Chapitre(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Nom du chapitre = " + name + "\n"
            + "Description du chapitre = " + description);
        return sb.toString();
    }
}
```

Solution Exercice 12 – classe ChapitreBaseSQLite

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;

/**
 * Created by kfostine on 2017-11-03.
 */
public class ChapitreBaseSQLite extends SQLiteOpenHelper {
    private static final String TABLE_CHAPITRES = "table_chapitres";
    private static final String COL_ID = "ID";
    private static final String COL_NAME = "NAME";
    private static final String COL_DESCRIPTION = "DESCRIPTION";

    private static final String CREATE_DB = "CREATE TABLE " +
            TABLE_CHAPITRES + "(" + COL_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COL_NAME + " TEXT NOT NULL, " + COL_DESCRIPTION + " TEXT NOT NULL);";

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

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

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

}
```

Voir les explications à la diapositive suivante

Explications de la classe ChapitreBaseSQLite

- Cette classe étend SQLiteOpenHelper, elle servira à créer et gérer les différentes versions de la base de données
- Elle dispose de plusieurs attributs
  - Le nom de la table
  - Les identifiants des 3 colonnes de la tables
  - La requête SQL de création de la table
- Elle dispose de 3 méthodes
  - Un constructeur : permet de créer un objet facilitant la génération et la gestion d'une BD. Elle utilise 4 arguments :
    - Le contexte utilisé pour créer la base de données
    - Le nom du fichier représentant la base de données
    - CursorFactory : utilisé pour surcharger la classe gérant la création des curseurs de la base de données. On peut passer la valeur null pour utiliser la fabrique par défaut
    - La version de la base de données
  - onCreate : Permet de créer la base de données à l'aide de la requête stockée grâce à la méthode execSQL
  - onUpgrade : Cette méthode est utilisée lors de la montée en version de la base de données.

Il faut ensuite créer la classe qui servira à gérer les données(insertion, récupération, suppression, … de la table)

Solution Exercice 12 – Classe ChapitreBD

```
package com.example.kfostine.exemplebdchapitre;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

import java.util.ArrayList;

/**
 * Created by kfostine on 2017-11-03.
 */
public class ChapitreBD {
    private static final int VERSION = 1;
    private static final String NOM_BD = "Chapitre.db";
    private static final String TABLE_CHAPITRES = "table_chapitres";
    private static final String COL_ID = "ID";
    private static final int  NUM_COL_ID = 0;
    private static final String COL_NAME = "NAME";
    private static final int  NUM_COL_NAME = 1;
    private static final String COL_DESCRIPTION = "DESCRIPTION";
    private static final int  NUM_COL_DESCRIPTION = 2;

    private SQLiteDatabase db;
    private ChapitreBaseSQLite chapitres;

    public ChapitreBD(Context context){
        chapitres = new ChapitreBaseSQLite(context, NOM_BD, null, VERSION);
    }
```

Solution Exercice 12 – Classe ChapitreBD

```
public void openForWrite(){
    db = chapitres.getWritableDatabase();
}

public void openForRead(){
    db = chapitres.getReadableDatabase();
}

public void close(){
    db.close();
}

public SQLiteDatabase getDb(){
   return db;
}

public long insertChapter(Chapitre chapitre){
    ContentValues content = new ContentValues();
    content.put(COL_NAME, chapitre.getName());
    content.put(COL_DESCRIPTION, chapitre.getDescription());
    return db.insert(TABLE_CHAPITRES, null, content);
}

public int updateChapter(int id, Chapitre chapitre){
    ContentValues content = new ContentValues();
    content.put(COL_NAME, chapitre.getName());
    content.put(COL_DESCRIPTION, chapitre.getDescription());
    return db.update(TABLE_CHAPITRES, content, COL_ID + " = " + id, null);
}
```

Solution Exercice 12 – Classe ChapitreBD

```
public int removeChapter(String name){
    return db.delete(TABLE_CHAPITRES, COL_NAME + " = " + name, null);
}

public Chapitre getChapter(String name){
    Cursor c = db.query(TABLE_CHAPITRES, new String[]{COL_ID, COL_NAME, COL_DESCRIPTION},
            COL_NAME + " like \"" + name + "\"", null, null, null, COL_NAME);
    return cursorToChapter(c);
}

public Chapitre cursorToChapter(Cursor c){
    if(c.getCount() == 0){
        c.close();
        return null;
    }
    Chapitre chapter = new Chapitre();
    chapter.setId(c.getInt(NUM_COL_ID));
    chapter.setName(c.getString(NUM_COL_NAME));
    chapter.setDescription(c.getString(NUM_COL_DESCRIPTION));
    c.close();
    return chapter;
}
```

Solution Exercice 12 – Classe ChapitreBD

```
    public ArrayList<Chapitre> getAllChapters(){
        Cursor c = db.query(TABLE_CHAPITRES, new String[]{COL_ID, COL_NAME, COL_DESCRIPTION},
                null, null, null, null, COL_NAME);
        if(c.getCount() == 0){
            c.close();
            return null;
        }
        ArrayList<Chapitre> chapterList = new ArrayList<Chapitre>();
        while(c.moveToNext()){
            Chapitre chapter = new Chapitre();
            chapter.setId(c.getInt(NUM_COL_ID));
            chapter.setName(c.getString(NUM_COL_NAME));
            chapter.setDescription(c.getString(NUM_COL_DESCRIPTION));
            chapterList.add(chapter);
        }
        c.close();
        return chapterList;
    }
}
```

Solution Exercice 12 – layout : activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.kfostine.exemplebdchapitre.MainActivity">

    <ListView
        android:id="@+id/list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         />
</RelativeLayout>
```

Solution Exercice 12 – Classe MainActivity

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView list = (ListView) findViewById(R.id.list);
        Chapitre chapitre1 = new Chapitre("La plateforme Android",
                "Présentation et historique de la plateforme Android");
        Chapitre chapitre2 = new Chapitre("Environment de développement",
                "Présentation et Installation de l'environnement de développement Android");
        Chapitre chapitre3 = new Chapitre("Les interfaces simples",
                "Présentation des interfaces simples");

        ChapitreBD chapitreBd = new ChapitreBD(this);
        chapitreBd.openForWrite();
        chapitreBd.insertChapter(chapitre1);
        chapitreBd.insertChapter(chapitre2);
        chapitreBd.insertChapter(chapitre3);

        ArrayList<Chapitre> chapterList = chapitreBd.getAllChapters();
        chapitreBd.close();

        ArrayAdapter<Chapitre> adapter = new ArrayAdapter<Chapitre>(this,
                android.R.layout.simple_list_item_1, chapterList);
        list.setAdapter(adapter);

    }
}
```

Exercice 13

- Refaire l'exercice 12 en utilisant 4 boutons sur l'activité principale pour :
  - Nouveau Chapitre : Interface qui permet à l'utilisateur de saisir des chapitres et de les insérer dans la BD
  - Liste des chapitres : Interface qui affiche les chapitres de la BD dans un ListView
  - Modification d'un chapitre : Interface qui permet à un utilisateur de modifier un chapitre par son ID et de retourner à l'écran principal
  - Suppression d'un chapitre : Interface qui permet à un utilisateur de supprimer un chapitre par son nom et de retourner à l'écran principal

Références

- https://developer.android.com/training/basics/data-storage/files.html
- http://vogella.developpez.com/tutoriels/android/persitance-preferences-fichiers/