Université d'Oran 1
Faculté des Sciences Exactes et Appliquées
Département d'Informatique

3ème année Licence – Semestre 2 **Année : 2019/2020**
Module : Applications Mobiles

Fiche TD/TP n°6 : Stockage des données

Objectif :

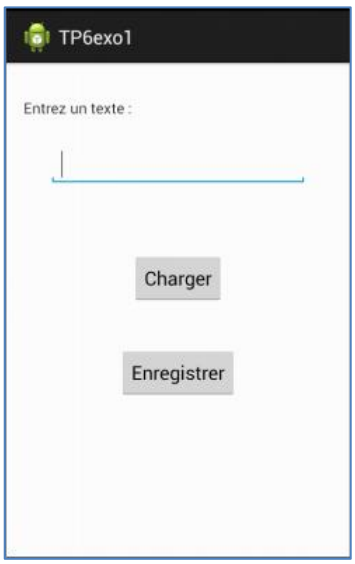
- Le but de ce TD/TP est de se familiariser avec le stockage des données sous Android (SQLite, Stockage interne, stockage externe, etc.)

Prérequis : Du cours 5 au cours 10.

Exercice 1 :

I. Enregistrement sur un stockage externe (carte SD) :

- On veut afficher un texte que nous avons saisi et enregistré dans le Smartphone.



Pour cela, vous devez utiliser la classe **FileOutputStream**. La méthode **openFileOutput ()** ouvre un fichier nommé pour l'écriture, avec le mode spécifié. Dans cet exercice, vous utilisez la constante **MODE_PRIVATE** pour indiquer que le fichier est lisible par toutes les autres applications:

```
FileOutputStream fOut = openFileOutput("fichiertexte.txt", MODE_PRIVATE);
```

Pour convertir un flux de caractères en un flux d'octets, vous utilisez une instance de la classe **OutputStreamWriter**, en lui passant une instance de l'objet **FileOutputStream**:

```
OutputStreamWriter osw = new OutputStreamWriter(fOut);
```

Vous utilisez ensuite sa méthode **write ()** pour écrire la chaîne dans le fichier. Pour vous assurer que tous les octets sont écrits dans le fichier, utilisez la méthode **flush ()**. Enfin, utilisez la méthode **close ()** pour fermer le fichier:

```
osw.write(str);
osw.flush();
osw.close();
```

Pour lire le contenu d'un fichier, vous utilisez la classe **FileInputStream**, avec la classe **InputStreamReader**:

```
FileInputStream fIn = openFileInput("fichiertexte.txt");
InputStreamReader isr = new InputStreamReader(fIn);
```

Puisque vous ne connaissez pas la taille du fichier à lire, le contenu est lu par blocs de 100 caractères dans un tampon (tableau de caractères). Les caractères lus sont ensuite copiés dans un objet String:

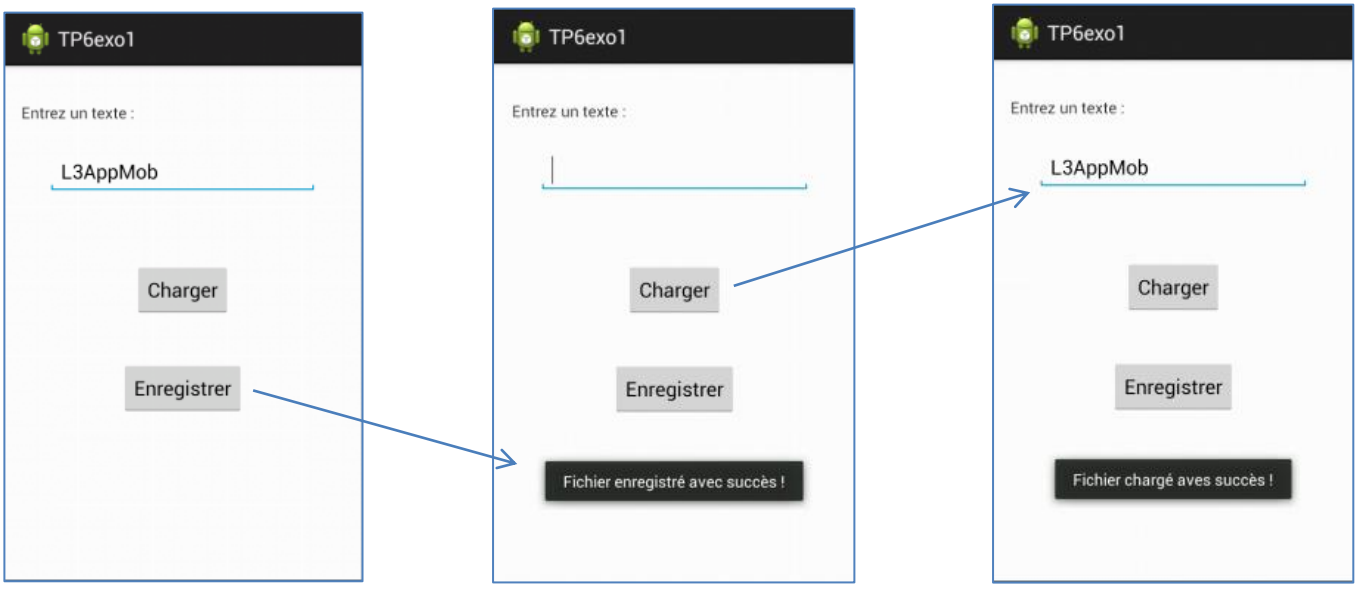
```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer)) > 0) {

//---convertir des caractères en un String---
String readString = String.copyValueOf(inputBuffer, 0,charRead);
s += readString;
inputBuffer = new char[READ_BLOCK_SIZE];
}
```

La méthode **read ()** de l'objet **InputStreamReader** vérifie le nombre de caractères lus et renvoie -1 si la fin du fichier est atteinte.

- Exécutez l'application, tapez du texte, puis cliquez sur le bouton Enregistrer.

- Si le fichier est enregistré avec succès, vous voyez la classe Toast affichant le "Fichier enregistré avec succès!" message.

- Cliquez sur le bouton Charger et vous devriez voir la chaîne apparaître à nouveau dans la vue EditText. Cela confirme que le texte est enregistré correctement.



II. Enregistrement sur un stockage externe (carte SD) :

Dans la section précédente, vous avez enregistré votre fichier sur le stockage interne de votre appareil Android. Parfois, il serait utile de les enregistrer sur un stockage externe (comme une carte SD) en raison de sa plus grande capacité, ainsi que de la capacité de partager facilement les fichiers avec d'autres utilisateurs (en retirant la carte SD et en la passant à quelqu'un d'autre).

Dans cet exercice, vous allez enregistrer des fichiers sur un stockage externe:

- En utilisant le projet créé dans la section précédente comme exemple, modifiez la méthode **onClick ()** du bouton **Enregistrer** en ajoutant ces lignes pour le stockage externe:

```
//--- Stockage SD Card ---

File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +"/MesFichiers");
directory.mkdirs();
File fichier = new File(directory, "fichiertexte.txt");
FileOutputStream fOut = new FileOutputStream(fichier);
```

- Le code précédent utilise la méthode **getExternalStorageDirectory ()** pour renvoyer le chemin d'accès complet au stockage externe. En règle générale, il doit renvoyer le chemin «**/sdcard**» pour un appareil réel et «**/mnt/sdcard**» pour un émulateur Android. Cependant, vous ne devriez jamais essayer de coder en dur le chemin vers la carte SD, car les fabricants peuvent choisir d'attribuer un nom de chemin différent à la carte SD. Veillez à utiliser la méthode **getExternalStorageDirectory ()** pour renvoyer le chemin d'accès complet à la carte SD. Vous créez ensuite un répertoire appelé **MesFichiers** sur la carte SD. Enfin, vous enregistrez le fichier dans ce répertoire.

- Pour charger le fichier à partir du stockage externe, modifiez la méthode **onClick ()** pour le bouton Charger:

```
File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +"/MesFichiers");
File fichier = new File(directory, "fichiertexte.txt");
FileInputStream fIn = new FileInputStream(fichier);
InputStreamReader isr = new InputStreamReader(fIn);
```

- Notez que pour écrire et lire sur le stockage externe, vous devez ajouter les permissions suivantes dans votre fichier **AndroidManifest.xml**:

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

- Exécutez l'application

Exercice 2 :

- Dans cet exercice, vous allez créer une **base de données SQLite** nommée **MaBDD** contenant une table nommée **contacts**. Cette table comporte trois colonnes: **_id**, **nom** et **e-mail**.
- Créez un nouveau projet Android et nommez-le **TP6exo2**.
- Ajoutez un nouvelle classe Java au package et nommez-la **DBAdapter.**
- Vous commencez par définir plusieurs constantes pour contenir les différents champs de la table que vous allez créer dans votre base de données:

```
static final String KEY_ROWID = "_id";
static final String KEY_NAME = "name";
static final String KEY_EMAIL = "email";
static final String TAG = "DBAdapter";
static final String DATABASE_NAME = "MaBDD";
static final String DATABASE_TABLE = "contacts";
static final int DATABASE_VERSION = 1;
static final String DATABASE_CREATE ="create table contacts (" +
        "_id integer primary key autoincrement, " +
        "name text not null, email text not null);";

final Context context;
DatabaseHelper DBHelper;
SQLiteDatabase db;

public DBAdapter(Context ctx){
    this.context = ctx;
    DBHelper = new DatabaseHelper(context);
}
```

- En particulier, la constante **DATABASE_CREATE** contient l'instruction SQL pour créer la table de contacts dans la base de données **MaBDD**.
- Pour créer une base de données dans votre application à l'aide de la classe **DBAdapter**, vous créez une instance de la classe **DBAdapter**. Le constructeur de la classe **DBAdapter** créera ensuite une instance de la classe **DatabaseHelper** pour créer une nouvelle base de données.
- Dans la classe **DBAdapter**, vous ajoutez également une classe privée qui hérite de la classe **SQLiteOpenHelper**.
- **SQLiteOpenHelper** est une classe d'assistance dans Android pour gérer la création de bases de données et la gestion des versions. En particulier, vous devez remplacer les méthodes **onCreate ()** et **onUpgrade ()**:

```
private static class DatabaseHelper extends SQLiteOpenHelper{

    DatabaseHelper(Context context)
    {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

@Override
public void onCreate(SQLiteDatabase db)
{
try {
    db.execSQL(DATABASE_CREATE);
    } catch (SQLException e) {
        e.printStackTrace();
            }
    }

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){

    Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");
    db.execSQL("DROP TABLE IF EXISTS contacts");
    onCreate(db);
    }
}
```

- La méthode **onCreate ()** crée une nouvelle base de données si la base de données requise n'est pas présente. La méthode **onUpgrade ()** est appelée lorsque la base de données doit être mise à niveau. Ceci est obtenu en vérifiant la valeur définie dans la constante **DATABASE_VERSION**. Pour cette implémentation de la méthode **onUpgrade ()**, il vous suffit de supprimer la table et de la recréer.

- Vous pouvez ensuite définir les différentes méthodes d'ouverture et de fermeture de la base de données, ainsi que les méthodes d'ajout / modification / suppression de lignes dans le tableau:

```
//--- Ouvrir la BDD ---
    public DBAdapter open() throws SQLException{

        db = DBHelper.getWritableDatabase();
        return this;
    }

//--- Fermer la BDD ---
    public void close(){
        DBHelper.close();
    }

//--- inserer un contact dans la BDD ---
    public long insertContact(String name, String email){
        ContentValues initialValues = new ContentValues();
        initialValues.put(KEY_NAME, name);
        initialValues.put(KEY_EMAIL, email);
        return db.insert(DATABASE_TABLE, null, initialValues);
    }

//--- Supprimer un contact ---
    public boolean deleteContact(long rowId){
        return db.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
    }

//--- Récupérer tous les contacts ---
    public Cursor getAllContacts(){
        return db.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
        KEY_EMAIL}, null, null, null, null, null);
    }
```

- Ajoutez deux autres méthodes qui permettent de récupérer et mettre à jour un contact en particulier :

```
//--- Récupérer un contact en particulier ---
    public Cursor getContact(long rowId) throws SQLException{
        Cursor mCursor = db.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
                KEY_NAME, KEY_EMAIL},
                KEY_ROWID + "=" + rowId, null,
                null, null, null, null);
        if (mCursor != null) {
            mCursor.moveToFirst();
        }
        return mCursor;
    }

//--- Mettre à jour un contact ---
    public boolean updateContact(long rowId, String name, String email){
        ContentValues args = new ContentValues();
        args.put(KEY_NAME, name);
        args.put(KEY_EMAIL, email);
        return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
    }
```

- Notez qu'Android utilise la classe **Cursor** comme valeur de retour pour les requêtes. Considérez le curseur comme un pointeur vers le jeu de résultats d'une requête de base de données. L'utilisation du curseur permet à Android de gérer plus efficacement les lignes et les colonnes selon les besoins.

- Vous utilisez un objet **ContentValues** pour stocker des paires (clé /valeur). Sa méthode **put ()** vous permet d'insérer des clés avec des valeurs de différents types de données.

- Une fois la classe d'assistance **DBAdapter** créée, vous êtes maintenant prêt à utiliser la base de données. Dans ce qui suit, vous allez effectuer des opérations CRUD régulières (création, lecture, mise à jour et suppression) généralement associées aux bases de données.

- En utilisant le même projet créé précédemment, ajoutez les instructions suivantes au fichier **MainActivity.java**, ces instructions vont permettre d'insérer un contact et aussi récupérer et afficher tous les contacts :

```
    DBAdapter db = new DBAdapter(this);

  //--- Ajouter des contacts ---
    db.open();
    db.insertContact("Rafik Merad", "rafik.merad@gmail.com");
    db.insertContact("Mohamed Sayah", "sayahmh@gmail.com");
    db.close();

 //--- Récupération de tous les contacts d'une table ---

    db.open();
    Cursor c = db.getAllContacts();
    if (c.moveToFirst()){
        do {DisplayContact(c);}
        while (c.moveToNext());
            }
    db.close();

}

public void DisplayContact(Cursor c){

    Toast.makeText(this,"id: " + c.getString(0) + "\n" +
            "Name: " + c.getString(1) + "\n" +
            "Email: " + c.getString(2),
            Toast.LENGTH_LONG).show();
            }
```

- Exécutez l'application. La figure suivante montre la classe Toast affichant les contacts extraits de la base de données.

Remarque :

La méthode **getAllContacts ()** de la classe **DBAdapter** récupère tous les contacts stockés dans la base de données. Le résultat est renvoyé sous la forme d'un objet Cursor. Pour afficher tous les contacts, vous devez d'abord appeler la méthode **moveToFirst ()** de l'objet **Cursor**. S'il réussit (ce qui signifie qu'au moins une ligne est disponible), vous affichez les détails du contact à l'aide de la méthode **DisplayContact ().** Pour passer au contact suivant, appelez la méthode **moveToNext ()** de l'objet **Cursor**.

- De la même manière essayez d'ajouter le code qui permet

- De récupérer un seul contact :

Indication : Pour récupérer un seul contact via son ID, appelez la méthode **getContact ()** de la classe **DBAdapter**

- De mettre à jour un contact :

Indication : Pour mettre à jour un contact particulier, appelez la méthode **updateContact ()** de la classe **DBAdapter** en transmettant l'ID du contact que vous souhaitez mettre à jour.

- De supprimer un contact :

Indication : Pour supprimer un contact, utilisez la méthode **deleteContact ()** de la classe **DBAdapter** en transmettant l'ID du contact que vous souhaitez supprimer.