

On veut, dans ce TP, construire une application Android qui permet de gérer les absences des étudiants. La liste des étudiants sera mise dans une base de données Android gérée par SQLite. La table Etudiant est définie par :

Etudiant (Id(Integer), CNE(Text) , Nom(Text) , Prenom(Text))

La table Date d'absence :

Date_d_Absence(Id(Integer), IdEtudiant(Integer), date(DATETIME))

- **Construction de la base de données**

Avant d'aller plus loin, vous devez écrire votre classe Etudiant avec toutes les méthodes getter et setter pour maintenir les informations sur un étudiant comme un objet de type Etudiant.

- a. Ecrire la classe Etudiant contenant les informations :

```
public class Etudiant implements Serializable {  
    private int id;  
    private String nom;  
    private String prenom;  
    private String CNE;  
  
}
```

Compléter cette classe avec des méthodes getter et setter et constructeurs appropriés (utiliser Eclipse pour générer ces méthodes).

Vous devez maintenant écrire votre propre classe pour gérer tous les opérations de la base de données comme l'ouverture de connexion, fermeture de connexion, insertion, mise à jour, lecture, suppression et d'autres choses. Cette classe doit étendre la classe SQLiteOpenHelper fournit dans l'API android.

- b. Ecrire une classe "DatabaseHelper" permettant de gérer une base de données android dont le début est :

```
public class DatabaseHandler extends SQLiteOpenHelper {  
    // Database Version  
    private static final int DATABASE_VERSION = 1;  
    // Database Name  
    private static final String DATABASE_NAME = " base-absences";  
    // Table ETUDIANTS  
    private static final String TABLE_ETUDIANTS = "ETUDIANTS";  
    private static final String KEY_ID = "id";  
    private static final String KEY_LAST_NAME = "nom";  
    private static final String KEY_FIRST_NAME = "prenom";  
    // Table DATE_D_ABSENCE  
    private static final String TABLE_DATE_ABSENCE = "DATE_D_ABSENCE";  
    private static final String KEY_ID_ABS = "idAbs";  
    private static final String KEY_ID_ETUDIANT = "idEtudiant";  
    private static final String KEY_DATE = "date";  
  
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd", Locale.getDefault());  
  
    public DatabaseHandler(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
}
```

L'attribut dateFormat permet de formater la date sous la "yyyy-MM-dd".

Après l'extension de la classe de SQLiteOpenHelper Eclipse va vous demander de définir deux méthodes onCreate () et onUpgrade ().

onCreate () – C'est là où vous allez d'écrire les instructions CREATE TABLE. Cette méthode va être invoquée lorsque la base de données est créée.

onUpgrade () - Cette méthode est appelée lorsque la base de données est mis à jour, comme par exemple la modification de la structure de la table, l'ajout de contraintes à la base de données, etc. Donc il suffit de supprimer les tables et les recréer à nouveau.

- c. Déclarer deux strings représentant les instructions SQLite permettant la création des deux tables Etudiant et date d'absence:

```
String CREATE_TABLE_ETUDIANT = "CREATE TABLE IF NOT EXISTS " + TABLE_ETUDIANTS
    + "("
    + KEY_ID + " INTEGER PRIMARY KEY," + KEY_LAST_NAME + " TEXT,"
    + KEY_FIRST_NAME + " TEXT" + ")";

String CREATE_TABLE_ABSENCES = "CREATE TABLE IF NOT EXISTS " + TABLE_DATE_ABSENCE
    + "("
    + KEY_ID_ABS + " INTEGER PRIMARY KEY,"
    + KEY_ID_ETUDIANT + " INTEGER,"
    + KEY_DATE + " DATETIME DEFAULT CURRENT_DATE"
    + " FOREIGN KEY (" + KEY_ID_ETUDIANT + ") REFERENCES "
    + TABLE_ETUDIANTS + " (" + KEY_ID + ")"
    + ")";
```

- d. Exécuter ces deux instructions lors de l'appel à onCreate():

```
Override
public void onCreate(SQLiteDatabase db) {
    // creating required tables
    db.execSQL(CREATE_TABLE_ETUDIANT);
    db.execSQL(CREATE_TABLE_ABSENCES);
}
```

- e. Définir le corps de la méthode onUpgrade()

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Drop the older tables if existed
    db.execSQL("DROP TABLE IF EXISTS " + CREATE_TABLE_ETUDIANT);
    db.execSQL("DROP TABLE IF EXISTS " + CREATE_TABLE_ABSENCES);
    // Create tables again
    onCreate(db);
}
```

Maintenant, vous devez écrire des méthodes pour manipuler la base de données tous les opérations de lecture et d'écriture.

- f. Mettez en œuvre les méthodes suivantes pour les deux tables :

```
// Adding new etudiant
public void addEtudiant(Etudiant etudiant) {}
// Getting single etudiant
public Etudiant getEtudiant(int id) {}
// Getting All Etudiants
public List<Etudiant> getAllEtudiants() {}
// Updating single etudiant
public int updateEtudiant(Etudiant etudiant) {}
// Deleting single etudiant
public void deleteEtudiant(Etudiant etudiant) {}
```

```

// mark absent
public void markeAbsent(Etudiant etudiant) {}
//unmark absent
public void unmarkeAbsent(Etudiant etudiant, Date date) {}
//count absences for a student
public int getAbsenceCount(Etudiant etudiant){}
public List<Date> getAbsenceDates(Etudiant etudiant){}

```

- La méthode d'ajout :

```

// Adding new etudiant
public void addEtudiant(Etudiant etudiant) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_LAST_NAME, etudiant.getNom());
    values.put(KEY_FIRST_NAME, etudiant.getPrenom());
    values.put(KEY_CNE, etudiant.getCNE());
    values.put(KEY_DATE, dateFormat.format(new Date()) );
    // Inserting Row
    db.insert(TABLE_ETUDIANTS, null, values);
    db.close(); // Closing database connection
}

```

- La méthode retourne étudiant par son ID :

```

// Getting single etudiant
public Etudiant getEtudiant(int id) {
    SQLiteDatabase db = this.getReadableDatabase();
    //first way
    /*String selectQuery = "SELECT * FROM " + TABLE_ETUDIANTS
        + " WHERE " + KEY_ID + " = " + id;
    Log.e("DtabaseHelper", selectQuery);
    Cursor cursor = db.rawQuery(selectQuery, null); */
    //end first way
    //second way
    Cursor cursor = db.query(TABLE_ETUDIANTS, new String[] { KEY_ID,
        KEY_LAST_NAME, KEY_FIRST_NAME, KEY_CNE }, KEY_ID + "=?",
        new String[] { String.valueOf(id) }, null, null, null );
    //end second way

    if (cursor != null)
        cursor.moveToFirst();

    Etudiant etudiant = new Etudiant(Integer.parseInt(cursor.getString(0)),
        cursor.getString(1), cursor.getString(2), cursor.getString(3));
    // return etudiant
    cursor.close();
    db.close();

    return etudiant;
}

```

- Récupérer tous les étudiants de la base en complétant la de la méthode :

```
public List<Etudiant> getAllEtudiants() { ...}.
```

Utiliser l'instruction select:

```
String selectQuery = "SELECT * FROM " + TABLE_ETUDIANTS;
```

et la boucle

```
do // add etudiant to the list } while (cursor.moveToNext());
```

- Pour mis à jour vous allez utiliser la méthode update() de la classe SQLiteDatabase

```
// Updating single etudiant
public int updateEtudiant(Etudiant etudiant) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_FIRST_NAME, etudiant.getPrenom());
    values.put(KEY_LAST_NAME, etudiant.getNom());
    values.put(KEY_CNE, etudiant.getCNE());
    // updating row
    return db.update(TABLE_ETUDIANTS, values, KEY_ID + " = ?",
        new String[] { String.valueOf(etudiant.getId()) });
}
```

- Ajouter la méthode addAllEtudiants(List<Etudiant> list) à la classe DatabaseHandler.
- Pour la méthode de suppression vous allez utiliser la méthode delete() de la classe SQLiteDatabase

```
// Deleting single etudiant
public void deleteEtudiant(Etudiant etudiant) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_ETUDIANTS, KEY_ID + " = ?",
        new String[] { String.valueOf(etudiant.getId()) });
    db.close();
}
```

- La méthode qui marque un étudiant absent est tous simplement une méthode d'ajout dans la table Date_d_Absence. La date d'absence par défaut est la date d'aujourd'hui et c'est le moteur SQLite qui va s'occuper d'insérer cette date.
- L'appel à la méthode public void unmarkeAbsent(Etudiant etudiant, Date date) supprimera un enregistrement de la table Date_d_Absence. Donc il suffit d'utiliser l'instruction delete(), Mais il ne faut pas oublier de formater la date.
- La méthode qui retourne le nombre de fois qu'un étudiant était absent est la suivante :

```
//count absences for a student
public int getAbsencesCount(Etudiant etudiant){
    String countQuery = "SELECT * FROM " + TABLE_DATE_ABSENCE + " WHERE "
        + KEY_ID_ETUDIANT + " = "
        + String.valueOf(etudiant.getId());
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(countQuery, null);
    cursor.close();
    // return count
    return cursor.getCount();
}

public int getAbsencesCount(){
    int count = 0;
    String countQuery = "SELECT * FROM " + TABLE_DATE_ABSENCE ;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(countQuery, null);
    count = cursor.getCount();
    cursor.close();
    db.close();
    // return count
    return count;
}
```

- Pour la méthode qui retourne les dates d'absences d'un étudiant vous allez utiliser l'instruction SQL Select :

```
public List<Date> getAbsenceDates(Etudiant etudiant){
    List<Date> abs = new ArrayList<Date>();
    String countQuery = "SELECT * FROM " + TABLE_DATE_ABSENCE + " WHERE "
        + KEY_ID_ETUDIANT + " = " + String.valueOf(etudiant.getId());
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(countQuery, null);
    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            try {
                abs.add(this.dateFormat.parse(cursor.getString(2)));
            } catch (ParseException e) {
                Log.getStackTraceString(e);
            }
        } while (cursor.moveToNext());
    }
    cursor.close();
    db.close();
    return abs;
}
```

- **Tester votre base de données**

- Tester votre base de données en appelant la méthode teterDB() suivante après setContentView() de l'activité principale de votre projet :

```
private void teterDB(){
    DatabaseHelper dbh = new DatabaseHelper(this);
    Etudiant ali = new Etudiant(1, "Ali", "Ahmed", "24584545");
    dbh.addEtudiant(new Etudiant(1, "Ali", "Ahmed", "24584545"));
    dbh.addEtudiant(new Etudiant(2, "Amar", "Salim", "5855885"));
    dbh.addEtudiant(new Etudiant(3, "Fati", "Fati", "15555"));
    dbh.addEtudiant(new Etudiant(4, "Salma", "Mona", "8845555"));

    dbh.markeAbsent(ali);
    Log.i("absences de ali ", dbh.getAbsencesCount(ali)+"");

    dbh.unmarkeAbsent(ali, dbh.getAbsenceDates(ali).get(0));
    Log.i("absences de ali ", dbh.getAbsencesCount(ali)+"");

    List<Etudiant> etudiants = dbh.getAllEtudiants();

    for (Etudiant etudiant : etudiants) {
        String log = "Id: "+etudiant.getId()+" ,Nom: "
            + etudiant.getNom() + " ,Prenom: "
            + etudiant.getPrenom() + " , CNE: "
            + etudiant.getCNE();
        Log.d("Etudiants: ", log);
    }
}
```

Une fois le teste est réussit commenter ou supprimer l'appel de la méthode de teste teterDB().

- **Lecture d'un fichier Texte**

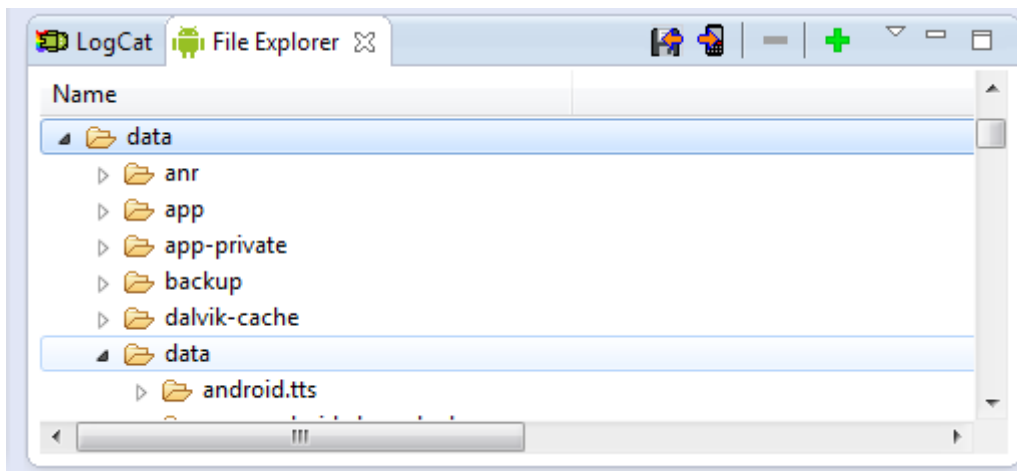
On veut, dans cette partie, lire un fichier texte stocké en interne contenant la liste des étudiants. Vous devez écrire une fonction uploadFile() retournant un objet de type List<Etudiant>. On suppose que le fichier se trouve dans le dossier files interne, utiliser la méthode openFileInput(), la classe

Scanner, ses méthodes `hasNextLine()` et `nextLine()`, et la méthode `split` de la classe `String` pour compléter la méthode suivante :

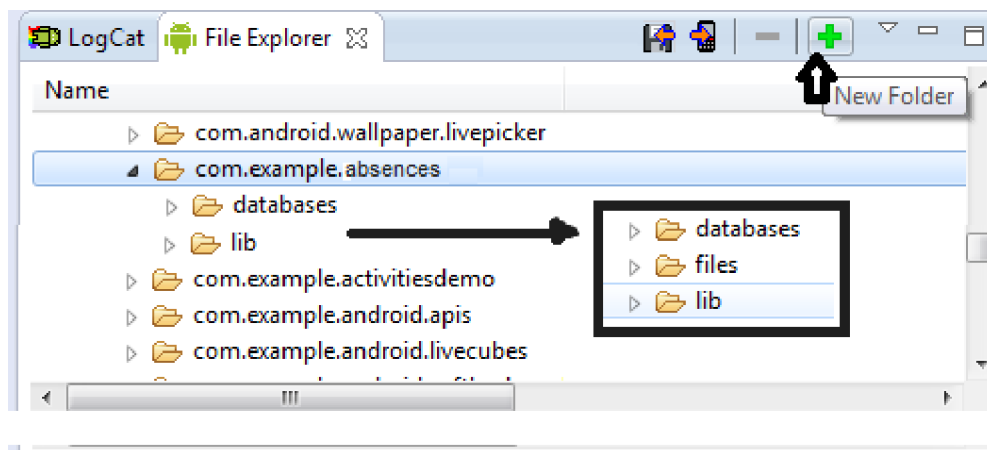
```
private List<Etudiant> uploadFile() {  
    List<Etudiant> etudiants = new ArrayList<Etudiant>();  
    //Votre code ici  
    return etudiants;  
}
```


Vous devez tous d'abord télécharger le fichier `listEtudiant.txt` dans votre le téléphone virtuelle (l'emulateur). Les étapes d'ajout du fichier `listEtudiant.txt` en interne du device sont les suivants :

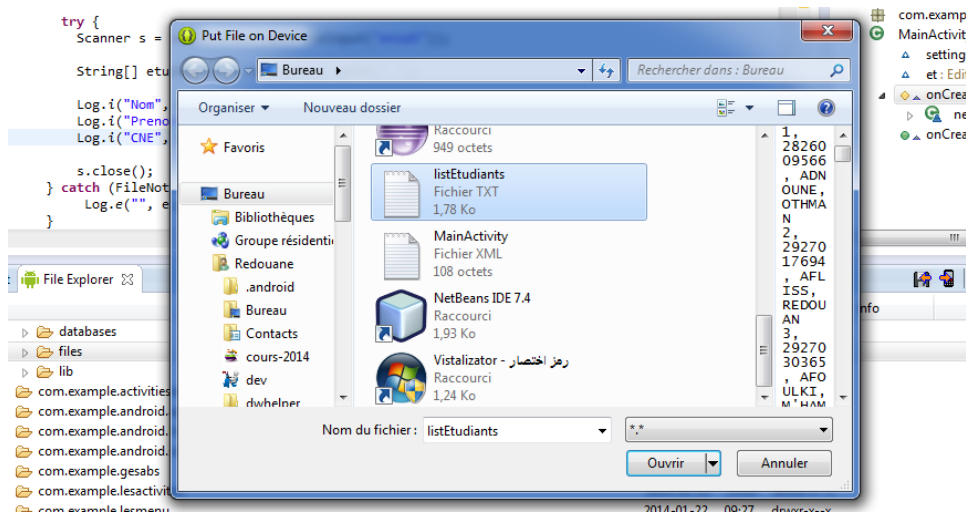
1. Ouvrir File Explorer :
Window → Show View → Other... → Android → File Explorer
2. Sélectionner `data` → `data` → `com.example.absences`



3. Cliquer sur le bouton **+** pour créer un dossier « files »



4. Sélectionner le répertoire « files » puis cliquer sur l'icône  pour ajouter le fichier `listEtudiant.txt`



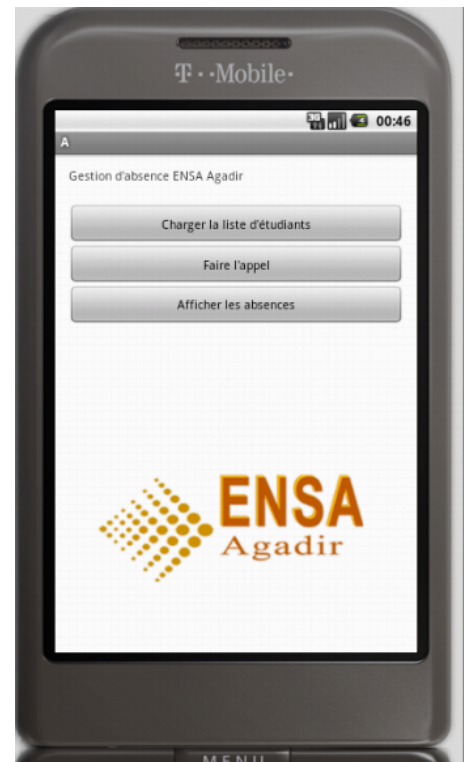
Ajouter deux boutons à l'interface utilisateur : le premier bouton pour charger la liste des étudiants dans la base et le deuxième pour faire l'appel et enregistrer les étudiants absents.

A ajouter dans le fichier layout.xml :

```
<Button
    android:id="@+id/button1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="24dp"
    android:padding="10dp"
    android:text="Charger la liste d'étudiants" />

<Button
    android:id="@+id/button2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button1"
    android:layout_below="@+id/button1"
    android:padding="10dp"
    android:text="Faire l'appel" />

<Button
    android:id="@+id/button3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button2"
    android:text="Afficher les absences" />
```



Configurer l'écouteur de l'évènement clique du bouton1.

Déclarer l'attribut db : protected DatabaseHelper db ;

```
db = new DatabaseHelper(this);
((Button)findViewById(R.id.button1)).setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        db.addAllEtudiants(uploadFile());
    }
});
```

Avant de configurer l'écouteur de l'évènement clique du bouton2. Créer une activité ListeAppelActivity héritant de ListActivity. Cette activité recevra la liste des étudiants dans l'extra de son Intent. N'oubliez pas d'ajouter le widget ListView avec l'Id @android:id/list dans le fichier layout de cette activité.

```
public class ListeAppelActivity extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_liste_appel);
        @SuppressWarnings("unchecked")
        List<Etudiant> values= (List<Etudiant>)
        this getIntent().getExtras().get("Liste");

        ArrayAdapter<Etudiant> adapter = new ArrayAdapter<Etudiant>(this,
            android.R.layout.simple_list_item_1, values);
        setListAdapter(adapter);
    }
}
```

Pour que l' ArrayAdapter<Etudiant> fonction avec ListView qui affiche du texte il faut ajouter la méthode toString à la classe Etudiant.

```
public String toString(){
    return this.nom+ " "+this.prenom;
}
```

- Maintenant l'évènement onclick pour le bouton 2 :

```
((Button)findViewById(R.id.button2)).setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(MainActivity.this,
            ListeAppelActivity.class);
        intent.putExtra("Liste", (Serializable) db.getAllEtudiants());
        startActivity(intent);
    }
});
```

- Maintenant l'évènement onclick pour le bouton 3 pour tester est ce que les absences sont bien enregistrés :

```
((Button)findViewById(R.id.button3)).setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        int nbAbsences = db.getAbsencesCount();
        Toast.makeText(MainActivity.this, "Nombre d'absences : "+
            nbAbsences, Toast.LENGTH_SHORT);
    }
});
```