


Android 4.4, KitKat

It's our goal with Android KitKat to make an amazing Android experience available for everybody.



CHAPITRE 10
GESTION DES DONNÉES

• Mr. MEGHAZI

2019-2020

Cours pour les Master II - GL

Contenu

- SharedPreferences
- Stockage Interne
- Stockage Externe
- Les bases de données SQLite

SharedPreferences

- De petites masses de données primitive
 - ▣ Nom d'utilisateur
 - ▣ E-mail
 - ▣

3

Stockage Interne

- Petites à des masses moyennes de données privées:
 - ▣ Fichiers temporaires
 - ▣ ...

4

Stockage Externe

- Masse de données plus importante et non-privé
 - Vidéo
 - Fichiers audio (chansons)
 - ...

5

Bases de données

- Stocke de petites à de grande masse de données privées et structurées.

6

SharedPreferences (1)

- Un **Map** persistant qui enveloppe des paires **Clé-Valeur** de données simples.
- *Automatiquement persisté* à travers les sessions d'applications

7

SharedPreferences (2)

- Souvent utilisé pour le stockage à long terme des données personnalisable d'applications.
 - Nom d'un compte
 - Nom de Réseau WIFI préféré
 - Données personnalisées de l'user
 - ...

8

SharedPreferences d'une activité

- Pour avoir un objet «SharedPreferences » associé à une activité donnée :

```
Activity.getSharedPreferences (int mode)  
MODE_PRIVATE
```

9

Les SharedPreferences nommées

```
Context.getSharedPreferences (  
    String name, int mode)
```

- **Name** – nom du fichier du **SharedPreferences**
- **Mode** – **MODE_PRIVATE**

10

Ecrire une SharedPreferences (1)

- Invoquer **SharedPreferences.edit()**
- Retourne une instance de : **SharedPreferences.Editor**

11

Ecrire une SharedPreferences (2)

- Ajouter des valeurs au «SharedPreferences » en utilisant l'instance du **SharedPreferences.Editor** :
 - **putInt**(String key, int value)
 - **putString**(String key, String value)
 - **remove**(String key)

12

Ecrire une SharedPreferences (3)

- Valider les valeurs éditées avec:

```
SharedPreferences.Editor.commit()
```

13

Lire les SharedPreferences

- Utiliser les méthodes des « SharedPreferences » pour lire les valeurs:

- `getAll()`
- `getBoolean(String key, ...)`
- `getString(String key, ...)`
- ..

14

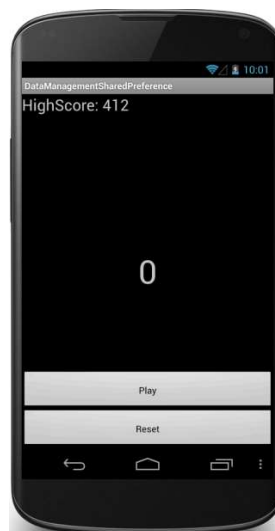
Exemple (1)

DATA MANAGEMENT SHARED PREFERENCES

- Quand l'utilisateur clique sur le bouton « Play », l'application affiche un nombre généré aléatoirement.
- L'application garde trace de la plus grande valeur vue.

15

Exemple (2)



16

Exemple (3)

DATA MANAGEMENT SHARED PREFERENCES

```
final SharedPreferences prefs = getPreferences(MODE_PRIVATE);

setContentView(R.layout.main);

// High Score
final TextView highScore = (TextView) findViewById(R.id.high_score_text);
highScore.setText(String.valueOf(prefs.getInt(HIGH_SCORE, 0)));

//Game Score
final TextView gameScore = (TextView) findViewById(R.id.game_score_text);
gameScore.setText(String.valueOf("0"));
```

17

Exemple (4)

DATA MANAGEMENT SHARED PREFERENCES

```
// Play Button
final Button playButton = (Button) findViewById(R.id.play_button);
playButton.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {

        Random r = new Random();
        int val = r.nextInt(1000);
        gameScore.setText(String.valueOf(val));

        // Get Stored High Score
        if (val > prefs.getInt(HIGH_SCORE, 0)) {

            // Get and edit high score
            SharedPreferences.Editor editor = prefs.edit();
            editor.putInt(HIGH_SCORE, val);
            editor.commit();

            highScore.setText(String.valueOf(val));

        }
    }
});
```

18

La classe « File »

- Une classe qui peut représenter une entité du système de fichiers identifiée par un chemin d'accès.
- Deux zones de stockage, **Interne** et **externe** :
 - **Mémoire interne**: fréquemment utilisée pour de petit ensembles de données et privés de l'application.
 - **Mémoire externe**: fréquemment utilisée pour de grand ensembles de données et non-privés.

19

Les API de la classe « File »

FileOutputStream

- **openFileOutput**(String name, int mode)
 - Ouvre un fichier privé en écriture. Crée le fichier si le fichier n'existe pas.

FileInputStream

- **openFileInput**(String name)
 - Ouvre un fichier privé en lecture.

N.B: Voir la documentation

20

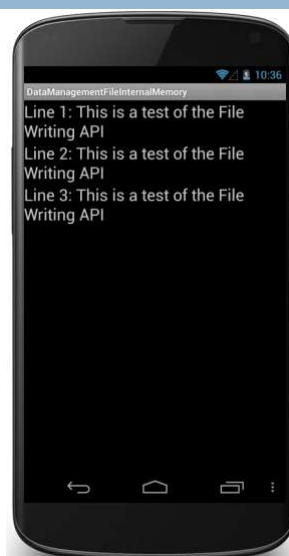
Exemple (1)

DATA MANAGEMENT FILE INTERNAL MEMORY

- Si un certain fichier texte n'existe pas l'application crée et écrit du texte là dedans.
- L'application lit les données ensuite du fichier texte et les affiche.

21

Exemple (2)



22

Exemple (3)

DATAMANAGEMENTFILEINTERNALMEMORY

```
private void writeFile() throws FileNotFoundException {
    FileOutputStream fos = openFileOutput(fileName, MODE_PRIVATE);
    PrintWriter pw = new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(fos)));
    pw.println("Line 1: This is a test of the File Writing API");
    pw.println("Line 2: This is a test of the File Writing API");
    pw.println("Line 3: This is a test of the File Writing API");
    pw.close();
}
```

23

Exemple (4)

DATAMANAGEMENTFILEINTERNALMEMORY

```
private void readFile(LinearLayout ll) throws IOException {
    FileInputStream fis = openFileInput(fileName);
    BufferedReader br = new BufferedReader(new InputStreamReader(fis));
    String line = "";
    while (null != (line = br.readLine())) {
        TextView tv = new TextView(this);
        tv.setTextSize(24);
        tv.setText(line);
        ll.addView(tv);
    }
    br.close();
}
```

24

Utilisation des fichiers des mémoires externes (1)

- Les supports amovibles peuvent apparaître / disparaître sans avertissement.
- `String Environment.getExternalStorageState()`
 - `MEDIA_MOUNTED`- present & mounted with read/write access
 - `MEDIA_MOUNTED_READ_ONLY`- present & mounted with read-only access
 - `MEDIA_REMOVED`- not present

25

Utilisation des fichiers des mémoires externes (2)

- Permissions pour écrire des fichiers dans la mémoire externe:

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

26

Exemple (1)

DATA MANAGEMENT FILE EXTERNAL MEMORY

1. Application qui lit une image du dossier ressources.
2. Copie le fichier sur le support de stockage externe.
3. Lit les données de l'image sur le support de stockage externe.
4. Affiche l'image

27

Exemple (2)

DATA MANAGEMENT FILE EXTERNAL MEMORY



28

Exemple (3)

DATA MANAGEMENT FILE EXTERNAL MEMORY

```
private void copyImageToMemory(File outFile) {
    try {
        BufferedOutputStream os = new BufferedOutputStream(
            new FileOutputStream(outFile));

        BufferedInputStream is = new BufferedInputStream(getResources()
            .openRawResource(R.raw.painter));

        copy(is, os);
    } catch (FileNotFoundException e) {
        Log.e(TAG, "FileNotFoundException");
    }
}
```

29

Exemple (4)

DATA MANAGEMENT FILE EXTERNAL MEMORY

```
private void copy(InputStream is, OutputStream os) {
    final byte[] buf = new byte[1024];
    int numBytes;
    try {
        while (-1 != (numBytes = is.read(buf))) {
            os.write(buf, 0, numBytes);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            is.close();
            os.close();
        } catch (IOException e) {
            Log.e(TAG, "IOException");
        }
    }
}
```

30

Les fichiers « cache » (1)

- Ce sont des fichiers temporaires qui peuvent être supprimés par le système lorsque le stockage est faible.
- Fichiers supprimés quand l'application qui l'utilise est désinstallée.

31

Les fichiers « cache » (2)

File Context.**getCacheDir()**

- Retourne le chemin absolu vers un répertoire spécifique à l'application qui peut être utilisé pour le stockage des fichiers temporaires.

32

Sauvegarde des fichiers « cache » (3)

File Context.**getExternalCacheDir()**

- Retourne un fichier représentant un répertoire sur un support de stockage externe pour les fichiers « caches »

33

SQLite

- SQLite fournit une base de donnée chargée en mémoire.
- Conçu pour opérer avec des BDs à petites tailles (<300Ko).
- Implémente la majorité du **SQL92**
- Prends en charge les Transactions **ACID**
 - A ?, C ?, I ?, D ?

34

SQLite – utilisation d'une BD (1)

- Les méthodes recommandées se basent sur une classe « Helper » nommée « **SQLiteOpenHelper** »
 - Sous Classe de « **SQLiteOpenHelper** »
 - Invoquer **super()** à partir du constructeur de la sous-classe pour initialiser la base de données sous-jacente.

35

SQLite – utilisation d'une BD (2)

- Réécriture (Override) **onCreate()**
- Réécriture (Override) **Upgrade()**
- Exécution des commandes **CREATE TABLE**

36

SQLite – utilisation d'une BD (3)

- Utiliser les méthodes de «[SQLiteOpenHelper](#)» pour ouvrir et renvoyer la base de données sous-jacente.
- Exécuter les opérations sur la base de données sous-jacente.

37

DataManagmentSQL

- Une application qui crée une base de données SQLite et insère des enregistrements avec des erreurs.
- Quand l'utilisateur appuie sur le bouton « Fix », l'application supprimera, mettra-à-jour et affichera les bons enregistrements.

38

DataManagmentSQL (1)

```
// Insert several artist records
private void insertArtists() {

    ContentValues values = new ContentValues();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Brayan Adams");
    mDB.insert(DatabaseOpenHelper.TABLE_NAME, null, values);

    values.clear();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Selena Gomez");
    mDB.insert(DatabaseOpenHelper.TABLE_NAME, null, values);

    values.clear();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Norah Jones");
    mDB.insert(DatabaseOpenHelper.TABLE_NAME, null, values);

    values.clear();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Ludwig von Beethoven");
    mDB.insert(DatabaseOpenHelper.TABLE_NAME, null, values);
}

// Returns all artist records in the database
private Cursor readArtists() {
    return mDB.query(DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.columns, null, new String[] {}, null, null,
        null);
}
```

39

DataManagmentSQL (2)

```
// Returns all artist records in the database
private Cursor readArtists() {
    return mDB.query(DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.columns, null, new String[] {}, null, null,
        null);
}
```

40

DataManagmentSQL (3)

```
// Modify the contents of the database
private void fix() {

    // Sorry Selena Gomez :-(
    mDB.delete(DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.ARTIST_NAME + "=?",
        new String[] { "Selena Gomez" });

    // fix the Canadian singer
    ContentValues values = new ContentValues();
    values.put(DatabaseOpenHelper.ARTIST_NAME, "Bryan Adams");

    mDB.update(DatabaseOpenHelper.TABLE_NAME, values,
        DatabaseOpenHelper.ARTIST_NAME + "=?",
        new String[] { "Brayan Adams" });

}
```

41

DataManagmentSQL (4)

```
// Delete all records
private void clearAll() {

    mDB.delete(DatabaseOpenHelper.TABLE_NAME, null, null);

}

// Close database
@Override
protected void onDestroy() {

    mDB.close();
    mDbHelper.deleteDatabase();

    super.onDestroy();

}
}
```

42

Examiner la BD à distance

- Les bases de données sont stockées sous:
 - /data/data/<package name>/databases/
- On peut examiner une BD avec SQLite3
 - # adb-s emulator-5554 shell
 - # sqlite3
/data/data/course.examples.DataManagement.DataBaseExample/
databases/artist_db