APPLICATIONS MOBILES

PERSISTANCE DES DONNÉES
OPTIONS DE STOCKAGE

- Pour stocker les données, Android fournit plusieurs options :
  - Préférences partagées
    - Utilisé pour les fichiers de préférences. C' est un mécanisme de sauvegarde de paires (clé/valeur) dans des fichiers de préférences (fichier de sauvegarde de paramètres de l' application).
  - Stockage interne
    - Stockage des données privées sur la mémoire interne de l' appareil
  - Stockage externe
    - Stockage des données publiques sur la mémoire externe de l' appareil
  - Bases de données SQLite
    - Stockage des données structurées sur des bases de données (généralement privées à l' application)
  - Réseau
    - Pour le stockage des données sur le web
- Pour partager des données privées avec d' autres applications, Android offre l' option de Content Providers

PERSISTANCE DES DONNÉES
PERSISTANCE LOCALE VS CLOUD

Persistance locale	Sur le Cloud
Mode Offline	Mode Online
Performance	Facile à partager
Difficultés de MAJ du schéma et de synchronisation	Peu de souci de synchronisation
	Consomme de la BP

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES

- Mécanisme de stockage clé/valeur
- Idéal pour stocker des données basiques (entiers, booléens,…)
- Convient aux Paramètres persistés

PERSISTANCE DES DONNÉES PRÉFÉRENCES PARTAGÉES

- Pour sauvegarder certaines informations, entre les lancements de l'application :
  - Paramètres
  - Ou des préférences que l'utilisateur choisit sur l'application
- SharedPreferences permet de sauvegarder ce genre d'informations sous forme de paires (clé/valeur)
  - Valeur peuvent être de l'un des types de base : boolean, int, float ou string
- Ces données persistent même si l'application n'est plus en exécution (stoppée, ou fermée)
- Toute application Android possède un espace de stockage pour les préférences partagées (espace privé à l'application vue qu'elle s'exécute dans sa SandBox)

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES

- La classe **SharedPreferences** permet de gérer des paires de clé/valeurs associées à une activité

- Pour récupérer un objet de type SharedPreferences :
  - getPreferences(int) : lorsque l'activité a un seul fichier de préférences
    - Elle appelle la méthode getSharedPreferences(String, int) avec comme string le nom de la classe de l'activité courante.
    - Renvoie les préférences de l'activité
  - getSharedPreferences(String, int) : lorsque l'activité possède plusieurs fichiers de préférences
    - le nom du fichier à ouvrir est donné comme premier paramètre à la méthode
    - renvoie les préférences partagées

- Le paramètre int : est le mode d'ouverture du fichier de préférences partagées

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES

- Exemple :

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
String nom = prefs.getString("login", null);
Long nom = prefs.getLong("taille", null);
```

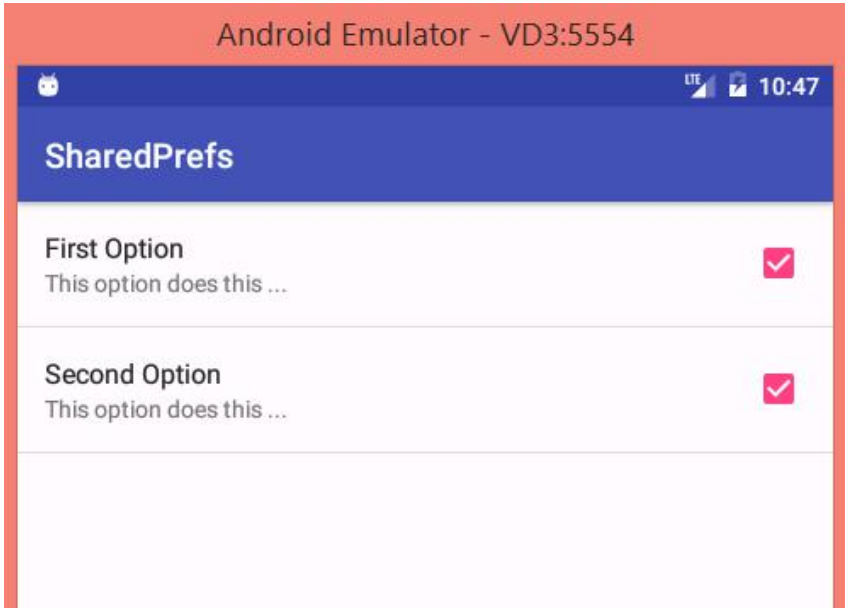
- Les méthodes getXXX lisent les données en fournissant une clé

- Le mode MODE_PRIVATE restreint l'accès au fichier créé à l'application.

- Les modes d'accès MODE_WORLD_READABLE et MODE_WORLD_WRITABLE permettent aux autres applications de lire/écrire ce fichier.

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES (XML)

- Système de préférences prévu par Android
- Intérêt : interface graphique pour modifier les préférences est déjà programmée
  - Donc pas besoin de créer une activité pour les préférences
  - Elle est générée automatiquement par Android (modifiable bien sûr ou personnalisable)



PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES (XML)

- Interface graphique des préférences déjà programmée :
- C.à.d : une description XML existe
  - modificateur prédéfinis pour chaque type de préférence
  - Qu'on peut utiliser pour modifier/afficher les options ou préférences voulues
- Exemple :

```
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:key="first_preferencescreen">
  <CheckBoxPreference
          android:key="wifi enabled"
          android:title="WiFi" />
  <PreferenceScreen
          android:key="second_preferencescreen"
          android:title="WiFi settings">
      <CheckBoxPreference
              android:key="prefer wifi"
              android:title="Prefer WiFi" />
      ... other preferences here ...
  </PreferenceScreen>
</PreferenceScreen>
```

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES (XML)

- Une activité spécifique a été programmée pour réaliser un écran d'édition de préférences. Il s'agit de *PreferenceActivity*.
- Pour afficher l'écran d'édition des préférences correspondant à sa description XML:
  - Il faut créer une nouvelle activité qui hérite de *PreferenceActivity*
- Pour associer le fichier xml de l'interface des préférences :
  - appeler la méthode **addPreferencesFromResource** en donnant l'id de la description XML

```
public class MyPrefs extends PreferenceActivity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferences);
}}
```

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES (XML)

- Pour lancer cette l'activité MyPrefs, on crée un bouton et une **Intent** correspondante.
- Attributs des préférences :
  - **android:title**: apparait comme nom de la préférence
  - **android:summary**: une phrase pour décrire la préférence

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

    <SwitchPreference
        android:defaultValue="true"
        android:key="example_switch"
        android:summary="this is a description summary"
        android:title="Some title" />
    <CheckBoxPreference
        android:key="first"
        android:title="First Option"
        android:summary="This option does this ..."
        android:dependency="example_switch">
    </CheckBoxPreference>
```

PERSISTANCE DES DONNÉES PRÉFÉRENCES PARTAGÉES (XML)

- D'autres attributs spécifiques à certains types de préférences peuvent être utilisés :
  - **android:summaryOn** : qui donne la chaine à afficher lorsque la préférence est cochée
  - **android:dependency** : faire dépendre une préférence d'une autre.

- Pour accéder aux valeurs des préférences :
  - on utilise la méthode **getDefaultSharedPreferences** de la classe **PreferenceManager**.
- pour récupérer une valeur des préférences :
  - On utilise sa clé (android:key) avec *getString* par exemple

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(
      getApplicationContext());
String login = prefs.getString("login","");
```

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES (TOGGLE ET TEXTE)

- Une case à cocher se fait à l'aide de *CheckBoxPreference*

```
<CheckBoxPreference android:key="wifi"
  android:title="Utiliser le wifi"
  android:summary="Synchronise l'application via le wifi."
  android:summaryOn="L'application se synchronise via le wifi."
  android:summaryOff="L'application ne se synchronise pas."
/>
```

- *Un champs texte est saisi via EditTextPreference*

```
<EditTextPreference android:key="login&"
  android:title="Login utilisateur"
  android:summary="Renseigner son login d'authentification."
  android:dialogTitle="Veuillez saisir votre login"
/>
```

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES (LISTES)

- Une entrée de préférence peut être liée à une liste de paires de clef-valeur dans les ressources:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="key">
        <item>"Petite"</item>
        <item>"Moyenne"</item>
        <item>"Grande"</item>
    </array>

    <array name="value">
        <item>"1"</item>
        <item>"5"</item>
        <item>"20"</item>
    </array>
</resources>
```

- qui se déclare dans le menu de préférences
  - Lorsque l'on choisit la valeur "Petite", la préférence *vitesse* est associée à "1"

```xml
<ListPreference android:key="Vitesse"
    android:title="Vitesse"
    android:entries="@array/key"
    android:entryValues="@array/value"
    android:dialogTitle="Choisir la vitesse"
    android:persistent="true">
</ListPreference>
```

PERSISTANCE DES DONNÉES
PRÉFÉRENCES PARTAGÉES (ÉCRITURE)

- Il est aussi possible d'écraser des préférences par le code, par exemple :
  - si une action fait changer le paramétrage de l'application ou
  - si l'on reçoit le paramétrage par le réseau.
- Pour modifier le fichier des préférences, récupéré avec getPreferences(Context.MODE_PRIVATE) :
  1. Appeler la méthode edit() pour avoir un SharedPreferences.Editor
  2. Ajouter les paires (clé/valeur) avec putBoolean() ou putString()
  3. Valider avec la méthode commit()

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
Editor editor = prefs.edit();
editor.putString("login", "jf");
editor.commit();
```

PERSISTANCE DES DONNÉES
STOCKAGE INTERNE

- Le stockage interne permet aux applications d’accéder et de créer leurs propres fichiers
- Par défaut, les fichiers d’une application sauvegardés sur la mémoire interne sont privés
- Ces fichiers sont supprimés, si l’application est désinstallée

PERSISTANCE DES DONNÉES
CLASSES & MÉTHODE DE LECTURE / ECRITURE

- La classe « FileOutputStream » crée un flux en écriture.
- La classe « FileInputStream » crée un flux en lecture.
- La méthode « openFileOutput » obtient un flux en écriture.
- La méthode « openFileInput » obtient un flux en lecture.
- Utilisent les mêmes principes de gestion de fichier de Java.

PERSISTANCE DES DONNÉES
ECRITURE - FICHIER INTERNE

- Pour créer et modifier un fichier privé en mémoire interne :
  - On utilise la méthode openFileOuput()

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

PERSISTANCE DES DONNÉES
STOCKAGE INTERNE (LECTURE)

- Pour lire un fichier privé en mémoire interne :
  - On utilise la méthode openFileInput()

```
String Message;
FileInputStream fis = openFileInput("FileTest");
InputStreamReader isr = new InputStreamReader(fis);
BufferedReader bf = new BufferedReader(isr);
StringBuffer sb = new StringBuffer();
while((Message = bf.readLine()) != null){
    sb.append(Message + "\n");
}
tv.setText(sb.toString());
```

PERSISTANCE DES DONNÉES
STOCKAGE EXTERNE

- Lit et écrit les fichiers sur un référentiel partagé
- Support amovible ou non
- Le support peut ne pas être disponible
- Peut utiliser des répertoires publics ou des répertoires dédiés

PERSISTANCE DES DONNÉES
STOCKAGE EXTERNE

```
public File getFichier(String nom) {
    // renvoie un fichier sur le répertoire téléchargements
    File file = new File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_DOWNLOADS), nom);

    return file;
}
```

PERSISTANCE DES DONNÉES
STOCKAGE EXTERNE (PERMISSIONS)

- L'accès au stockage externe est sujet à des permissions
- La permission READ_EXTERNAL_STORAGE permet d'accéder en lecture à des fichiers sur le stockage externe
- La permission WRITE_EXTERNAL_STORAGE permet d'accéder en ecriture à des fichiers sur le stockage externe

PERSISTANCE DES DONNÉES
STOCKAGE EXTERNE (PERMISSIONS)

- getExternalStorageDirectory: accède au répertoire de stockage externe

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

```
private void Sauver(String text) {
File file = new File(Environment.getExternalStorageDirectory(),
"monfichier.txt");
try {
FileOutputStream stream = new FileOutputStream(file);
stream.write(text.getBytes());
stream.close();
} catch (FileNotFoundException e) {
FragmentTools.alertbox(getActivity(), "fichier non trouvé");
} catch (IOException e) {
// TODO Auto-generated catch block
FragmentTools.alertbox(getActivity(), "erreur d'écriture");;
}

}
```

PERSISTANCE DES DONNÉES
BASES DE DONNÉES SQLITE

- Android permet un accès structuré en combinant SQLite et les fournisseurs de contenu

- SQLite est une BDD relationnelle légère et open source supportée nativement par Android

PERSISTANCE DES DONNÉES
BASES DE DONNÉES SQLITE

- BDD relationnelle et très rapide lancée en 2000
- Open Source
- Supportée nativement dans Android
- Les BDD sont stockées dans le Répertoire /data/package
- Accès exclusif à l'application qui a créé la BDD

PERSISTANCE DES DONNÉES
BASES DE DONNÉES SQLITE (CURSOR)

- Les curseurs sont des pointeurs sur les résultats d'une requête.
- Un curseur permet de renvoyer les données de la position en cours
- Les curseurs peuvent changer de position en utilisant moveToXXX
- getCount renvoie le nombre de lignes
- getXXX renvoie la valeur d'une colonne en donnant sa position
- getColumnName renvoie le nom d'une colonne

PERSISTANCE DES DONNÉES
BASES DE DONNÉES SQLITE (REQUETES)

- SQliteDatabase supporte deux types de requêtes: structurées et natives
- Les méthodes « query » et « rawQuery »
- Les deux méthodes renvoient des curseurs

PERSISTANCE DES DONNÉES
BASES DE DONNÉES SQLITE (CREATION)

- Utiliser SqliteOpen Helper
  - Abstraction permettant de décider si la BDD doit être créée ou mise à jour
- Accès direct
  - Méthode openOrCreateDatabase

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

  private static final int DATABASE_VERSION = 2;
  private static final String DICTIONARY_TABLE_NAME = "dictionary";
  private static final String DICTIONARY_TABLE_CREATE =
              "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
              KEY_WORD + " TEXT, " +
              KEY_DEFINITION + " TEXT);";

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

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

PERSISTANCE DES DONNÉES
BASES DE DONNÉES SQLITE (LECT/ECRIT)

- Pour réaliser des écritures ou lectures, on utilise les méthodes **getWritableDatabase()** et **getReadableDatabase()** qui renvoient

- une instance de **SQLiteDatabase.** Sur cet objet, une requête peut être exécutée au travers de la méthode **query()**:

- L'objet de type Cursor permet de traiter la réponse (en lecture ou écriture), par exemple:
  - getCount(): nombre de lignes de la réponse
  - moveToFirst(): déplace le curseur de réponse à la première ligne
  - getInt(int columnIndex): retourne la valeur (int) de la colonne passée en paramètre
  - getString(int columnIndex): retourne la valeur (String) de la colonne passée en paramètre
  - moveToNext(): avance à la ligne suivante
  - getColumnName(int): donne le nom de la colonne désignée par l'index

BIBLIOGRAPHIE

- Applications Mobile; cours de R.meghatria et M.Khaled ; UDBKM 2018
- **Développement Android - Jean-Francois Lalande** - March 2019
- developer.android.com 2020.