

IFT1155 — Démonstration 3

Guillaume Poirier-Morency `dift1155@iro.umontreal.ca`

Menu

- ▶ persistence
- ▶ tâche asynchrone

Persistence

- ▶ préférences partagées
- ▶ resources
- ▶ système de fichiers
- ▶ base de donnée relationnelle (SQLite)

Préférences partagées

- ▶ stockage clé-valeur
- ▶ écoute des changements
- ▶ éditeur transactionnel

```
getSharedPreferences ("mes-preferences", MODE_PRIVATE)
    .edit ()
    .putInt ("best score", 5)
    .apply (); // ou commit pour vérifier le succès
```

Resources

Les ressources inclut dans le *build* peuvent être accédée directement.

- ▶ statique (ne peuvent être modifiées)
- ▶ coûteux (espace occupé par l'application)
- ▶ facile à intégrer (aucune permissions, accès direct, très rapide)

```
InputStream dataStream = getResources ().openRawResource (R
```

Pour débiter, on peut stocker les données des trajets dans les ressources.

Système de fichiers

- ▶ stockage de fichiers
- ▶ isolation par application

```
FileInputStream inStream = openFileInput ("fichier.txt");  
FileOutputStream outStream = openFileOutput ("fichier.txt")
```

Base de données relationnelles

- ▶ implémente un modèle entité-relation
- ▶ entités et relations représentées par des tables
- ▶ attributs représenté par des colonnes
- ▶ proche du format CSV et autre format de type *record*

```
SQLiteDatabase db = openOrCreateDatabase ("stm-gtfs.sqlite3",
```

```
Cursor trips = db.rawQuery ("select * from trips where route_id = 1", null);
```

Quelques particularités:

- ▶ 4 types de données, INTEGER, TEXT, REAL et BLOB
- ▶ requêtes préparées (?, :param, @param)

Les données de la STM seront stockés dans une base de données SQLite.

AsyncTask

Il y a trois types génériques qui doivent être définis:

- ▶ type des données en entrée
- ▶ type pour la progression (typiquement `java.lang.Integer`)
- ▶ type pour le résultat en sortie

Principalement utile pour les tâches courtes qui risquent de bloquer le *UI thread*.

Pour récupérer des données et populer l'interface, il est préférable d'utiliser un `Loader`.

- ▶ intégration avec le cycle de l'application
- ▶ recharge du contenu au changement

Pour les tâches longue, il est préférable d'utiliser un `Service` dédié et communiquer avec un `Intent`.


```
new AsyncTask<A, B, C> () {  
    @Override  
    protected void doInBackground (A...) {  
        // exécuter la tâche  
        publishProgress (10); // 10%  
    }  
  
    @Override  
    protected onProgressUpdate(B...) {  
        // présenter la progression  
    }  
  
    @Override  
    protected void onPostExecute (C) {  
        // présenter le résultat  
    }  
}
```

Dans le cadre du TP

- ▶ insérer les données dans la base de données
- ▶ calculer les meilleurs trajets