PROGRAMMATION DE COMPOSANTS MOBILES (ANDROID)

WIESLAW ZIELONKA

WWW.IRIF.UNIV-PARIS-DIDEROT.FR/~ZIELONKA

les fichiers dans la mémoire interne de l'appareil

mémoire interne : `getFilesDir() : File`

`getCacheDir()  : File`

retournent objet `File` pointant vers la mémoire interne (mémoire cache interne) où l'activité peut stocker les fichiers.

Notez que en Kotlin au lieu d'écrire `getFilesDir()` on écrit simplement `filesDir` (bien qu'on peut aussi écrire `getFilesDir()`).

`Donc` pour obtenir l'objet `File` pointant vers un fichier dans la mémoire interne dont le nom est `fileName (de type String)` on écrit `:`

```
val file = File( filesDir, nomFichier )
```

Chaque application possède sa propre mémoire interne qui est inaccessible pour d'autres application.

les fichiers dans la mémoire externe de l'appareil

mémoire externe : `getExternalFilesDir() : File`

`getExternalCacheDir() : File`

retournent l'objet `File` pointant vers la mémoire externe de l'appareil. Les deux méthodes prennent en paramètre une constante de la classe Environment

par exemple :

`getExternalFile( Environment.DIRECTORY_DOCUMENT )`

retourne l'objet File vers le répertoire qui sert à stocker des documents. La classe Environment définit beacoup d'autres types de répertoire. Toutes les application partagent la mémoire interne.

Pour obtenir un objet File pour un fichier dans le nom est `fileName` dans le répertoire ci-dessus :

```
val file = File(getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS),
fileName)
```

Les fichiers dans la mémoire interne sont disponibles pour toutes les applications.

File -> FileReader ou File -> FileWriter

Une fois nous avons l'objet

```
file : File
```

correspondant à un fichier, nous pouvons obtenir les `FileReader` ou `FileWriter` appropriés :

```
val fileReader  = FileReader( file )

val fileWriter = FileWriter( file )
```

qui possèdent une kyrielle de méthodes pour la lecture et écriture dans un fichier.

N'oubliez pas `close()` ou `flush()` à la fin.

Mais en cas d'exception pendant la lecture et écriture il faut appliquer close() aussi.

Le code devient encombrant.

fermeture automatique de fichier

Pour éviter close() explicite et éviter les traitement d'exceptions nous pouvons utiliser la construction avec use{} . On suppose que `file` est une référence vers un objet `File` :

```
file.write().use{

            it.writer( texte_à_écrire_dans le fichier )

}
```

la fonction d'extension `writer()` retourne un `FileWriter` et `use{}` applique correctement `close() sur ce FileWriter` même en cas d'exception.

it dans use{} désigne le FileWriter (le récepteur de use).

fermeture automatique de fichier

Pour éviter close() explicite et éviter les traitement d'exceptions nous pouvons utiliser la construction avec use{} . On suppose que `file` est une référence vers un objet `File` :

```
file.reader().use{

        val lines = it.readLines( )

        // traiter la liste  lines : List<String>

          // de lignes

}
```

la fonction d'extension `reader()` retourne un `FileReader` et `use{}` applique correctement `close() sur ce FileReader` même en cas d'exception.

it dans use{} désigne le FileReader (le récepteur de use).

lire un fichier text avec useLines

```
val all = file.readerr().useLines { lines ->
    lines.reduce{some, t -> "$some\n$t" }
}
```

useLines() exécute un block de code et renvoie une Sequence de lignes (de Strings)
obtenue en lisant les lignes du fichier.
Une Séquence de Kotlin ressemble à un Stream de java.

reduce() est une méthode de Séquence qui fait la même chose que reduce() pour les Streams en java

Le résultat :

all = un String avec toutes les lignes contacténées ensemble

fichiers dans les ressources

Placer le fichier dans le répertoire /res/raw/ et ouvrir avec

```
openRawResource(R.raw.nom_fichier) : InputStream
```

Les fichiers dans /res/raw/ ne sont pas modifiables.