

INFO-H-200

Programmation orientée objet

Séance d'exercices 10
Persistance

Université libre de Bruxelles
École polytechnique de Bruxelles

Professeur : Hugues Bersini

2015-2016

Persistence

La **persistence** des données réfère au mécanisme responsable de la sauvegarde et de la restauration de données. Ces mécanismes font en sorte qu'un programme puisse se terminer sans que ses données et son état d'exécution ne soient perdus.

- La sauvegarde sous forme de fichiers texte (Manuel)
- La sérialisation (Semi automatique)
- La sauvegarde dans une base de données (Manuel)

Persistence

Sur fichiers texte

Exemple : Sur fichiers textes

```
public class TextSaved implements Serializable {
    private Integer value;
    private String text;
    public TextSaved(String text, Integer value) {
        this.value = value;
        this.text = text;
    }
}
```

Les attributs étant privés, il est nécessaire de rajouter des méthodes "get" pour y accéder. Il est nécessaire de sauvegarder les attributs 'value' et 'text'. Nous aurons donc un fichier contenant deux valeurs séparées par une virgule, soit : "value, text"

```
TextSaved t = new TextSaved ("Hello World !", 0);
String data = String.valueOf(t.getValue());
data = data + ",";
data = data + t.getText();
data = data + "\n";
BufferedWriter file = new BufferedWriter(new FileWriter(filename));
file.write(data);
file.close();

BufferedReader file2 = new BufferedReader(new FileReader(filename));
while ((line = file2.readLine()) != null) {
    String dat = line.replace("\n", "");
    String[] values = dat.split(",");
    TextSaved t2 = new TextSaved(values[0], Integer.valueOf(values[1]));
}
file2.close();
```

Persistence

Sérialisation

Persistence

La **sérialisation** consiste à écrire des données présentes en mémoire vers un flux de données binaires, ce procédé va donc nous permettre de rendre nos objets persistants.

L'interface **Serializable** permet d'identifier les classes sérialisables, les classes **ObjectOutputStream** et **ObjectInputStream** implémentent, quant à elles, les mécanismes de sérialisation

Persistence

Pour sérialiser un objet d'une classe, celle-ci doit implémenter l'interface **Serializable** ou hériter d'une classe sérialisable

L'interface **Serializable** ne possède ni attribut ni méthode. Elle sert uniquement à identifier une classe sérialisable. Tous les attributs de l'objet sont sérialisés mais à certaines conditions :

- être sérialisable ou être un type primitif (qui sont tous sérialisables)
- ne pas être déclaré à l'aide du mot clé static
- ne pas être déclaré à l'aide du mot clé transient
- être hérité d'une classe mère sérialisable

Persistence

Le mot clé **transient** permet d'interdire la sérialisation d'un attribut d'une classe. Il est en général utilisé pour les données "sensibles" telles que les mots de passe ou tout simplement pour les attributs n'ayant pas besoin d'être sérialisé.

Le **serialVersionUID** est un "numéro de version", associé à toute classe sérialisable, qui permet de s'assurer, lors de la désérialisation, que les versions des classes Java soient concordantes. Si le test échoue, une `InvalidClassException` est levée.

```
| private static final long serialVersionUID = 42L;
```


Example

```
public class TextSaved implements Serializable {  
    private static final long serialVersionUID = 43L;  
    private Integer value;  
    private String text;  
    public TextSaved(String text) {  
        value = 0;  
        this.text = text;  
    }  
}
```


Exemple

Pour l'opération inverse, nous allons utiliser la méthode `readObject()` de la classe **ObjectInputStream**.

```
TextSaved t = null;  
ObjectInputStream ois= new ObjectInputStream(new  
    FileInputStream("personne.serial"));  
t = (TextSaved ) ois.readObject();  
ois.close();
```

Sérialisation et héritage

- ① Premier cas de figure, lorsqu'une classe hérite d'une classe sérialisable, elle se comporte comme si elle avait implémenté l'interface `Serializable`. Les attributs de la classe mère sont sérialisés selon son implémentation.
- ② Deuxième cas de figure, une classe implémente l'interface `Serializable` et hérite d'une classe non sérialisable.
 - les attributs hérités ne sont pas sérialisés.
 - la classe étendue doit posséder un constructeur par défaut accessible ; dans le cas contraire, une `InvalidClassException` est levée à l'exécution.

Personnalisation de la sérialisation

- ① définir notre propre mécanisme de sérialisation mais ceci uniquement pour les attributs propres à la classe (les attributs sérialisables hérités d'une classe sérialisable reste gérés par le mécanisme Java).
- ② sérialiser des attributs que le mécanisme par défaut ne sérialiserait pas (un attribut static par exemple).

Exemple

```
public class TextSaved {  
    private transient int serialisationCount = 0;  
  
    private void writeObject(ObjectOutputStream out) throws IOException {  
        out.defaultWriteObject();  
        out.writeInt(serialisationCount);  
    }  
    private void readObject(ObjectInputStream in) throws IOException,  
        ClassNotFoundException {  
        in.defaultReadObject();  
        serialisationCount = in.readInt();  
    }  
}
```

Persistence

Base de données

Persistence

Avec la plupart des bases de données, les interactions avec une application se font via l'utilisation de requêtes **SQL** (Structured Query Language).

Pour la persistance des données, on va se limiter à trois éléments :

- la création d'une base de données ;
- l'ajout d'un élément ;
- la récupération des éléments.

Example

```
SQLiteConnection db;  
File f = new File(filename);  
db = new SQLiteConnection(f);  
db.open(true);  
db.exec("CREATE TABLE products(id INTEGER PRIMARY KEY, name TEXT, size INTEGER,  
    price INTEGER);");
```

Example

```
db.exec("INSERT INTO products(name, size, price) " + "VALUES('car', 3, 19)");

string t = "car";
int a = 3;
int b = 18;
db.exec("INSERT INTO products(name, size, price) " + "VALUES('" + t + "'," + a +
        ", " + b + ")");
```

Example

```
SQLiteStatement st = db.prepare("SELECT id, name, size, price FROM products;");  
int colnum = st.columnCount();  
string cn = st.getColumnName(i);  
string cv = st.columnValue(i).toString();
```