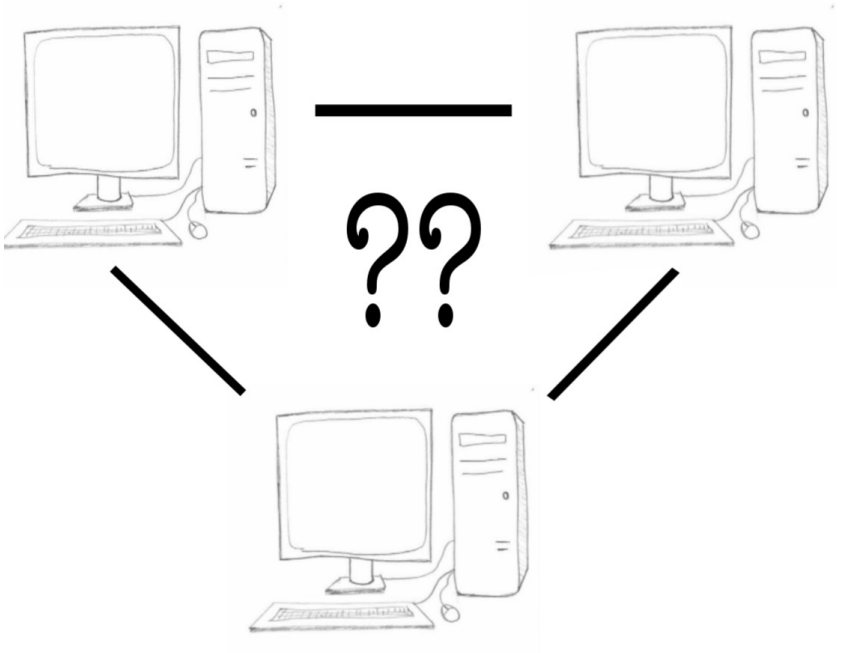
PROGRAMMATION RÉSEAU

Arnaud Sangnier
sangnier@irif.fr

La sérialisation en Java



La sérialisation en bref

- Cela consiste à stocker un objet sous une certaine forme en dehors de la JVM
- Pourquoi ?
  - Avoir une persistance des données
    - On peut stocker un objet et le ``récréer'' ensuite même lors d'une autre exécution
    - Les objets sérialisés peuvent être stockés dans des fichiers ou des bases de données
  - Pour pouvoir transmettre un objet
    - Par exemple sur le réseau
    - Ou bien d'une application à une autre
      - Une application calcule un certain objet et le met ensuite à disposition d'une autre application

Pas seulement de Java à Java

- Le procédé de sérialisation peut être indépendant des langages
- Exemple de frameworks utilisant la sérialisation :
  - .NET de Microsoft
  - C++ (pas de manière native)
  - OCaml
  - Python
  - Etc
- Autres termes pour sérialisation
  - *Marshalling/Unmarshalling*
  - Parfois linéarisation

La sérialisation en Java

- Quels objets peut-on sérialiser en Java
  - Tous ceux implémentant l'interface **Serializable**
  - Notons que cette interface est « vide », elle ne contient aucune méthode implémentée
  - Cependant un objet n'implémentant pas Serializable ne pourra pas être sérialisé
    - **Ainsi les objets champs d'un objet implémentant Serializable doivent aussi implémenter Serializable**
- Il ne faut pas sérialiser n'importe quel objet
  - Par exemple, il n'y a pas de sens à sérialiser un **InputStream**

Comment cela marche-t-il ?

- Comment la sérialisation est-elle faite :
  - L'objet est décomposé en éléments de plus en plus petits (jusqu'à arriver aux éléments de base) et chacun de ces éléments est encodé
  - Ces objets composent un graphe
    - Par exemple, pour se rappeler que le champ de deux objets différents contient le même objet
  - Les champs d'un objet doivent être sérialisables
  - La structure de l'objet doit être conservé pour pouvoir reconstruire l'objet au moment de la « désérialisation »
    - Par exemple, si on a des tableaux, des listes, etc, leur structure doit être préservée

Comment a lieu le stockage ?

- Il existe différents formats possibles :
  - XML
    - Lisible → **XMLEncoder** et **XMLDecoder**
    - Ou binaire
  - JSON (essentiellement lié à JavaScript)
  - YAML (YAML Ain't Markup Language)
  - XRD (External Data Representation)
  - Formats binaires spécifiques (par exemple en Java)

Exemple XML(1)

```
public class SerXMLOut{
    public static void main(String[]args){
        try{
            Joueur j1=new Joueur("Alice",12);
            Joueur j2=new Joueur("Bob",14);
            FileOutputStream fo=new FileOutputStream("Joueurs.xml");
            XMLEncoder xe=new XMLEncoder(fo);
            xe.writeObject(j1);
            xe.writeObject(j2);
            xe.close();
            fo.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Exemple XML(2)

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_65" class="java.beans.XMLDecoder">
 <object class="Joueur">
  <void property="age">
   <int>12</int>
  </void>
  <void property="nom">
   <string>Alice</string>
  </void>
 </object>
 <object class="Joueur">
  <void property="age">
   <int>14</int>
  </void>
  <void property="nom">
   <string>Bob</string>
  </void>
 </object>
</java>
```

Fichiers joueurs.xml

Exemple XML(3)

```
public class SerXMLIn{
    public static void main(String[]args){
        try{
            int nbPers=2;
            Joueur j=null;
            FileInputStream fi=new FileInputStream("Joueurs.xml");
            XMLDecoder xd=new XMLDecoder(fi);
            while(nbPers>0){
                j=(Joueur)xd.readObject();
                System.out.println(j.toString());
                nbPers--;
            }

            xd.close();
            fi.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Le stockage dans un fichier

- Sérialiser dans un fichier en Java
  - Déclarer les objets que l'on sérialise comme implémentant l'interface Serializable
  - On peut ensuite écrire des objets grâce à des flux d'écriture d'objets
    - Classe **ObjectOutputStream**
      - Méthode **void writeObject(Object o)**
      - Si l'objet ou un des objets appartenant à ces champs n'est pas sérialisable, une exception est levée
  - Pour lire les objets
    - Classe **ObjectInputStream**
      - Méthode **Object readObject()**
      - Il faut caster l'objet dans la classe désirée

Exemple (1)

```
import java.io.*;

public class Personne implements Serializable {
    private String nom;
    private Chien chien;

    public Personne(String _nom,Chien _chien){
        this.nom=_nom;
        this.chien=_chien;
    }

    public String toString(){
        return (nom+", "+chien.toString());
    }

    public Chien getChien(){
        return chien;
    }
}
```

Exemple (2)

```
import java.io.*;

public class Chien implements Serializable{
    private String nom;

    public Chien(String _nom){
        this.nom=_nom;
    }

    public String toString(){
        return nom;
    }
}
```

Exemple (3)

```
import java.io.*;

public class SerOut{
    public static void main(String[]args){
        try{
            Chien c1=new Chien("Laika");
            Personne p1=new Personne("Alice",c1);
            Personne p2=new Personne("Bob",c1);
            Personne p3=new Personne("Charles",c1);
            FileOutputStream fo=new FileOutputStream("PersonneObj.bin");
            ObjectOutputStream os=new ObjectOutputStream(fo);
            os.writeObject(p1);
            os.writeObject(p2);
            os.writeObject(p3);
            os.close();
            fo.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Exemple (4)

```
import java.io.*;

public class SerIn{
    public static void main(String[]args){
        try{
            int nbPers=3;
            Personne p=null;
            FileInputStream fi=new FileInputStream("PersonneObj.bin");
            ObjectInputStream ois=new ObjectInputStream(fi);
            while(nbPers>0){
                p=(Personne)ois.readObject();
                System.out.println(p.toString());
                nbPers--;
            }

            ois.close();
            fi.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Sérialiser que certains champs

- Il est possible de préciser qu'au moment de la sérialisation, on ne souhaite pas retenir certains champs d'un objet
  - On utilise le mot clef **transient**
  - L'information contenue dans ces champs ne sera pas enregistré au moment de la sérialisation
- Au moment de la désérialisation, ces champs sont mis à **null**
  - **ATTENTION** aux NullPointerException

Exemple

```
import java.io.*;

public class Personne implements Serializable {
    private String nom;
    private transient Chien chien;

    public Personne(String _nom,Chien _chien){
        this.nom=_nom;
        this.chien=_chien;
    }

    public String toString(){
        return (nom+", "+chien.toString());
    }

    public Chien getChien(){
        return chien;
    }
}
```

Changement

- Il existe un mécanisme de gestion de version de classe
- Il permet de vérifier si la classe a changé

Gestion de versions de classe

- En fait chaque classe implémentant **Serializable** possède un numéro de version
  - Il est stocké dans un champ de type long nommé serialVersionUID
  - C'est soit l'utilisateur qui met sa valeur en faisant par exemple
    - **private static final long serialVersionUID=1 ;**
  - Sinon le compilateur le rajoute
  - La valeur de ce champ est toujours ajouté au moment de la sérialisation
  - Cela permet ainsi de détecter si la classe a changé entre une sérialisation et une désérialisation
  - En cas d'incompatibilité de version de classe, une exception est levée au moment de la désérialisation
  - Si le numéro est géré *'à la main*', il faut faire attention de le changer quand on modifie la classe

Exemple

```
import java.io.*;

public class Personne2 implements Serializable {
    private String nom;
    int age;

    private static final long serialVersionUID=1;

    public Personne2(String _nom,int _age){
        this.nom=_nom;
        this.age=_age;

    }

    public String toString(){
        return (nom+", "+age);
    }
}
```

Sérialisation personnalisée

- Il est également possible de personnaliser la façon dont un objet est sérialisé
- Par exemple, pour omettre certains champs ou rajouter de l'information
- Pour cela il faut que l'objet à sérialiser implémente les méthodes :
  - **private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException**
  - **private void writeObject(ObjectOutptStream oos) throws IOException**
- Ces méthodes sont appelées lors de la désérialisation et de la sérialisation
- Attention, il faut que l'ordre dans lequel on lit les informations soit le même que l'ordre dans lequel on écrit

Exemple

```
public class Personne3 implements Serializable {
    private String nom;
    int age;
    int num;

    public Personne3(String _nom,int _age,int _num){
        this.nom=_nom;
        this.age=_age;
        this.num=_num;

    }

    private void readObject(ObjectInputStream ois)
        throws IOException, ClassNotFoundException{
        nom=(String)ois.readObject();
        age=ois.readInt();

    }

    private void writeObject(ObjectOutputStream oos)
        throws IOException{
        oos.writeObject(nom);
        oos.writeInt(age);
    }

    public String toString(){
        return (nom+", "+age+", "+num);
    }
}
```