Université de Carthage

Institut National des Sciences Appliqués et de Technologie

Année Universitaire 2018-2019

COMPTE RENDU
Applications Réparties
TP I

Réalisée par :

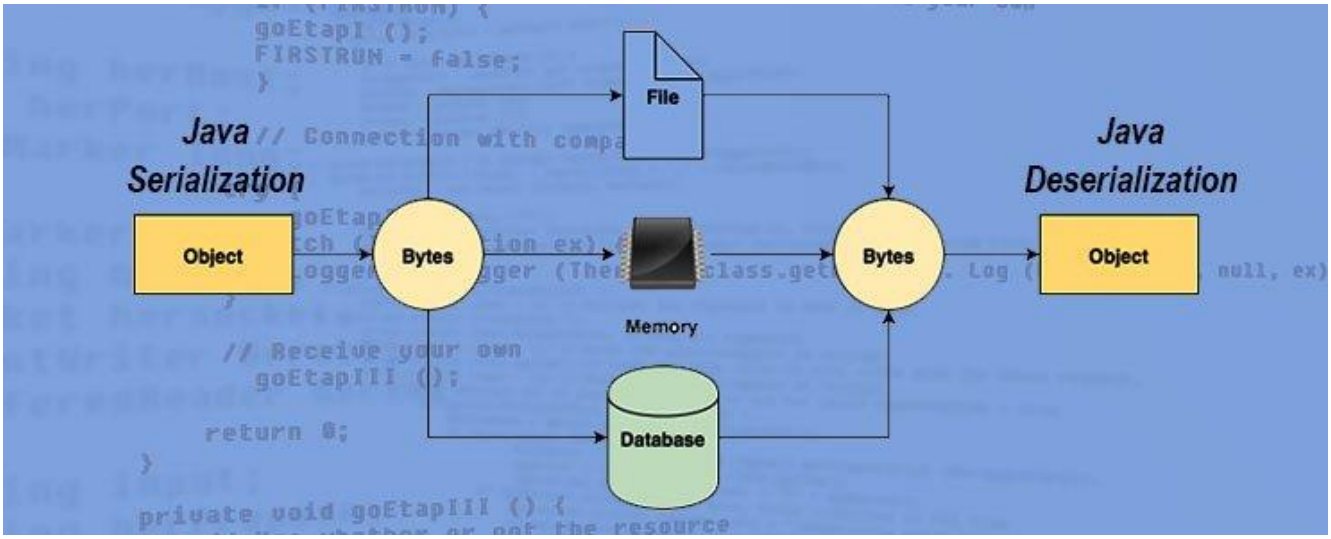
SNOUSSI Anis

BARGHOUDA Mohamed Lamine

LA SERIALISATION

Introduction à la Sérialisation

La sérialisation est un procédé introduit dans le JDK version 1.1 qui permet de rendre un objet ou un graphe d'objets de la JVM persistant pour stockage ou échange et vice versa. Cet objet est mis sous une forme sous laquelle il pourra être reconstitué à l'identique. Ainsi il pourra être stocké sur un disque dur ou transmis au travers d'un réseau pour le créer dans une autre JVM. C'est le procédé qui est utilisé, par exemple, par RMI. La sérialisation est aussi utilisée par les beans pour sauvegarder leurs états.



Au travers de ce mécanisme, Java fournit une façon facile, transparente et standard de réaliser cette opération : ceci permet de facilement mettre en place un mécanisme de persistance. Il est de ce fait inutile de créer un format particulier pour sauvegarder et relire un objet. Le format utilisé est indépendant du système d'exploitation. Ainsi, un objet sérialisé sur un système peut être réutilisé par un autre système pour récréer l'objet.

Première Classe Serializable et Première Sérialisation

La sérialisation utilise l'interface Serializable et les classes ObjectOutputStream et ObjectInputStream.Cette interface ne définit aucune méthode mais permet simplement de marquer une classe comme pouvant être sérialisée.

```
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

public class Personne implements java.io.Serializable {
    private String nom = "";
    private String prenom = "";
    private int taille = 0;
    public Personne(String nom, String prenom, int
taille) {
        this.nom = nom;
        this.taille = taille;
        this.prenom = prenom;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public int getTaille() {
        return taille;
    }
    public void setTaille(int taille) {
        this.taille = taille;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
}
```

Classe Personne

```
import java.io.*;
public class SerializerPersonne {
    public static void main(String args[]) {
        Personne personne = new Personne("Dupond", "Jean", 175);
        try {
            FileOutputStream fichier = new FileOutputStream("personne.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fichier);
            oos.writeObject(personne);
            oos.flush();
            oos.close();
        } catch (java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

Le programme qui sérialise la classe Personne

On définit un fichier avec la classe FileOutputStream. On instancie un objet de classe ObjectOutputStream en lui fournissant en paramètre le fichier : ainsi, le résultat de la sérialisation sera envoyé dans le fichier.

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ ls
Personne.java  SerializerPersonne.java

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ javac Personne.java

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ javac SerializerPersonne.java

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java SerializerPersonne

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ ls
Personne.class  Personne.java  personne.ser  SerializerPersonne.class  SerializerPersonne.java
```

Après l'exécution de cet exemple, un fichier nommé « personne.ser » est créé. On peut visualiser son contenu mais surtout pas le modifier car sinon il serait corrompu. En effet, les données contenues dans ce fichier ne sont pas toutes au format caractères.

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ xxd personne.ser
00000000: aced 0005 7372 0008 5065 7273 6f6e 6e65  ....sr..Personne
00000010: da15 d73f 0707 1005 0c00 0078 7077 1700  ...?.......xpw..
00000020: 0644 4e4f 5055 4400 044e 4145 4a00 0564  .DNOPUD..NAEJ..d
00000030: 2d00 0331 3731 78                        -..171x
```

Le contenue du fichier : personne.ser

DéSérialisation

La classe ObjectInputStream permet de désérialiser un objet.

```
import java.io.*;
public class DeSerializerPersonne {
    public static void main(String argv[]) {
        try {
            FileInputStream fichier = new FileInputStream("personne.ser");
            ObjectInputStream ois = new ObjectInputStream(fichier);
            Personne personne = (Personne) ois.readObject();
            System.out.println("Personne : ");
            System.out.println("nom : " + personne.getNom());
            System.out.println("prenom : " + personne.getPrenom());
            System.out.println("taille : " + personne.getTaille());
        } catch (java.io.IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Résultat de l'exécution :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ javac DeSerializerPersonne.java

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java DeSerializerPersonne
Personne :
nom : dupond
prenom : jean
taille : 175
```

On créée un objet de la classe FileInputStream qui représente le fichier contenant l'objet sérialisé puis un objet de type ObjectInputStream en lui passant le fichier en paramètre. Un appel à la méthode readObject() retourne l'objet avec un type Object.

Exceptions

A. Si la classe Personne a changé entre le moment de sérialisation et désérialisation

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java DeSerializerPersonne
java.io.InvalidClassException: Personne; local class incompatible: stream classdesc serialVersionUID = -2732040933229522939,
local class serialVersionUID = 1522311822689512223
        at java.base/java.io.ObjectStreamClass.initNonProxy(Unknown Source)
        at java.base/java.io.ObjectInputStream.readNonProxyDesc(Unknown Source)
        at java.base/java.io.ObjectInputStream.readClassDesc(Unknown Source)
        at java.base/java.io.ObjectInputStream.readOrdinaryObject(Unknown Source)
        at java.base/java.io.ObjectInputStream.readObject0(Unknown Source)
        at java.base/java.io.ObjectInputStream.readObject(Unknown Source)
        at DeSerializerPersonne.main(DeSerializerPersonne.java:7)
```

B. Si le fichier sérialisé a été corrompu

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java DeSerializerPersonne
java.io.StreamCorruptedException: invalid stream header: ACFF0005
        at java.base/java.io.ObjectInputStream.readStreamHeader(Unknown Source)
        at java.base/java.io.ObjectInputStream.<init>(Unknown Source)
        at DeSerializerPersonne.main(DeSerializerPersonne.java:6)
```

C. Si l'objet est transtypé vers une classe qui n'existe plus

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ mv Personne.class Personne2.class

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java DeSerializerPersonne
java.lang.ClassNotFoundException: Personne
        at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(Unknown Source)
        at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(Unknown Source)
        at java.base/java.lang.ClassLoader.loadClass(Unknown Source)
        at java.base/java.lang.Class.forName0(Native Method)
        at java.base/java.lang.Class.forName(Unknown Source)
        at java.base/java.io.ObjectInputStream.resolveClass(Unknown Source)
        at java.base/java.io.ObjectInputStream.readNonProxyDesc(Unknown Source)
        at java.base/java.io.ObjectInputStream.readClassDesc(Unknown Source)
        at java.base/java.io.ObjectInputStream.readOrdinaryObject(Unknown Source)
        at java.base/java.io.ObjectInputStream.readObject0(Unknown Source)
        at java.base/java.io.ObjectInputStream.readObject(Unknown Source)
        at DeSerializerPersonne.main(DeSerializerPersonne.java:7)
```

Le mot clé transient

Le contenu des attributs est visible dans le flux dans lequel est sérialisé l'objet. Il est ainsi possible pour toute personne ayant accès au flux de voir le contenu de chaque attribut même si ceux si sont private.
Ceci peut poser des problèmes de sécurité surtout si les données sont sensibles.
Java introduit le mot clé transient qui précise que l'attribut qu'il qualifie ne doit pas être inclus dans un processus de sérialisation et donc de désérialisation.

Modifications dans Personne.java :

```
...

private transient String codeSecret = "";

public Personne(String nom, String prenom, int taille, String codeSecret) {
    this.nom = nom;
    this.taille = taille;
    this.prenom = prenom;
    this.codeSecret = codeSecret;
}

public void setCodeSecret(String codeSecret) {
    this.codeSecret = codeSecret;
}

public String getCodeSecret() {
    return codeSecret;
}


...
```

Résultat :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java DeSerializerPersonne
Personne :
nom : Dupond
prenom : Jean
taille : 175
code Secret : null
```

Sérialisation Complexes

1.Sérialisation et héritage

On crée une classe Etudiant (Sérialisable) qui hérite de Personne avec les attributs supplémentaires.

personne.java

```
public class Etudiant extends Personne {
    private int numEtudiant;
    private String niveau;

    public Etudiant(String nom, String prenom, int taille, int numEtudiant,
String niveau) {
        super(nom, prenom, taille);
        this.numEtudiant = numEtudiant;
        this.niveau = niveau;
    }
    public void setNumEtudiant(int numEtudiant) {
        this.numEtudiant = numEtudiant;
    }
    public void setNiveau(String niveau) {
        this.niveau = niveau;
    }
    public int getNumEtudiant() {
        return numEtudiant;
    }
    public String getNiveau() {
        return niveau;
    }
}
```

SerialiserEtudiant.java

```
import java.io.*;
public class SerializerEtudiant {
    public static void main(String args[]) {
        Etudiant etudiant = new Etudiant("SNOUSSI", "Anis", 175, 1700248, "GL2");
        try {
            FileOutputStream fichier = new FileOutputStream("etudiant.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fichier);
            oos.writeObject(etudiant);
            oos.flush();
            oos.close();
        } catch (java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

DeserlialiserEtudiant.java

```
import java.io.*;
public class DeSerializerPersonne {
    public static void main(String args[]) {
        try {
            FileInputStream fichier = new FileInputStream("personne.ser");
            ObjectInputStream ois = new ObjectInputStream(fichier);
            Personne personne = (Personne)
            ois.readObject();
            System.out.println("Personne : ");
            System.out.println("nom : " + personne.getNom());
            System.out.println("prenom : " + personne.getPrenom());
            System.out.println("taille : " + personne.getTaille());
            System.out.println("code Secret : " + personne.getCodeSecret());
        } catch (java.io.IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Le résultat de l'exécution :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java DeSerializerEtudiant
Personne :
nom : BARGHOUDA
prenom : Med Lamine
taille : 175
num Carte Etudiant : 1700248
niveau : GL2
```

2.Sérialisation des objets attributs

Chaque personne possède une Adresse (classe sérialisable) et un Compte bancaire qui lui-même est associé à la même adresse.

Adresse.java

```
class Adresse implements java.io.Serializable {
    private int codePostal = 0;
    private String rue = "";
    Adresse(int codePostal, String rue) {
        this.codePostal = codePostal;
        this.rue = rue;
    }
    void setRue(String rue) {
        this.rue = rue;
    }
    String getRue() {
        return this.rue;
    }
    void setCodePostal(int codePostal) {
        this.codePostal = codePostal;
    }
    int getCodePostal() {
        return this.codePostal;
    }
}
```

CompteBancaire.java

```
public class CompteBancaire implements java.io.Serializable {
    private int idBanque;
    private Adresse adr;
    public CompteBancaire(int idBanque, int codePostal, String rue) {
        this.idBanque = idBanque;
        adr = new Adresse(codePostal, rue);
    }
    public void setIdBanque(int idBanque) {
        this.idBanque = idBanque;
    }
    public int getIdBanque() {
        return idBanque;
    }
    public Adresse getAdresse() {
        return adr;
    }
}
```

Etudiant.java

```
public class Etudiant extends Personne {
    private int numEtudiant;
    private String niveau;
    private CompteBancaire banque;
    private Adresse adr;

    public Etudiant(String nom, String prenom, int taille, int numEtudiant, String niveau,
CompteBancaire banque, Adresse adr) {
        super(nom, prenom, taille);
        this.numEtudiant = numEtudiant;
        this.niveau = niveau;
        this.banque = banque;
        this.adr = adr;
    }
    public void setNumEtudiant(int numEtudiant) {
        this.numEtudiant = numEtudiant;
    }
    public void setNiveau(String niveau) {
        this.niveau = niveau;
    }
```

```
public int getNumEtudiant() {
        return numEtudiant;
    }
    public String getNiveau() {
        return niveau;
    }
    public CompteBancaire getCompteBancaire() {
        return banque;
    }
    public Adresse getAdresse() {
        return adr;
    }
}
```

SerialiserEtudiant.java

```
import java.io.*;
public class SerializerEtudiant {
    public static void main(String args[]) {
        Adresse adr = new Adresse(200, "Ariana");
        CompteBancaire banque = new CompteBancaire(11012, 200, "Ariana");
        Etudiant etudiant = new Etudiant("BARGHOUDA", "Med Lamine", 175, 1700248, "GL2",
banque, adr);
        try {
            FileOutputStream fichier = new FileOutputStream("etudiant.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fichier);
            oos.writeObject(etudiant);
            oos.flush();
            oos.close();
        } catch (java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

DeserlialiserEtudiant.java

```
import java.io.*;
public class DeSerializerEtudiant {
    public static void main(String args[]) {
        try {
            FileInputStream fichier = new FileInputStream("etudiant.ser");
            ObjectInputStream ois = new ObjectInputStream(fichier);
            Etudiant etudiant = (Etudiant)
            ois.readObject();
            System.out.println("Personne : ");
            System.out.println("nom : " + etudiant.getNom());
            System.out.println("prenom : " + etudiant.getPrenom());
            System.out.println("taille : " + etudiant.getTaille());
            System.out.println("num Carte Etudiant : " + etudiant.getNumEtudiant());
            System.out.println("niveau : " + etudiant.getNiveau());
            System.out.println("rue (Etudiant): " + etudiant.getAdresse().getRue());
            System.out.println("code postal  (Etudiant): " +
etudiant.getAdresse().getCodePostal());
            System.out.println("id Banque :" + etudiant.getCompteBancaire().getIdBanque());
            System.out.println("rue (compte bancaire): " +
etudiant.getCompteBancaire().getAdresse().getRue());
            System.out.println("code postal (compte bancaire): " +
etudiant.getCompteBancaire().getAdresse().getCodePostal());
        } catch (java.io.IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Résultat :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java DeSerializerEtudiant
Personne :
nom : BARGHOUDA
prenom : Med Lamine
taille : 175
num Carte Etudiant : 1700248
niveau : GL2
rue (Etudiant): Ariana
code postal  (Etudiant): 200
id Banque :11012
rue (compte bancaire): Ariana
code postal (compte bancaire): 200
```

REMARQUES :

- **Les membres « static » d'un objet ne sont jamais sérialisés car ils ne concernent pas l'état d'un objet mais de tous les objets d'une même classe.**

- **Lorsqu'un objet est désérialisé, le constructeur de sa classe n'est pas invoqué et les variables d'instances ne sont pas initialisées avec les valeurs qui pourraient leur être assignées.**

- **Lorsqu'on sérialise un Objet qui possède un référence sur un autre objet de type différent, ce dernier sera sérialisé avec lui, c'est pourquoi il doit aussi implémenter l'interface Serializable, sinon on aura un exception :**

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ java SerializerEtudiant
java.io.NotSerializableException: CompteBancaire
        at java.base/java.io.ObjectOutputStream.writeObject0(Unknown Source)
        at java.base/java.io.ObjectOutputStream.defaultWriteFields(Unknown Source)
        at java.base/java.io.ObjectOutputStream.writeSerialData(Unknown Source)
        at java.base/java.io.ObjectOutputStream.writeOrdinaryObject(Unknown Source)
        at java.base/java.io.ObjectOutputStream.writeObject0(Unknown Source)
        at java.base/java.io.ObjectOutputStream.writeObject(Unknown Source)
        at SerializerEtudiant.main(SerializerEtudiant.java:10)
```

3.L'interface Externalizable

L'implémentation de l'interface Externalizable permet d'avoir un contrôle très fin sur les opérations de sérialisation et désérialisation lorsque les mécanismes de sérialisation par défaut ne répondent pas au besoin.
L'interface Externalizable hérite de l'interface Serializable. Elle définit deux méthodes :

Méthode	Rôle
Void readExternal(ObjectInput in)	Désérialiser de manière personnalisée l'objet à partir du flux passé en paramètre. Les méthodes de l'objet de type DataInput permettent de lire des valeurs primitives et la méthode readObject() de la classe ObjectInput permet de lire et créer des objets
Void writeExternal(ObjectOutput out)	Sérialiser de manière personnalisée l'état de l'objet dans le flux passé en paramètre. Les méthodes de l'objet de type DataOutput permettent d'écrire des valeurs primitives et la méthode writeObject() de la classe ObjectOutput permet d'écrire des objets

EXEMPLE : class Personne.java

```
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.Date;

public class Personne implements java.io.Externalizable {

    private String nom = "";
    private String prenom = "";
    private int taille = 0;
    private transient String codeSecret = "";
```

```
    public Personne(String nom, String prenom, int taille, String codeSecret) {
        this.nom = nom;
        this.taille = taille;
        this.prenom = prenom;
        this.codeSecret = codeSecret;
    }
    public Personne() {

    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public int getTaille() {
        return taille;
    }

    public void setTaille(int taille) {
        this.taille = taille;
    }

    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public void setCodeSecret(String codeSecret) {
        this.codeSecret = codeSecret;
    }

    public String getCodeSecret() {
        return codeSecret;
    }

    public void writeExternal(final ObjectOutput out) throws IOException {

        out.writeUTF(new StringBuilder(this.nom).reverse().toString().toUpperCase());
        out.writeUTF(new StringBuilder(this.prenom).reverse().toString().toUpperCase());
        out.writeInt(this.taille * 2019);
        out.writeUTF(new StringBuilder(Integer.toString(Integer.parseInt(this.codeSecret) *
19)).reverse().toString());
    }
```

```
    public void readExternal(final ObjectInput in ) throws IOException,
ClassNotFoundException {

        this.nom = new StringBuilder( in .readUTF()).reverse().toString().toLowerCase();
        this.prenom = new StringBuilder( in .readUTF()).reverse().toString().toLowerCase();
        this.taille = ( in .readInt()) / 2019;
        this.codeSecret = new StringBuilder(Integer.toString(Integer.parseInt( in
.readUTF()) / 19)).reverse().toString();
    }

}
```

Par défaut, la sérialisation d'un objet qui implémente cette interface ne prend en compte aucun attribut de l'objet. Seul le type de la classe est par défaut écrit dans le flux lors de la sérialisation : l'écriture de l'état de l'objet et sa restauration sont de la responsabilité du développeur.

Remarques :

1. Comme le développeur est celui qui choisit comment sérialiser l'objet, le mot clé transient devient inutile avec une classe qui implémente l'interface Externalizable.
2. Les données du flux de sérialisation sont facilement exploitables même si c'est un format binaire. Dans l'exemple, ci-dessus le code secret apparait en clair puisque c'est une chaîne de caractères.

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ xxd personne.ser
00000000: aced 0005 7372 0008 5065 7273 6f6e 6e65  ....sr..Personne
00000010: da15 d73f 0707 1005 0c00 0078 7077 2300  ...?.......xpw#.
00000020: 0644 7570 6f6e 6400 044a 6561 6e00 0000  .Dupond..Jean...
00000030: af00 0f6d 6f6e 2063 6f64 6520 7365 6372  ...mon code secr
00000040: 6574 78                                  etx
```

C'est pourquoi on a choisi faire un obscurcissement de la donnée dans notre exemple.

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP
$ xxd personne.ser
00000000: aced 0005 7372 0008 5065 7273 6f6e 6e65  ....sr..Personne
00000010: da15 d73f 0707 1005 0c00 0078 7077 1700  ...?.......xpw..
00000020: 0644 4e4f 5055 4400 044e 4145 4a00 0564  .DNOPUD..NAEJ..d
00000030: 2d00 0331 3731 78                        -..171x
```

3. Lors de la désérialisation d'un objet Externalizable, une nouvelle instance est créée en invoquant le constructeur par défaut puis la méthode readExternal() est invoquée. Une classe qui implémente l'interface Externalizable doit donc obligatoirement proposer un constructeur par défaut, sinon une exception de type InvalidClassException est levée.

La différence entre Serializable et Externalizable

Bien que ces deux interfaces soient utilisées pour sérialiser/désérialiser un objet, il existe plusieurs différences entre l'utilisation des interfaces Serializable et Externalizable :

- L'interface Serializable est un marqueur, il n'y a donc pas de méthode à implémenter par défaut. L'interface Externalizable hérite de Serializable et définit deux méthodes
- L'utilisation de Serializable implique l'utilisation du mécanisme de sérialisation par défaut (les mécanismes utilisés sont contrôlés par la plate-forme Java) alors que l'utilisation d'Externalizable implique l'utilisation de mécanismes de sérialisation personnalisés (Les mécanismes utilisés sont contrôlés par le développeur)
- L'utilisation d'Externalizable peut permettre de faciliter la compatibilité en cas de changement dans la classe mais en contrepartie chaque changement de la classe peut impliquer un changement dans le code des méthodes readExternal() et writeExternal()
- L'utilisation d'Externalizable est généralement plus performante car l'utilisation de Serializable pour une sérialisation standard implique l'utilisation de l'introspection. Ceci est particulièrement vrai pour les anciennes versions de la JVM
- Les méthodes readExternal() et writeExternal() de l'interface Externalizable annulent et remplacent les méthodes readObject() et writeObject() de l'interface Serializable si elles existent