Analyse et Programmation Orientées Objets / Java

La sérialisation

Sérialisation (1)

- Rendre un objet persistant
  - Survivre à l'exécution d'un programme
  - Transfert via un réseau
- Objet sérialisable
  - si sa classe implémente l'interface «**Serializable**»
- Interface « **Serializable** »
  - Ne contient aucune méthode
  - Sert juste à identifier les classes sérialisables
- Exception «**NotSerializableException**»
  - Tentative de sérialisation d'un objet non sérialisable

Sérialisation (2)

➔ Informations sérialisées

- le nom complet de la classe de l'objet
- les noms de ses champs, et pour chacun de ces champs, son type et sa valeur

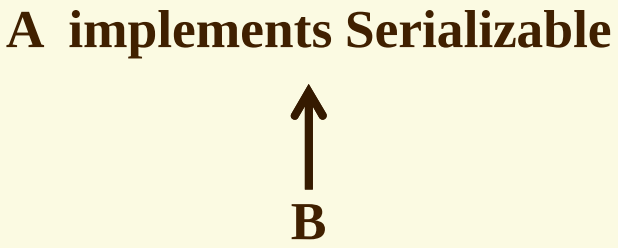
➔ Sérialisation en chaîne

- L'objet et tous les objets qu'il référence
- Les classes concernées doivent **toutes implémenter** l'interface « Serializable »

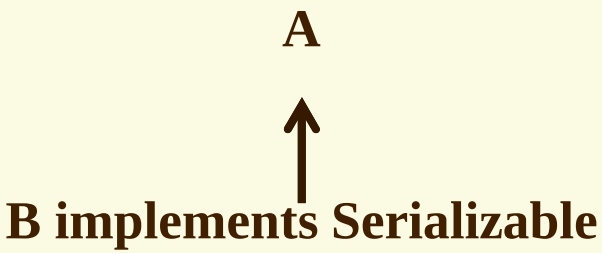
➔ Toutes les classes Collection sont sérialisables

Sérialisation et héritage (3)

➔ 1er cas
- Pas de problème



➔ 2e cas
- les attributs de A ne sont pas sérialisés
- A doit posséder un constructeur par défaut public, sinon
- « **InvalidClassException** » levée



Flux pour les objets

➔ **ObjectOutputStream** pour la sortie

- **writeObject**

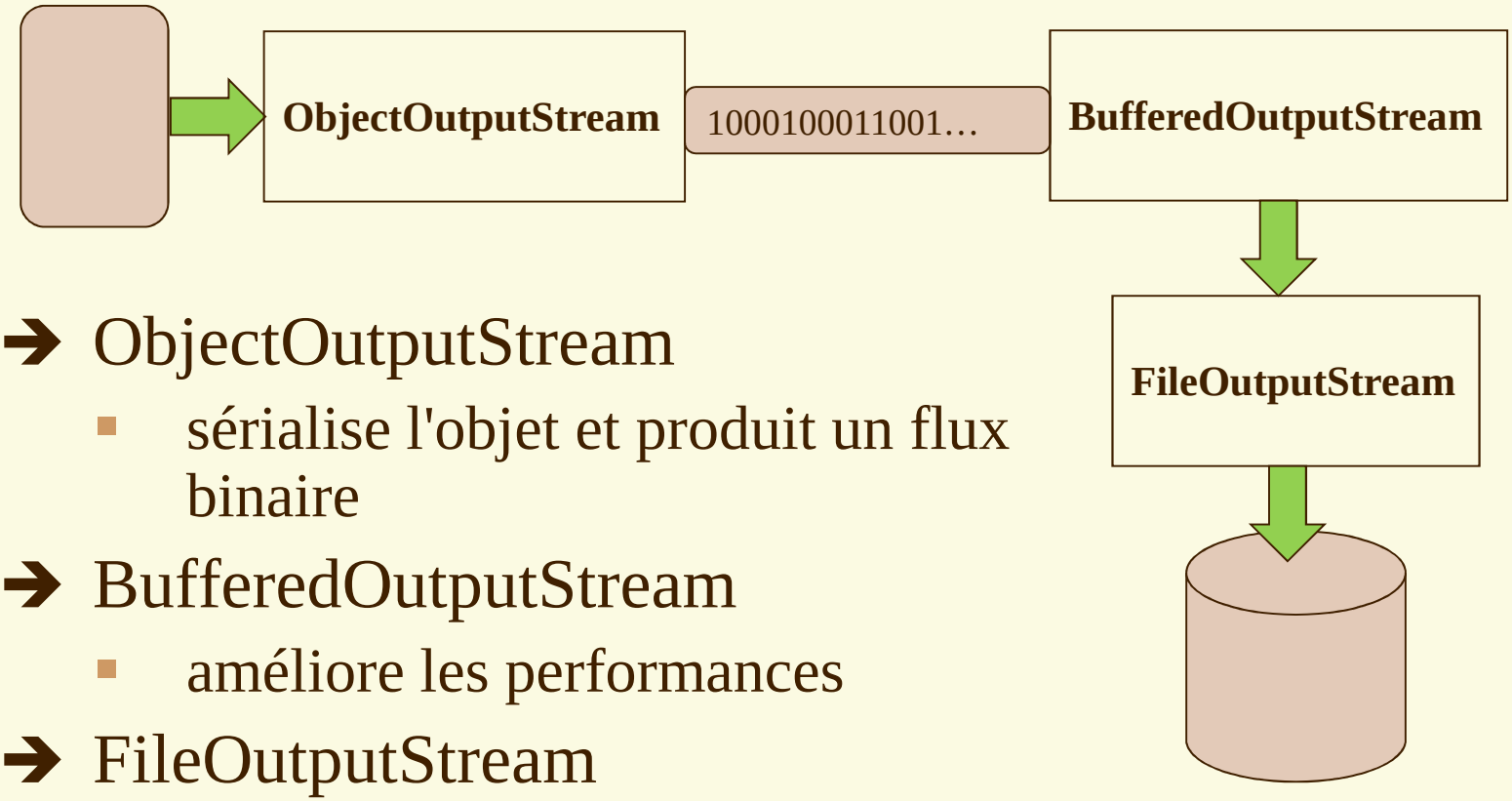
➔ **ObjectInputStream** pour l'entrée

- **readObject**

➔ Possibilité de sérialiser des éléments de type primitif

- **writeBoolean**, **writeByte**, **writeChar**, **writeDouble**, **writeInt**,…

Principe de la sortie sérialisée



- ObjectOutputStream
  - sérialise l'objet et produit un flux binaire
- BufferedOutputStream
  - améliore les performances
- FileOutputStream
  - dirige le flux binaire vers un fichier

Classe Personne

```
public class Personne implements Serializable
{
    String nom;
    int age;
    ArrayList enfants;

    public Personne(String n, int a, ArrayList e)
    {
       nom = n; age = a; enfants = e;
    }

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

Méthode save

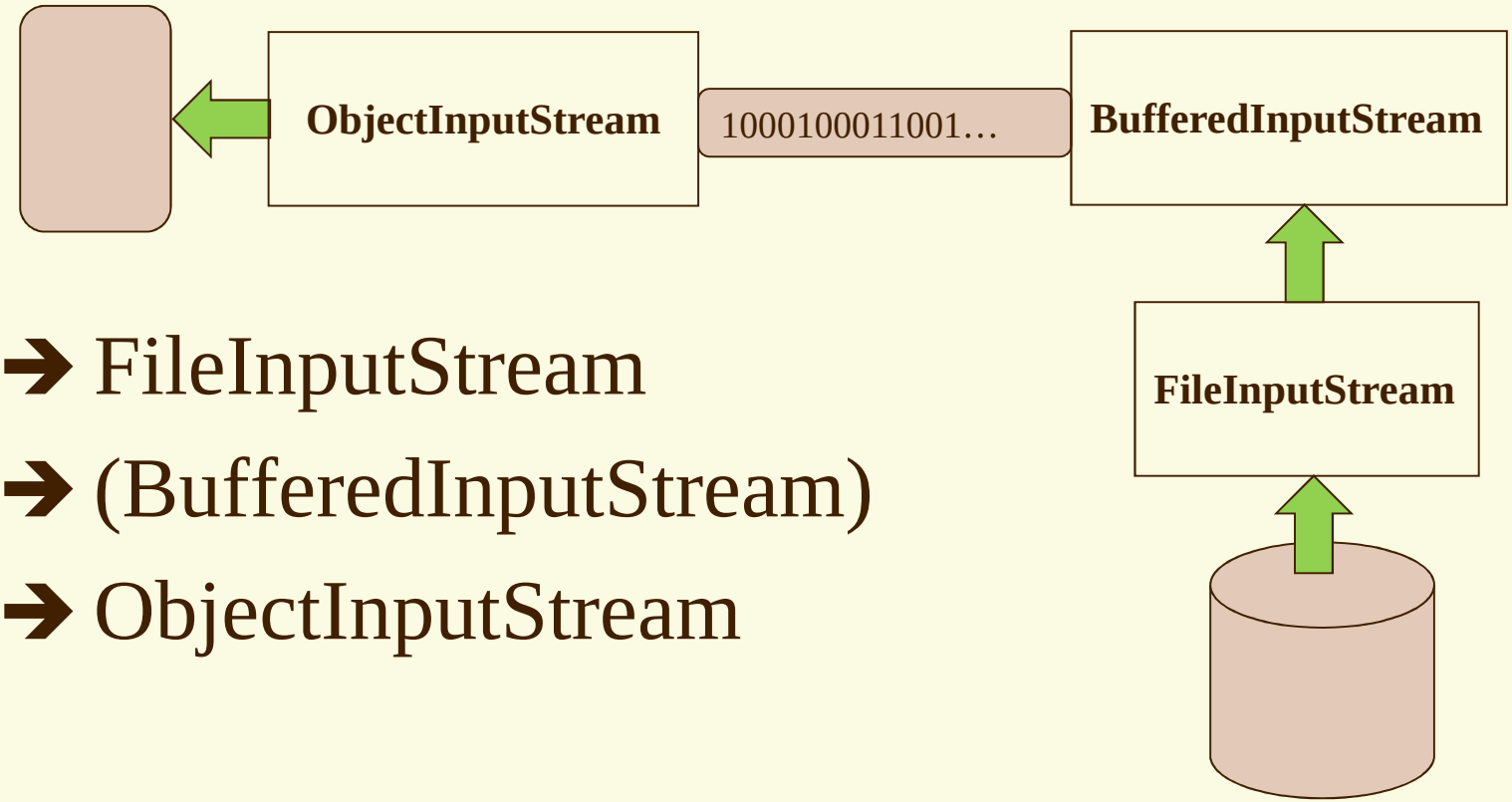
```
public static void save(String fileName, Personne p) throws
IOException
{
FileOutputStream fos = new FileOutputStream(fileName);
BufferedOutputStream bos = new BufferedOutputStream(fos);
ObjectOutputStream oos = new ObjectOutputStream(bos);

        try{ oos.writeObject(p); }

        finally{ oos.close(); }

}
```

Principe de l’entrée sérialisée



➔ FileInputStream

➔ (BufferedInputStream)

➔ ObjectInputStream

Méthode load

```
public static Personne load (String fileName) throws
IOException
{
        Personne p = null;
        ObjectInputStream ois = new ObjectInputStream(new
BufferedInputStream(new FileInputStream(fileName)));

        try {
                p = (Personne) ois.readObject();
        }
        catch(Exception e){   }
        finally { ois.close(); }
        return p;
}
```

Utilisation

```
public static void main(String[] args) throws Exception
{
        {
                ArrayList enfants = new ArrayList();
                enfants.add ("toto");
                enfants.add ("tata");
                Personne p = new Personne("titi", 40, enfants);
                Personne.save("titi.data", p);
                // ou p.save("titi.data");
        }

        {
                Personne p1 = Personne.load("titi.data");
                System.out.println(p1);
        }
}
```

titi 40 : [toto, tata]

Redéfinition des méthodes

➔ Si la sérialisation/désérialisation d'un objet requiert un traitement spécial

➔ Signatures des méthodes à redéfinir

- private void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException;

- private void writeObject(java.io.ObjectOutputStream stream) throws IOException

Attribut «**serialVersionUID**»

- Problème si la classe est modifiée entre sérialisation et désérialisation

- Cet attribut permet d'affecter un numéro de version à la classe
  - À définir dans la classe
  - **private static final long serialVersionIUD = 100L;**

- Est utilisé lors de la désérialisation afin de s'assurer que les versions des classes Java sont concordantes
  - Exception «**InvalidClassException**» levée si numéros de version différents

Modificateur « **transient** »

➔ Empêcher la sérialisation de certains attributs

- mot-clé «**transient**»
- **private transient String codeSecret;**

➔ Désérialisation «**null**» dans les attributs «**transient**»

Sérialisation personnalisée

- ➔ Implémentation de l'interface «**Externalizable**»
  - hérite de l'interface «**Serializable**»
  - Méthodes à définir
    - void **writeExternal**(ObjectOutput out) throws IOException
    - void **readExternal**(ObjectInput in) throws IOException, ClassNotFoundException
- ➔ Seule l'identité de la classe est gérée automatiquement
  - Par défaut, aucun attribut n'est sérialisé
  - Donc, mot-clé «**transient**» inutile
- ➔ Plus rapide, mais moins sécurisée (méthodes publiques)

Classe « **proxy** »

- Consiste à créer une classe plus simple
  - Ne contenant que les attributs « utiles » à la sérialisation
- Classe "**normale**"
  - **private** Object writeReplace() **throws** ObjectStreamException
- **Classe proxy**
  - **private** Object readResolve() **throws** ObjectStreamException
- http://blog.paumard.org/cours/java/chap10-entrees-sorties-serialization.html