Clonage

Conception Orientée Objet

Jean-Christophe Routier
Licence mention Informatique
Université Lille 1

Université Lille1 Sciences et Technologies | UFR IEEA Formations en Informatique de Lille 1

Rappel : référence et affectation

```
public classe UneClasse {
   public int i;
}

UneClasse reference1 = new UneClasse();
reference1.i = 5;
UneClasse reference2 = reference1;
```

`reference1` et `reference2` désignent le **même** objet

```
reference2.i = 25;
System.out.println(reference1.i); // -> 25
```

Comment dupliquer un objet ?

Cloner un objet

- intérêt : avoir deux versions d'un même objet susceptibles d'évoluer différemment

dans la classe `Object` :

- `protected Object clone() throws CloneNotSupportedException`
- Action de `Object.clone()` :
  réservation de l'espace mémoire : copie bit à bit.
- `Object.clone()` est **protected**
- la valeur de retour est `Object` ⇒ downcast

```
Value v = new Value();
Value w = (Value) v.clone(); // illegal clone() non accessible
```


Permettre le clonage

- implémenter l'interface `Cloneable`
  `public interface Cloneable`
  Sert de d'indicateur. Intérêt :
  - "typer" les objets clonables (test par `instanceof`)
  - permettre aux développeurs d'avoir des classes d'objets pas clonable (`Object.clone()` vérifie si `Cloneable`)
- déclarer `public` la méthode `clone()`
- appeler (systématiquement) `super.clone();`

Exemple

```
public class SomeValues implements Cloneable {
   String name;
   public SomeValues(String name) {
      this.name = name;
   }
   public Object clone() throws CloneNotSupportedException {
      return super.clone();
   }
}
```


Gérer son propre clonage

Ajouter des “manipulations” dans `clone()`.

```
public class SomeValues implements Cloneable {
   String name;
   public SomeValues(String name) {
      this.name = name;
   }
   public Object clone() throws CloneNotSupportedException {
      SomeValues monClone = (SomeValues) super.clone();
      monClone.name = "clone de "+this.name;
      return monClone;
   }
}
```


Attention : copie de références

Le clonage par défaut est une copie bit à bit :

⇒ en cas de clonage, c'est la référence des attributs qui est copiée.

“**shallow**” copy

Extrait API doc pour `ArrayList`:

```
clone :
public Object clone()
   Returns a shallow copy of this ArrayList instance.
      (The elements themselves are not copied.)
   Returns:
      a clone of this ArrayList instance.
   Overrides:
      clone in class Object
```



```
public class SomeValues implements Cloneable {
   protected List<Integer> al = new ArrayList<Integer>();
   protected String name;
   public SomeValues(String name) {
      this.name = name;
      al.add(Math.random()*100);
      al.add(Math.random()*100);
   }
   public void dump() {
      // affiche this.name et les éléments de this.al
   }
   public Object clone() throws CloneNotSupportedException {
      SomeValues sv = (SomeValues) super.clone();
      sv.name = "clone de "+this.name;
      return sv;
   }
}
```

```java
/** clonage "sans prise en compte" de la liste */
public class SomeValues1 extends SomeValues implements Cloneable {
   public SomeValues1(String name) { super(name); }
   public Object clone() throws CloneNotSupportedException {
      return super.clone();
   }
}
```

```java
/** clonage par copie des valeurs de la List */
class SomeValues2 extends SomeValues implements Cloneable {
  public SomeValues2(String name) { super(name); }
  public Object clone() throws CloneNotSupportedException {
    SomeValues2 sv = (SomeValues2) super.clone();
    sv.al = new ArrayList<Integer>();
    for(Integer i :  this.al) {// objets dans al doivent être clonables
      (sv.al.add((Integer) i.clone());
    }
    return sv;
  }
}
```



```java
/** mise en évidence du problème de la "shallow copy" dans le clonage */
public class TestClone {
  public static void main(String[] args) {
    SomeValues1 sv1 = new SomeValues1("sv1");
    sv1.dump(); sv2.dump();
    SomeValues1 cloneSv1 = null;
    try {
      cloneSv1 = (SomeValues1) sv1.clone();
      cloneSv2 = (SomeValues2) sv2.clone();
    catch(CloneNotSupportedException e) {}
    cloneSv1.dump();
    cloneSv2.dump();
    sv1.al.set(0,0); //modification
    sv2.al.set(0,0);
    sv1.dump(); // affichage de la modification réalisée
    sv2.dump();
    cloneSv1.dump(); // !!!  la liste du clone aussi a été modifiée !!!
    cloneSv2.dump();
  }
}
```

```
+-----------------------+
| sv1 :|16|94|          |
| sv2 :|86|75|          |
| clone de sv1 :|16|94| |
| clone de sv2 :|86|75| |
|                       |
| sv1 :|0|94|           |
| sv2 :|0|75|           |
| clone de sv1 :|0|94|  |
| clone de sv2 :|86|75| |
+-----------------------+
```


java.io

Conception Orientée Objet

Jean-Christophe Routier
Licence mention Informatique
Université Lille 1

Université Lille1 Sciences et Technologies | UFR IEEA Formations en Informatique de Lille 1

CC BY NC SA

Entrées-sorties : java.io.*

- basées sur la notion de stream
  = canal de communication entre un *écrivain* et un *lecteur*.

Classes de streams de base :

- `InputStream`/`OutputStream` classes abstraites définissant les opérations de base pour la lecture/écriture de séquence d'octets
- `Reader`/`Writer` classe abstraites définissant les opérations de base pour la lecture/écriture de séquence de données de type cacactère et supportant l'*Unicode*.

méthodes principales : `read(...)` et `write(...)`, `close()`

`java.io.IOException`

Les principaux streams

- `InputStream[Reader|Writer]` classes pour transformer des caractères en octets et vice-versa.
- `Data[Input|Output]Stream` streams spécialisés qui ajoutent la possibilité de lire/écrire des données de types de base.
- `Object[Input|Output]Stream` streams spécialisés dans la lecture/écriture d'objets Java sérialisés. Ici lecture implique reconstruction d'objet.
- `Buffered[Input|Output]Stream`/`Buffered[Reader|Writer]` ajoutent un système de tampon (buffer) pour améliorer les performances.
- `Piped[Input|Output]Stream`/`Piped[Reader|Writer]` Streams fonctionnant par paires, utilisés notamment pour la communication entre threads.
- `File[Input|Output]Stream`/`File[Reader|Writer]` Streams permettant de lire/écrire dans un fichier.

utiliser les variantes `Reader`/`Writer` pour des données caractères

"Combinaison" de streams

- Systèmes d'"enveloppes" successives, mises en cascade
  ↪ design-patter decorator
- **bufferisation** envelopper le stream à "bufferiser" dans un `Buffered***Stream` :

```
InputStream bufferedIn = new BufferedInputStream(unInputStream);
```

- **streams de données** pour accéder facilement aux données de types primitifs :

```
DataInputStream dis = new DataInputStream( System.in );
doudle d = dis.readDouble();
```

- **transformations en chaîne de caractères** `PrintWriter` méthodes `print()`, `println()`


```
import java.io.*;
public class UtilInput {
...
   public static String readString() throws java.io.IOException {
      Reader isReader = new InputStreamReader(System.in);
      BufferedReader bReader = new BufferedReader(isReader);
      return bReader.readLine();
   }
}
```

ou

```
(new BufferedReader(new InputStreamReader(System.in))).readLine();
```

ou

```
new java.util.Scanner(System.in).nextLine();
```


Fichiers

le classe `File` : informations sur fichiers + manipulations

accès séquentiel : `RandomAccessFile` (`seek()`)

Streams de fichiers

```
FileInputStream in = new FileInputStream("/tmp/exemple");
DataInputStream dataIn = new DataInputStream(in);
BufferedInputStream bufDataIn = new BufferedInputStream(dataIn);

FileOutputStream out = new FileOutputStream(unObjetFile);
FileWriter fwOut = new FileWriter(out);
```

```
import java.io.*;
...
public void copieFichierTexte(String nomSource, String nomCible) {
   File source = new File(nomSource);
   File cible = new File(nomCible);
   try{
      BufferedReader in = new BufferedReader(new FileReader(source));
      PrintWriter out = new PrintWriter(new FileWriter(cible));
      String ligne;
      while ((ligne = in.readLine()) != null) {
         out.println(ligne);
      }
      out.close();
      in.close();
   }
   catch(IOException e) { e.printStackTrace(); }
}
```


Sérialisation

Conception Orientée Objet

Jean-Christophe Routier
Licence mention Informatique
Université Lille 1

Université Lille1 Sciences et Technologies | UFR IEEA Formations en Informatique de Lille 1

cc BY NC SA

La Sérialisation

Sérialisation

La sérialisation c'est transformer un objet (en mémoire) en une suite d'octets le représentant.

$$\text{objet} \xrightarrow{\text{sérialisation}} \text{suite de bytes} \xrightarrow{\text{désérialisation}} \text{objet}$$

$$\text{objet } o \text{ ds JVM}_1 \longrightarrow \text{suite de bytes} \longrightarrow \left(\begin{array}{l}\text{fichier}\\ \text{réseau}\end{array} \longrightarrow\right) \text{objet } o \text{ ds JVM}_2$$

- y compris entre JVM d'OS différents... (application : RMI)
- Permet la persistance entre sessions

Mise en œuvre, sérialisation par défaut

La classe de l'objet doit implémenter `java.io.Serializable`

- Sérialisation `ObjectOutputStream`/ `writeObject()`
  la sérialisation réalise la cloture transitive des dépendances sur l'objet sérialisé (graphe d'objets)
  si o, à sérialiser, possède la référence sur des objets o_1, o_2, ils seront eux aussi sérialisés (et donc y compris les objets qu'ils référencent : clôture transitive du graphe de dépendances des objets)
- Désérialisation `ObjectInputStream`/ `readObject()`
  (évidemment le "`.class`" doit être accessible à la JVM d'accueil)

writeObject()

java.io.ObjectOutputStream

`public final void writeObject(Object obj) throws IOException`

Write the specified object to the `ObjectOutputStream`. The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all of its supertypes are written. Default serialization for a class can be overridden using the `writeObject` and the `readObject` methods. Objects referenced by this object are written transitively so that a complete equivalent graph of objects can be reconstructed by an `ObjectInputStream`.(...)

Throws :

- *InvalidClassException* - Something is wrong with a class used by serialization.
- *NotSerializableException* - Some object to be serialized does not implement the `java.io.Serializable` interface.
- *IOException* - Any exception thrown by the underlying OutputStream.

readObject()

java.io.ObjectInputStream

`public final Object readObject()`

`throws OptionalDataException, ClassNotFoundException, IOException`

Read an object from the `ObjectInputStream`. The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all of its supertypes are read. Default deserializing for a class can be overriden using the `writeObject` and `readObject` methods. Objects referenced by this object are read transitively so that a complete equivalent graph of objects is reconstructed by `readObject`. The root object is completly restored when all of its fields and the objects it references are completely restored. (...)

Throws:

- *ClassNotFoundException* Class of a serialized object cannot be found.
- *InvalidClassException* Something is wrong with a class used by serialization.
- *StreamCorruptedException* Control information in the stream is inconsistent.
- *OptionalDataException* Primitive data was found in the stream instead of objects.
- *IOException* Any of the usual Input/Output related exceptions.


```
package essais.serialisable;
public class SerialisableData implements java.io.Serializable {
   private int i;
   private SerialisableData sd = null;
   public SerialisableData(int i, int j) {
      this(i);
      sd = new SerialisableData(j);
   }
   public SerialisableData(int i) {
      this.i = i;
   }
   public String toString() {
      return ""+i+" et "+(sd==null ?  "" :  sd.toString());
   }
} // SerialisableData
```



```
import java.io.*;
public class TestSerialisation {
  public static void main(String[] args)
      throws IOException, ClassNotFoundException {
    SerialisableData sd1 = new SerialisableData(1,2);
    SerialisableData sd2 = new SerialisableData(5,8);
    ObjectOutputStream out = new ObjectOutputStream(
                             new FileOutputStream("sdata.dat"));
    out.writeObject(sd1); out.writeObject(sd2);
    out.close(); }
} // TestSerialisation                          +----------------------
  ===== DANS UN AUTRE main INDEPENDANT ===      | sd1 :  1 et 2
public class TestDeSerialisation {              | sd2 :  5 et 8
  public static void main(String[] args)        +----------------------
      throws IOException, ClassNotFoundException {
    ObjectInputStream in = new ObjectInputStream(
                             new FileInputStream("sdata.dat"));
    SerialisableData sd1 = (SerialisableData) in.readObject();
    SerialisableData sd2 = (SerialisableData) in.readObject();
    in.close();
    System.out.println(\n sd1 :"+sd1+"\n sd2 :"+sd2+"\n");
  }
} // TestDeSerialisation
```

transient

- le modificateur d'attribut `transient` spécifie que l'attribut ne doit pas être sérialisé (ni donc restauré)
- valeur `null` à la désérialisation
- nécessité de prise en charge par le programme de l'attribut délaissé

Personnalisation de la sérialisation

ajouter à une classe `Serializable` les méthodes :

- `private void writeObject(ObjectOutputStream out) throws IOException`
- `private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException`

- `private` !
- appelées automatiquement par `ObjectOutputStream.writeObject()` et `ObjectInputStream.readObject()`
- `ObjectOutputStream.defaultWriteObject()` et `ObjectInputStream.defaultReadObject()`
  ↪ gère tous les attributs ni `static` ni `transient`


```
import java.io.*;
public class SerialisableData2 implements java.io.Serializable {
   private int i;
   private transient String code;
   private SerialisableData2 sd;
   public SerialisableData2(int i, int j) {
      this(i); sd = new SerialisableData2(j);
   }
   public SerialisableData2(int i) { this.i = i; code = ""+i; }
   private void writeObject(ObjectOutputStream out) throws IOException {
      out.defaultWriteObject();
      out.writeObject(crypte(code));
   }
   private void readObject(ObjectInputStream in)
         throws IOException, ClassNotFoundException {
      in.defaultReadObject();
      code = decrypte((String)in.readObject());
   }
   private String crypte(String code) { return code+"CRYPTE"; }
   private String decrypte(String code){
      return code.substring(0,code.length()-6);
   }
   public String toString() {
      return ""+i+" code :"+code+(sd==null ?  "" :  " et "+sd);
   }
} // SerialisableData2
```


Cohérence de la sérialisation?

Pb :

- Deux objets o_1 et o_2 partagent une référence sur l'objet o_3
- On sérialise o_1 et o_2 (qui chacun sérialise o_3) via le même stream
- Qu'en est il lors de leur désérialisation vis-à-vis de o_3 ?

Cohérence de la sérialisation?

Pb :

- Deux objets o_1 et o_2 partagent une référence sur l'objet o_3
- On sérialise o_1 et o_2 (qui chacun sérialise o_3) via le même stream
- Qu'en est il lors de leur désérialisation vis-à-vis de o_3 ?

- Les dépendances d'objets sont conservées !


```
public class SerialisableData3 implements java.io.Serializable {
   private int i;
   public SerialisableData3 sd = null;
   public SerialisableData3(int i, SerialisableData3 sd) {
      this(i); this.sd = sd;
   }
   public SerialisableData3(int i) { this.i=i; }
   public boolean equals(SerialisableData3 sd3) {
      return this.i == sd3.i &&
         ((this.sd ==null && sd3.sd == null) || this.sd.equals(sd3.sd));
   }
   public boolean memeSd(SerialisableData3 sd3) {
      return this.sd == sd3.sd;
   }
} // SerialisableData3
...
SerialisableData3 sd = new SerialisableData3(12);
SerialisableData3 sd1 = new SerialisableData3(1,sd);
SerialisableData3 sd2 = new SerialisableData3(5,sd);

...  Sérialisation
...  puis ouverture d'un stream de lecture "in"...

SerialisableData3 sd1 = (SerialisableData3) in.readObject();
SerialisableData3 sd2 = (SerialisableData3) in.readObject();
System.out.println("sd1 et sd2, meme Sd ?  "+ sd1.memeSd(sd2));
                                          +------------------------------
                                          | sd1 et sd2, meme Sd ?  true
                                          +------------------------------
```