Programmation Par Objets

Fichiers et Streams

package java.io

Fichiers et Streams

- Les Stream's sont des flots de données, abstractions de différentes sources dont les blocs mémoires (`ByteArray`), les canaux internet (`URLConnection`), les fichiers (vus ici).
- On distingue:
  - les flots de caractères : `Reader/Writer`
    - fichiers de caractères : `FileReader/FileWriter`
    - formatage : `StreamTokenizer` (`Scanner` en 5.0) et `PrintWriter`
  - les flots binaires (de bytes) : `InputStream/OutputStream`
    - fichiers binaires `FileInputStream/FileOutputStream`
    - formatage de données: `DataInputStream/OutputStream`
    - fichiers d'objets : "sérialisation"
- Ce sont des flots séquentiels, seul flot à accès direct : `RandomAccessFile`

Fichiers de caractères

- **FileReader**

```
Reader // racine abstract des flots de caractères
   InputStreamReader // pont bytes -> char selon Charset
            (ASCII, ISO, UTF,...)
     FileReader // sur fichier quelconque
```

- Principal constructeur

```
public FileReader(String fileName)
 throws FileNotFoundException
```

- Principale méthode

```
// lecture de caractère, renvoie -1 si fin de fichier
public int read() throws IOException
```

- Pour bufferiser et disposer de la lecture de ligne (readLine), wrapper dans :

```
public class BufferedReader extends Reader
 public BufferedReader(Reader in) // constructeur
```


Fichiers de caractères

- **FileWriter**

```
Writer
   OutputStreamWriter // pont char -> bytes selon Charset
     FileWriter
```

- Principaux constructeurs

```
public FileWriter(String fileName, boolean append)
   throws IOException
public FileWriter(String fileName)
   throws IOException
```

- Principale méthode

```
public void write(int c) throws IOException
```

- Bufferisation par:

```
public class BufferedOutputStream
   extends FilterOutputStream
```

- **Exemple**: commande copie de fichiers textes

```
bash> java copie source.txt dest.txt
```

Fichiers de caractères : exemple

```
import java.io.*;
public class copie {
   public static void main(String argv[]) throws IOException {
        FileReader in = null;
        FileWriter out = null;
        if(argv.length < 2) {
         System.err.println("copie <source> <dest>");
         System.exit(1);
        }
        try {
        in = new FileReader(argv[0]);
        out = new  FileWriter(argv[1]);
        } catch (IOException ex) {
         System.err.println("erreur d'acces a "+argv[0]+"/"+argv[1]);
         System.exit(2);
        }
        int c;
        while ((c=in.read())!=-1) out.write(c);
        in.close(); out.close();
}...}
```


Formatage textes

- Sortie formatée : **`PrintWriter`** (cf. `System.out`)

```
public class PrintWriter
 PrintWriter(File file) //constructeur wrappeur
 print(x) et println(x) sur tout type
 printf(String format, Object… args) //5.0
```

- Entrée formatée
  - 5.0 : **`java.util.Scanner`**

```
public class Scanner implements Iterator<String> {
 public String next()
 public boolean hasNext()
 public int nextInt()
 public double nextDouble()
...}
```

  - avant 5.0 : utiliser **`java.io.StreamTokenizer`**

Fichiers binaires

- Les classes `FileInputStream` et `FileOutputStream` offrent les primitives de lecture/écriture de byte.
- Les classes wrappers `DataInputStream` et `DataOutputStream` permettent la lecture/écriture des types primitifs (`int`, `double`, `boolean`, `char`, ...) dans leur représentation en byte.

```
public class FileInputStream extends InputStream
   FileInputStream(String name)
   public int read() throws IOException
    //8bits de poids faibles, -1 si fin de flot

public class FileOutputStream extends OutputStream
   FileOutputStream(String name)
   FileOutputStream(String name, boolean append)
   void write(int b) //principale methode
```


Fichiers binaires

```
public class DataInputStream extends FilterInputStream
   DataInputStream(InputStream in) // sur FileInputStream
   public final int readInt()
        throws IOException // EOFException si fin
   public final double readDouble()
        throws IOException //idem
   public final boolean readBoolean()
        throws IOException //idem

public class DataOutputStream extends FilterOutputStream
   DataOutputStream(OutputStream out) //sur FileOutputStream
   public final void writeInt(int v) throws IOException
   public final void writeDouble(double v) throws IOException
   public final void writeBoolean(boolean v)
        throws IOException
```

Fichiers binaires : exemple

1) Conversion fichier texte de reels (`argv[0]`) en fichier binaire (`argv[1]`)

- en 5.0 : Scanner

```
public class convertir {
 public static void main(String argv[])throws IOException{
  Scanner in = new Scanner(new FileReader(argv[0]));
  DataOutputStream out
   = new DataOutputStream(new FileOutputStream (argv[1]));
  while (in.hasNext()) out.writeDouble(in.nextDouble());
  in.close()
  out.close();
 }
}
```


Fichiers binaires : exemple

1bis) Conversion fichier texte de reels (`argv[0]`) en fichier binaire (`argv[1]`)

- avant 5.0 : StreamTokenizer

```
public class convertir {
 public static void main(String argv[])throws IOException{
  StreamTokenizer tok
   = new StreamTokenizer(new FileReader(argv[0]));
  DataOutputStream out
   = new DataOutputStream(new FileOutputStream (argv[1]));
  while (tok.nextToken()!=StreamTokenizer.TT_EOF)
   out.writeDouble(tok.nval);
  in.close();
  out.close();
 }
}
```

Fichiers binaires : exemple (suite)

2) ranger les réels (binaires) de `argv[0]` > `argv[1]` dans `argv[2]`

```
public class filtrer {
 public static void main(String argv[]) throws IOException {
 double x, seuil=(Double.valueOf(argv[1])).doubleValue();
 DataInputStream in
  = new DataInputStream(new FileInputStream(argv[0]));
 DataOutputStream out
  = new DataOutputStream(new FileOutputStream (argv[2]));
 try {
  x=in.readDouble();
  while (true) {
   if (x>seuil) out.writeDouble(x);
   x=in.readDouble();
  }} catch (EOFException ex) {} // fin de lecture
 in.close();
 out.close();
}}
```


Fichiers d'objets : la sérialisation

- La sauvegarde externe d'un objet nécessite de générer une représentation de son état sous la forme d'une séquence de bytes suffisante en vue de sa restitution:
  - identification de sa classe,
  - état: valeurs de ses variables d'instances (en particulier références entre objets à travers des « identifiants »),
  - identifiant de l'objet (pour recoller les références)
- Le mécanisme de génération s'appelle la «sérialisation».
- Ce mécanisme est complexe, ne serait-ce que parce que l'objet peut référencer d'autres objets au travers de ses v.i., qu'il faut sérialiser également...
- La sérialisation est un mécanisme général ne s'appliquant pas seulement à la sauvegarde d'objets mais aussi au transfert d'objets sur le réseau (`java.net, java.rmi`).

Sérialisation

- Ce mécanisme est offert par défaut dans la classe `Object` mais n'est applicable que si la classe de l'objet implémente l'interface (vide) `java.io.Serializable`.
- Pour rendre des objets sérialisables, il suffit donc de déclarer:
  `class <Classe> implements Serializable {...}`
- Les flots `ObjectInputStream` et `ObjectOutputStream` permettent de manipuler des fichiers binaires (`FileInputStream`/`FileOutputStream`) d'objets sérialisés par les méthodes `readObject/writeObject`.
- *Remarque* : comme les `DataInputStream` et `DataOutputStream`, ces classes offrent des méthodes de lecture/écriture de types de base (`readInt/writeInt`, `readDouble/writeDouble`) et peuvent donc mixer objets et valeurs types primitifs dans leur représentation binaire.

Sérialisation

```
public class ObjectOutputStream extends OutputStream
  public ObjectOutputStream(OutputStream out)
  //constructeur
  public final void writeObject(Object obj)
      throws NotSerializableException
```

- `writeObject(obj)` sérialise `obj` sur le flot
- l'exception `NotSerializableException` est provoquée si la sérialisation rencontre un objet non sérialisable (dont la classe n'implémente pas `Serializable`).

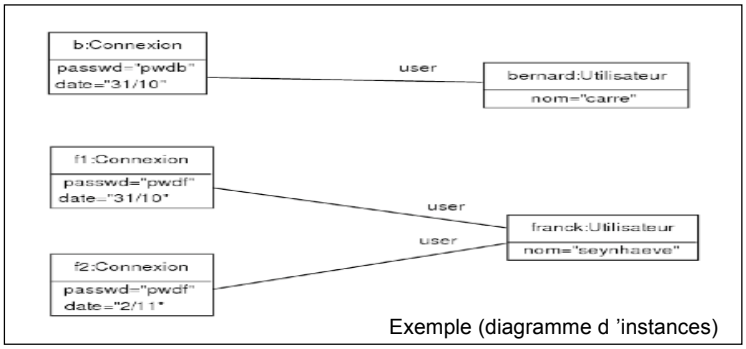
Sérialisation

```
public class ObjectInputStream extends InputStream
  public ObjectInputStream(InputStream in)
      //constructeur
  public final Object readObject()
      throws ClassNotFoundException
```

- `readObject()` lit un objet du flot.
- si la classe de l'objet n'est pas connue de l'environnement, l'exception `ClassNotFoundException` est provoquée.
- Attention: `readObject()` rendant un `Object`, il est nécessaire de **downcaster** la lecture.

Sérialisation : exemple

Trace de connexions (logs) d'utilisateurs sur un système (diagramme de classes)



Exemple (diagramme d 'instances)

Sérialisation : exemple (suite)

```
public class Utilisateur implements Serializable {
   private String nom;
   public Utilisateur(String nom) {this.nom=nom;}
   public String toString() {
    return (super.toString()+" "+nom); //super=>no d'objet
}}

public class Connexion implements Serializable {
 private Utilisateur user;
 private String date;
 private transient String passwd; // ne sera pas sauvegardé
 public Connexion (Utilisateur u, String passwd, String d) {
   this.user=u; this.passwd=passwd; this.date=d;}
 public String toString() {
   return("user:"+user+" date:"+date+" passwd:"+passwd+"\n");
}}
```


Sérialisation : exemple (suite)

```
public class Logs {
 public static void main (String argv[])
  throws IOException, ClassNotFoundException {
   Utilisateur bernard=new Utilisateur("carre"),
    franck=new Utilisateur("seynhaeve");
   Connexion b = new Connexion(bernard,"pwdb","31/10"),
    f1 = new Connexion(franck,"pwdf","31/10"),
    f2 = new Connexion(franck,"pwdf","2/11");
   System.out.print("avant serialisation:\n"+b+f1+f2);
//...

/* resultat
avant serialisation:
user:Utilisateur@111f71 carre date:31/10 passwd:pwdb
user:Utilisateur@273d3c seynhaeve date:31/10 passwd:pwdf
user:Utilisateur@273d3c seynhaeve date:2/11 passwd:pwdf
*/
```

Sérialisation : exemple (suite)

```
// ... sauvegarde
 ObjectOutputStream out
  = new ObjectOutputStream(new FileOutputStream("logs.bin"));
 out.writeObject(b);
 out.writeObject(f1); out.writeObject(f2); out.close();
// relecture
 ObjectInputStream in
  = new ObjectInputStream(new FileInputStream("logs.bin"));
 b=(Connexion)in.readObject();
 f1=(Connexion)in.readObject();
 f2=(Connexion)in.readObject();
 System.out.print("apres serialisation:\n"+b+f1+f2);
}}

/* resultat : noter le recolement de l'objet Utilisateur@2125f0
apres serialisation: */
user:Utilisateur@e3e60 carre date:31/10 passwd:null
user:Utilisateur@2125f0 seynhaeve date:31/10 passwd:null
user:Utilisateur@2125f0 seynhaeve date:2/11 passwd:null
```


Sérialisation : quelques règles

- Si une classe est sérialisable, toutes ses sous-classes le sont (par héritage de l'interface **`Serializable`**).
- On peut écarter une information de la sérialisation par le modifier de variable **`transient`** (sa valeur ne sera pas sauvegardée).
- Il est possible de définir ses propres opérations de sérialisation selon un protocole précis (en implémentant la sous-interface **`Externalizable`** de **`Serializable`**).
- Le mécanisme de sérialisation traite le cas de graphes d'objets partagés (cf. exemple) éventuellement cycliques (par marquage).
- Le partage d'objets ne peut être restitué qu'au sein d'un même stream (même espace mémoire). Il n'est pas possible de "recoler" des objets partagés issus de streams (fichiers) distincts.
- Beaucoup de classes Java sont sérialisables, en particulier les **tableaux et les collections**. Elles sérialisent automatiquement leurs éléments (si leur classe est déclarée sérialisable).

Sérialisation de collections : exemple

- Exemple : “backup” de la bibliothèque

```
public class Ouvrage implements Serializable ...
public class Bibliotheque {
 protected TreeMap<String,Ouvrage> ouvrages
  = new TreeMap<String,Ouvrage> ();
 public void save(String backupName) throws IOException {
  ObjectOutputStream out
   = new ObjectOutputStream (
    new FileOutputStream (backupName));
  out.writeObject(ouvrages);
  }
  public void load(String backupName)
        throws IOException, ClassNotFoundException {
   ObjectInputStream in
    = new ObjectInputStream (
     new FileInputStream (backupName));
   ouvrages=(TreeMap<String,Ouvrage>)in.readObject();
    //downcast
  }
```


Sérialisation de collections : exemple

```
// Application
   Bibliotheque bib = new Bibliotheque();
   bib.add(“I101”,new Ouvrage(“C”,”Kernighan”));
   bib.add(“I345”,new Ouvrage(“Java”,”Eckel”));
   bib.save(“base.bin”);
   bib.add(“B300”,new Ouvrage(“Le Chat”, “Gelluck”));
   System.out.println(“Apres ajout:”);
   bib.listing();
   bib.load(“base.bin”); // restitution de la sauvegarde
   System.out.println(“avant ajout (sauvegarde):”);
   bib.listing();
   }
}
// resultats
Apres ajout:
B300:Le Chat Gelluck
I101:C Kernighan
I345:Java Eckel
Avant ajout (sauvegarde):
I101:C Kernighan
I345:Java Eckel
```