

SÉRIALISATION D'OBJETS

package java.io

Walter Rudametkin
Maître de Conférences
Bureau F011
Walter.Rudametkin@polytech-lille.fr

Fichiers et Streams

- Les Streams sont des flots de données, abstractions de différentes sources dont les blocs mémoires (ByteArray), les canaux internet (URLConnection), les fichiers.
- Streams de caractères: Reader/Writer
 - Dont fichiers: FileReader/FileWriter
 - Formatage par « wrapping »:
 - En lecture: Scanner (en 5.0)
 - En écriture par PrintWriter
- Streams binaires (de bytes): InputStream/OutputStream
 - Dont fichiers: FileInputStream/FileOutputStream
 - Formatage par « wrapping »:
 - En lecture: DataInputStream
 - En écriture: DataOutputStream

Exemple

1. Fichier texte de reels (argv[0]) => fichier binaire (argv[1])

```
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();
    }
}
```

Exemple (suite)

2. Ranger les réels de argv[0] > argv[1] dans argv[2]

```
public class filtrer {
    public static void main(String argv[] throws IOException {
        double x, seuil=Double.parseDouble(argv[1]);

        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();
    }
}
```

Sérialisation d'objets

- 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 et récursivement (graphes d'objets)...
- La sérialisation est un mécanisme général ne s'appliquant pas seulement à la sauvegarde d'objets sur fichiers mais aussi au transfert d'objets sur le réseau (java.net, java.rmi, JEE).

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 java.io.Serializable.
- Pour rendre des objets sérialisables, il suffit donc de déclarer:


```
class <Classe> implements Serializable {...}
```
- Les « wrappers » (binaires) ObjectInputStream et ObjectOutputStream permettent de manipuler des fichiers binaires (FileInputStream/FileOutputStream) d'objets sérialisés par les méthodes readObject/writeObject.
- *Remarque*: comme les « wrappers » 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 {

    //constructeur "wrapper"
    public ObjectOutputStream(OutputStream out) {...}

    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).
- <https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html>

Sérialisation

```
public class ObjectInputStream extends InputStream {

    //constructeur "wrapper"
    public ObjectInputStream(InputStream in) {...}

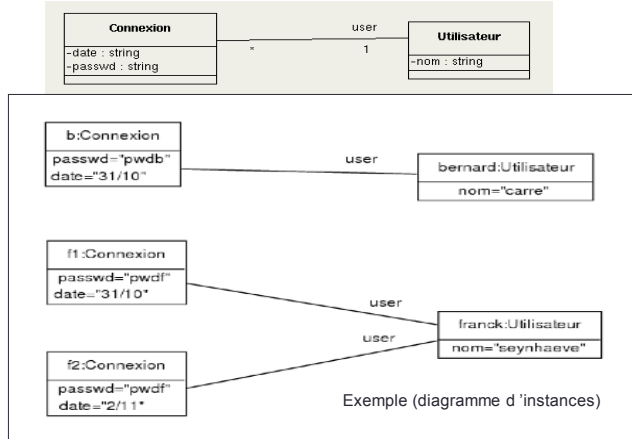
    public final Object readObject() throws
        ClassNotFoundException {...}

    ...
}
```

- 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.
- <https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html>

Exemple

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



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");
    }
}
```

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 sur la console */
        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
    }
}
```

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 modificateur de variable `transient` (sa valeur ne sera pas sauvegardée).
- 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 « backup » 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

```
public class Ouvrage implements Serializable ...

public class Bibliotheque {
    protected Map<String,Ouvrage> ouvrages = new
        TreeMap<String,Ouvrage> ();

    public void save(String backupName) throws IOException {
        ObjectOutputStream out = new ObjectOutputStream (
            new FileOutputStream (backupName));
        out.writeObject(ouvrages);
        out.close();
    }

    public void load(String backupName) throws
        IOException,ClassNotFoundException {
        ObjectInputStream in = new ObjectInputStream (
            ew FileInputStream (backupName));
        ouvrages=(Map<String,Ouvrage>)in.readObject();//downcast
        in.close();
    }
    //...
}
```

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("Après ajout:");
bib.listing();
bib.load("base.bin"); // restitution de la sauvegarde
System.out.println("avant ajout (sauvegarde):");
bib.listing();
}

// resultats
Après ajout:
B300:Le Chat Gelluck
I101:C Kernighan
I345:Java Eckel
avant ajout (sauvegarde):
I101:C Kernighan
I345:Java Eckel
```