

## Feuille de travaux dirigés n° 8

### Chapitre 8 - Flots d'entrées/sorties



Dans cette série d'exercices, nous proposons un rapide tour d'horizon des Entrées/sorties et des flux, relativement liées aux collections et aux itérateurs, vus dans le chapitre 6.

Voici une suggestion d'exercices et d'activités à faire dans l'ordre des sections pour ce chapitre 8 :

1. Entrées-sorties standard ou fichier : section 1.
2. Gestion de fichier : section 2.
3. Flux d'entrées-sorties : section 3.
4. Entrées-sorties binaires : section 4.

Les exercices peuvent être implantés en Java, C++ ou Smalltalk. Il n'y a pas de consigne générale à ce sujet.

## 1 Entrées-sorties standard ou fichier

### Exercice 8.1 (Entrées-sorties standard)

Ecrire un programme qui lit un flot de donnée en entrée et restitue sur un flot en sortie.

- a) Entrée standard / sortie standard
- b) Lecture fichier texte / sortie standard
- c) Lecture fichier texte / sortie fichier texte
- d) Reprendre les questions en formattant la sortie sur 30 caractères / ligne.

### Exercice 8.2 (Notes)

Reprendre le contexte de l'exercice sur les notes dans le chapitre 6.

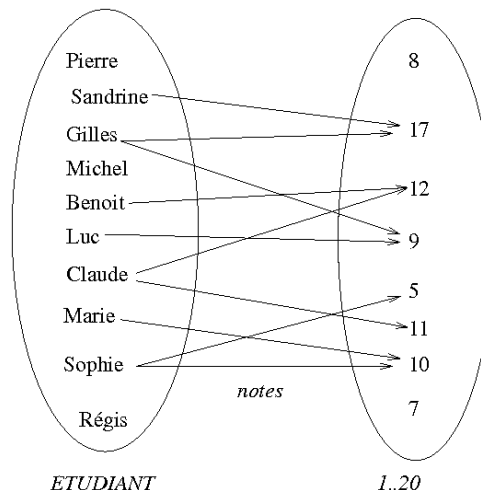


Figure 1 : *Relation entre ensembles*

- Ecrire les notes par étudiant dans la console.
- Reprendre la question a) mais écrire dans la console erreur le cas des étudiants n'ayant pas la moyenne (Java, C++).
- Lire les notes des étudiants dans un fichier texte, avec analyse lexicale des valeurs (entiers ou réels).
- Ecrire les notes des étudiants dans un fichier texte.

### Exercice 8.3 (Hôpital)

On reprendre l'exercice sur l'hôpital dans le TD n° 6.

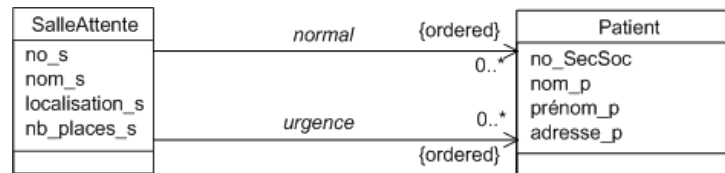


Figure 2 : *UML - Hôpital orienté*

- Reprendre le programme pour lire les données des patients et des salles au clavier ou dans un fichier.
- Ajouter de quoi analyser et traiter les événements stockés sous forme de commandes dans un fichier texte :

```
normal no-patient1
normal no-patient2
sortie no-salle1
urgent no-patient3
sortie no-salle2
sortie no-salle3
...
```

- Ecrire le programme de simulation qui restitue (au choix) sur la console ou console-err ou dans des fichiers, les traces d'exécution.

### Exercice 8.4 (Pluviométrie)

Le but du programme est de calculer des niveaux pluviométriques annuels dans des villes à partir de relevés mensuels saisis au préalable. Un relevé (une fiche) contient les informations suivantes : lieu, mois, hauteur. Lors de la saisie on contrôle la validité des numéros de mois et des hauteurs. Le nombre de relevés n'est pas fixé à l'avance. Il s'agit donc de faire la somme par mois des hauteurs de précipitations. Les résultats sont affichés à l'écran.

- Ecrire un programme qui permette la saisie des fiches et la restitution des sommes dans l'E/S standard.
- Même chose avec des fichiers texte.
- Même chose en levant et traitant des exceptions.

## 2 Gestion de fichiers

### Exercice 8.5 (Information de fichiers)

Ecrire un programme qui permette d'afficher les diverses informations sur un répertoire donné en paramètre (nom, localisation, droits, taille, dates...).

### Exercice 8.6 (Gestion de fichiers)

Réaliser une interface avec le système de gestion de fichier qui permette de définir une méthode pour chaque commande standard du gestionnaire de fichiers (inspiré d'Unix) :

- cd : change le répertoire courant
- cp : copie un fichier
- ls : affiche la liste des fichiers d'un répertoire (par défaut du répertoire courant)
- type : affiche le type de fichier.

- isReadable : rend vrai si accessible en lecture
- isWritable : rend vrai si accessible en écriture
- mkdir : crée un répertoire
- pwd : affiche le chemin du répertoire courant
- rm : supprime un fichier
- rmdir : supprime un répertoire

#### **Exercice 8.7 (Codage de fichiers)**

Ecrire un programme qui permette de lire un fichier ISO et de l'écrire en UTF-8.

#### **Exercice 8.8 (Archive)**

Ecrire un programme qui permette d'archiver/séarchiver au format ZIP.

### **3 Flux d'entrées-sorties**

#### **Exercice 8.9 (Flux d'entrées-sorties générique)**

Reprendre l'exercice 8.1 en définissant une seule méthode paramétrée par les flots entrants et sortants.

#### **Exercice 8.10 (Pipe)**

Reprendre l'exercice de la file du TD n° 6.

- Implanter la file comme un flot de données en entrée et sortie.
- Connecter deux files en mode pipeline : la sortie de la première est l'entrée de la seconde.

### **4 Entrées-sorties binaires**

#### **Exercice 8.11 (Entrées-sorties binaires/Parking)**

Reprendre l'exercice sur le parking du TD n° 2.

- Concevoir et implanter une méthode qui puisse stocker l'état du parking (avec ses véhicules) à un instant donné.
- Concevoir et implanter une méthode qui puisse faire l'opération inverse, de restauration du parking (avec ses véhicules) à partir d'un fichier donné en paramètre.

#### **Exercice 8.12 (Entrées-sorties binaires/Hôpital)**

Reprendre l'exercice sur les salles d'attente de l'exercice 8.3.

- Concevoir et implanter une méthode qui puisse stocker l'état d'une salle d'attente (avec ses patients) à un instant donné.
- Concevoir et implanter une méthode qui puisse faire l'opération inverse, de restauration de la salle (avec ses patients) à partir d'un fichier donné en paramètre.



## Feuille de travaux dirigés n° 8 Chapitre 8 - Flots d'entrées/sorties

### — CORRECTION —



Nous proposons principalement des solutions en Java.

#### Corrigé exercice 8.1 (Entrées-sorties standard)

Non corrigé puisque c'est une activité de découverte. Nous donnons simplement un exemple :

```
package td8;
```

```
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class IficOfic {
    public static void main(String[] args) throws IOException {
        int c;
        // lecture au clavier des noms de fichiers
        StringTokenizer ligne;
        String fentree, fsortie;
        BufferedReader stdin = new BufferedReader(new InputStreamReader(
            System.in));
        // déclare le tampon de lecture (pour types primitifs)
        try {
            System.out
                .print("donner les noms de fichiers 'fentree.txt fsortie.txt' : ");
            // line = stdin.readLine().split("(")*(")");
            // lecture et extraction des deux valeurs séparées par un blanc
            ligne = new StringTokenizer(stdin.readLine());
            fentree = ligne.nextToken();
            fsortie = ligne.nextToken();
            // utiliser parse si autre chose que String
            FileReader entrée = new FileReader(fentree);
            FileWriter sortie = new FileWriter(fsortie);
            while ((c = entrée.read()) != -1)
                sortie.write(c);
            entrée.close();
            sortie.close();
        } catch (IOException e) {
            System.err.println("IOException thrown " + e.toString());
            // return false;
        }
    }
}
```

Sous Eclipse, pour éviter des noms de répertoire, placer le fichier à la racine du projet.

#### Corrigé exercice 8.2 (Notes)

Non corrigé.

### Corrigé exercice 8.3 (Hôpital)

Non corrigé.

### Corrigé exercice 8.4 (Pluviométrie)

Voir la fiche `fiche1_ex.pdf` du TD n° 1, pour une solution en Java et Smalltalk.

### Corrigé exercice 8.5 (Information de fichiers)

Exemmmple trouvé sur <http://www.siteduzero.com/tutoriel-3-65524-les-flux-d-entrees-sorties-1-2.html>.

```
// Package à importer afin d'utiliser l'objet File
import java.io. File ;

public class Main {
    public static void main(String[] args) {
        //Création de l'objet File
        File f = new File("test.txt");
        System.out.println("Chemin absolu du fichier : " + f.getAbsolutePath());
        System.out.println("Nom du fichier : " + f.getName());
        System.out.println("Est-ce qu'il existe ? " + f.exists());
        System.out.println("Est-ce un répertoire ? " + f.isDirectory());
        System.out.println("Est-ce un fichier ? " + f.isFile());

        System.out.println("Affichage des lecteurs racines du PC : ");
        for(File file : f.listRoots ())
        {
            System.out.println( file .getAbsolutePath());
            try {
                int i = 1;
                //On parcourt la liste des fichiers et répertoires
                for( File nom : file . listFiles (){
                    //S'il s'agit d'un dossier, on ajoute un "/"
                    System.out.print(" \t\t " + ((nom.isDirectory()) ? nom.getName()+ "/" : nom.getName()));
                    if ((i%4) == 0){
                        System.out.print("\n");
                    }
                    i++;
                }
                System.out.println("\n");
            } catch (NullPointerException e) {
                //L'instruction peut générer un NullPointerException s'il n'y a
                //pas de sous-fichier !!
            }
        }
    }
}
```

### Corrigé exercice 8.6 (Gestion de fichiers)

Exemple dont nous avons perdu la source.

```
package io;

import java.io.*;

public class Copy {
    public static void main (String[] argv) {
        // Test sur le nombre de paramètres passés
        if (argv.length != 2) {
            System.out.println("Usage> java Copy sourceFile destinationFile");
            System.exit(0);
        }
        try {
            // Préparation du flux d'entrée
            File sourceFile = new File(argv[0]);
```

```

FileInputStream fis = new FileInputStream(sourceFile);
BufferedInputStream bis = new BufferedInputStream(fis);
long l = sourceFile.length();
// Préparation du flux de sortie
FileOutputStream fos = new FileOutputStream(argv[1]);
BufferedOutputStream bos = new BufferedOutputStream(fos);
// Copie des octets du flux d'entrée vers le flux de sortie
for(long i=0;i<l;i++) {
    bos.write(bis.read());
}
// Fermeture des flux de données
bos.flush();
bos.close();
bis.close();
} catch (Exception e) {
    System.err.println("File access error !");
    e.printStackTrace();
}
System.out.println("Copie terminée");
}
}

```

Voir d'autres exemples à [http://java.developpez.com/faq/java/?page=langage\\_fichiers](http://java.developpez.com/faq/java/?page=langage_fichiers).  
Ex :

```

public static void listeRepertoire ( File repertoire ) {
    System.out.println ( repertoire.getAbsolutePath());

    if ( repertoire.isDirectory ( ) ) {
        File [] list = repertoire.listFiles ();
        if ( list != null){
            for ( int i = 0; i < list.length; i++) {
                // Appel récursif sur les sous-répertoires
                listeRepertoire ( list [i] );
            }
        } else {
            System.err.println ( repertoire + " : Erreur de lecture.");
        }
    }
}
}

```

Voici une autre version avec fenêtre de dialogues, qui adapte la version précédente en s'inspirant d'un exemple de [Cha03].

```

package td8;

import java.io.*;
import javax.swing.JFileChooser;

public class CopyFromDialog {

    public static void main(String[] argv) throws IOException {
        // Préparation du flux d'entrée
        JFileChooser dialogueS = new JFileChooser(" . ");
        File sourceFile;
        if (dialogueS.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
            sourceFile = dialogueS.getSelectedFile();
            FileInputStream fis = new FileInputStream(sourceFile);
            BufferedInputStream bis = new BufferedInputStream(fis);
            long l = sourceFile.length();
            JFileChooser dialogueO = new JFileChooser(" . ");
            File outputFile;
            if (dialogueO.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
                outputFile = dialogueO.getSelectedFile();
                try {
                    // Préparation du flux de sortie
                    FileOutputStream fos = new FileOutputStream(outputFile);
                    BufferedOutputStream bos = new BufferedOutputStream(fos);
                    // Copie des octets du flux d'entrée vers le flux de sortie
                    for (long i = 0; i < l; i++) {
                        bos.write(bis.read());
                    }
                }
            }
        }
    }
}

```

```

        // Fermeture des flux de données
        bos.flush();
        bos.close();
        bis.close();
    } catch (Exception e) {
        System.err.println("File access error !");
        e.printStackTrace();
    }
    System.out.println("Copie de "+sourceFile+ " dans "+ outputFile + " terminée");
}
}
System.exit(0);
}
}

```

### Corrigé exercice 8.7 (Codage de fichiers)

Par défaut Java utilise le format Unicode pour les caractères `char`.

<http://download.oracle.com/javase/tutorial/i18n/text/convertintro.html>

Voici un exemple trouvé sur [http://java.developpez.com/faq/java/?page=langage\\_chaine](http://java.developpez.com/faq/java/?page=langage_chaine) pour passer de l'UTF à l'ASCII.

```

try {
    String chaine = "Ma chaîne à traduire !";

    // traduction en tableau de code ASCII :
    byte[] bytes = chaine.getBytes("ASCII");

    // affichage à l'écran :
    for( int i=0; i<bytes.length; i++ ) {
        System.out.println( bytes[i] );
    }
} catch( java.io.UnsupportedEncodingException e ) {
    // Le codage n'est pas reconnu.
    e.printStackTrace();
}

```

### Corrigé exercice 8.8 (Archive)

Solution de [SM03].

```
package serialisation.devjavachap14;
```

```

import java.io.BufferedOutputStream;
import java.io.BufferedInputStream;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

```

```

/**
 * Created by IntelliJ IDEA.
 * User: Pierre-Yves Saumont
 * Date: 10 févr. 2003
 * Time: 23:29:44
 * To change this template use Options | File Templates.
 */
public class Zip {
    public static void main(String[] args) throws IOException {
        FileInputStream o = Util.binOriginal
            ("Entrez le nom du fichier texte a copier : ");
        FileOutputStream d = Util.binCopie
            ("Entrez le nom a donner a la copie du fichier : ");
        BufferedInputStream original = new BufferedInputStream(o);
        BufferedOutputStream cB = new BufferedOutputStream(d);
        ZipOutputStream copie = new ZipOutputStream(cB);
        int c;
        copie.setMethod(ZipOutputStream.DEFLATED);
        copie.putNextEntry(new ZipEntry("fichier1.txt"));
    }
}

```



```

        while ((c = original.read()) != -1)
            copie.write(c);

        original.close();
        copie.close();
    }
}

package serialisation.devjavachap14;

import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;

/**
 * Created by IntelliJ IDEA.
 * User: Pierre-Yves Saumont
 * Date: 10 févr. 2003
 * Time: 23:32:50
 * To change this template use Options | File Templates.
 */
public class Unzip {
    public static void main(String[] args) throws IOException {
        FileInputStream o = new FileInputStream(args[0]);
        ZipInputStream z =
            new ZipInputStream(new BufferedInputStream(o));
        ZipEntry ze;
        ze = z.getNextEntry();
        int c;
        while ((c = z.read()) != -1)
            System.out.write(c);
        z.close();
    }
}

```

## Corrigé exercice 8.9 (Flux d'entrées-sorties générique)

```

package td8;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class GenericIO {

    public static void main(String[] args) throws IOException {
        BufferedReader entree;
        BufferedWriter sortie;

        System.out.println("Début de la copie générique");
        if (!((args.length == 2) || (args.length == 0))) {
            System.err.println("Usage: java GenericIO in out");
            System.err.println("or: java GenericIO");
            System.exit(1);
        }
        entree = new BufferedReader(new InputStreamReader(System.in));
        sortie = new BufferedWriter(new OutputStreamWriter(System.out));
        if (args.length == 2) {
            System.out.println("répertoire courant = "+System.getProperties().get("user.dir"));
            System.out.println("GenericIO "+args[0]+" to "+args[1]);
            entree = new BufferedReader(new FileReader(args[0]));
            sortie = new BufferedWriter(new FileWriter(args[1]));
        }
        else System.out.println("terminer par Ctrl+D (linux) ou Ctrl+Z (windows)");
    }
}

```

```

try {
    int c;
    while ((c = entree.read()) != -1) {
        sortie.write(c);
        System.err.print((char)c); //pour contrôle
    }
    sortie.flush();
    if (args.length == 2) {
        entree.close();
        sortie.close();
    }
} catch (IOException e) {
    System.err.println("IOException thrown " + e.toString());
    // return false;
}
System.out.println("\n Fin de la copie générique");
System.exit(0);
}
}

```

### Corrigé exercice 8.10 (Pipe)

En Java, on utilise les classes `PipedInputStream` et `PipedOutputStream`.

Exemple de *JavaTech* [LTL05], trouvé sur <http://www.particle.kth.se/~lindsey/JavaCourse/Book/Part1/Supplements/Chapter09/pipeStreams.html>.

*"The tutorial at How to Use Pipe Streams - Sun Java Tutorial provides a simple example of tying three threads together. We modified that example as shown below to illustrate how one might carry out a sequence of analysis steps. The first thread does a calculation on data read from a file and then writes the result of the calculation to a piped output stream. The second thread reads this data from the other end of this pipe and does a second level of calculations on it. (In this case the trivial calculations are for illustrative purposes only.*

*The piped streams are set up before the threads are created and started. In the `main()` method, the analysis sequence is directed by this single line of code*

```
analyzeOut(analyze2(analyze1(file_in)));
```

*The file is fed into the input of the first analysis thread, whose output then goes to the second analysis thread. Finally, the output of the latter goes to a static method for sending the output of the second thread to the console.*

*The first thread is set up and started in the static method `analyze1()`, which is passed a reference to the input file stream. The first step is to buffer the stream. A `PipedWriter` is created and then wrapped with a `PipedReader`. The `Analyze1Thread` is then created and started. It will write its output to the `PipedWriter` and that data will be read via the `PipedReader` that is returned via this method."*

```
import java.io.*;
import java.util.*;
```

```
/**
 * Demonstrate piped streams. Data is read from file and
 * read and "analyzed" by one thread, which in turn passes
 * its result via a pipe to a second thread for further analysis.
 * This thread in turn passes its data via a pipe to a method
 * for output to a file.
 */
```

```
public class PipedAnalyzer {

    public static void main(String[] args) throws IOException {

        File file = null;

        // Get the file from the argument line.
        if (args.length > 0) file = new File(args[0]);
        if (file == null) {
            System.out.println("Input data file required");
            System.exit(0);
        }
        // Create a file reader for the data file.
        FileReader file_in = new FileReader(file);

        // Carry out the two levels of analysis and then
    }
}

```

```

// write out the results.
analyzeOut (analyze2 (analyze1 (file_in )));

// Close the file stream.
file_in .close ();

} // main

/** Set up the piped streams for the first thread. Then create and
 * start the thread.
 */
public static Reader analyze1 (Reader source)
    throws IOException {

    // Buffer the input data.
    BufferedReader buf_in = new BufferedReader (source);

    // Create a pipe for data sent out by the Analyze1Thread
    PipedWriter pipe_out = new PipedWriter ();

    // What goes into the pipe can be read by the other thread via pipe_in
    PipedReader pipe_in = new PipedReader (pipe_out);

    // Start the thread for the first level analysis. It
    // will take data from buf_in and write it into pipe_out.
    new Analyze1Thread (pipe_out, buf_in).start ();

    // The other thread will be able to read via pipe_in what
    // was written into pipe_out.
    return pipe_in;

} // analyze1

/** Set up the piped streams for the second thread. Then create and
 * start the thread.
 */
public static Reader analyze2 (Reader source) throws IOException {

    // Buffer the input data.
    BufferedReader buf_in = new BufferedReader (source);

    // Create a pipe for data sent out by the Analyze2Thread
    PipedWriter pipe_out = new PipedWriter ();

    // What goes into the pipe can be read by the other thread via pipe_in
    PipedReader pipe_in = new PipedReader (pipe_out);

    // Start the thread for the second level analysis. It
    // will take data from buf_in and write it into pipe_out.
    new Analyze2Thread (pipe_out, buf_in).start ();

    // The other thread will be able to read via pipe_in what
    // was written into pipe_out.
    return pipe_in;

} // analyze2

/**
 * Read the output from the final analysis step and print to
 * the console.
 */
public static void analyzeOut (Reader source) throws IOException {

    // Buffer the input data.
    BufferedReader buf_in = new BufferedReader (source);
    System.out.println ("PipedAnalysis Output:");

    String result;
    while ((result = buf_in.readLine ()) != null)
        System.out.println (result);
    buf_in.close ();
}

```

```

    } // analyzeOut
} // PipedAnalyzer

import java.io.*;
import java.util.*;

/**
 * Helper class for PipedAnalyzer.
 * Illustrates how a first level of analysis
 * would be done on data in a sequence of threads
 * connected by piped streams.
 */
class Analyze1Thread extends Thread {
    private PrintWriter fOut;
    private Scanner fScanner;

    public Analyze1Thread (Writer out, BufferedReader in) {
        fOut = new PrintWriter (out);
        fScanner = new Scanner (in);
    }

    /** Read through the data until it runs out.
     * Do a calculation upon each value and
     * send the result to the output stream.
     */
    public void run () {
        double d = 0.0;
        if (fOut != null && fScanner != null) {
            try {
                while (true) {
                    d = fScanner.nextDouble ();
                    // Send as string to the PrintWriter stream
                    fOut.printf ("%12.5f %n", calculate (d));
                    fOut.flush ();
                }
            }
            catch (NoSuchElementException nse)
            {} // data finished
            catch (Exception e) {
                System.err.println ("Analyzer1Thread error = " + e);
            }
        }
        fOut.close ();
    } // run

    /** Do trivial calculation. */
    double calculate (double d) {
        return d*d;
    }
} // Analyze1Thread

import java.io.*;
import java.util.*;

/**
 * Helper class for PipedAnalyzer.
 * Illustrates the second level of analysis
 * on data in a sequence of threads connected
 * by piped streams.
 */
class Analyze2Thread extends Thread {
    private PrintWriter fOut;
    private Scanner fScanner;

    public Analyze2Thread (Writer out, BufferedReader in) {
        fOut = new PrintWriter (out);
        fScanner = new Scanner (in);
    } // ctor

```

```

/** Read through the data until it runs out.
 * Do a calculation upon each value and
 * send the result to the output stream.
 */
public void run () {
    double d = 0.0;
    if (fOut != null && fScanner != null) {
        try {
            while (true) {
                d = fScanner.nextDouble ();
                // Print as string to the PrintWriter stream.
                fOut.printf ("%12.5f %n",calculate (d));
                fOut.flush ();
            }
        }
        catch (NoSuchElementException nse)
        {} // data finished
        catch (Exception e) {
            System.err.println ("Analyzer1Thread error = " + e);
        }
    }
    fOut.close ();
} // run

/** Do trivial calculation. */
double calculate (double d) {
    return 2.0 * d;
}

} // Analyze2Thread

```

Le fichier `dataInput.txt` sert d'entrée, il contient

```

2,0
3,4
454,0
2,1
13,9
632,34
10
99,99

```

Ce n'est pas la version originale car nous utilisons la virgule et non le point comme séparateur décimal.

### Corrigé exercice 8.11 (Entrées-sorties binaires/Parking)

Non corrigé directement, mais reprenons un exemple similaire extrait de [SM03]. Il suffit de l'adapter.

```
package serialisation.devjavachap14;
```

```
import java.io. Serializable ;
```

```

/**
 * Created by IntelliJ IDEA.
 * User: Pierre-Yves Saumont
 * Date: 10 févr. 2003
 * Time: 23:35:15
 * To change this template use Options | File Templates.
 */
public class Voiture implements Serializable {
    Moteur moteur;
    Carrosserie carrosserie;
    transient int essence;

    Voiture (String m, String c) {
        moteur = new Moteur(m);
        carrosserie = new Carrosserie(c);
    }
    String getMoteur() {
        return moteur.getValeur();
    }
}

```

```

    }
    String getCarrosserie() {
        return carrosserie.getValeur();
    }
    void setCarburant(int e) {
        essence += e;
    }
    int getCarburant() {
        return essence;
    }
}

package serialisation.devjavachap14;

import java.io.Serializable;

/**
 * Created by IntelliJ IDEA.
 * User: Pierre-Yves Saumont
 * Date: 10 févr. 2003
 * Time: 23:38:03
 * To change this template use Options | File Templates.
 */
public class Moteur implements Serializable {
    String valeur;

    Moteur (String s) {
        valeur = s;
    }

    String getValeur() {
        return valeur;
    }
}

```

```

package serialisation.devjavachap14;

import java.io.Serializable;

/**
 * Created by IntelliJ IDEA.
 * User: Pierre-Yves Saumont
 * Date: 10 févr. 2003
 * Time: 23:37:01
 * To change this template use Options | File Templates.
 */
public class Carrosserie implements Serializable {
    String valeur;

    Carrosserie (String s) {
        valeur = s;
    }

    String getValeur() {
        return valeur;
    }
}

```

```

package serialisation.devjavachap14;

import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * Created by IntelliJ IDEA.
 * User: Pierre-Yves Saumont
 * Date: 10 févr. 2003
 * Time: 23:33:53
 * To change this template use Options | File Templates.
 */

```

```

public class Serialisation {
    public static void main (String[] args) throws IOException {
        Voiture voiture = new Voiture("V6", "Cabriolet");
        voiture.setCarburant(50);
        FileOutputStream f = new FileOutputStream("garage");
        ObjectOutputStream o = new ObjectOutputStream(f);

        o.writeObject(voiture);
        o.close();
    }
}

package serialisation.devjavachap14;

import java.io.ObjectInputStream;
import java.io.FileInputStream;
import java.io.IOException;

/**
 * Created by IntelliJ IDEA.
 * User: Pierre-Yves Saumont
 * Date: 10 févr. 2003
 * Time: 23:39:45
 * To change this template use Options | File Templates.
 */
public class Deserialisation {
    public static void main (String[] args)
        throws IOException,
        ClassNotFoundException {
        FileInputStream f = new FileInputStream("garage");
        ObjectInputStream o = new ObjectInputStream(f);

        Voiture voiture = (Voiture)o.readObject();
        o.close();

        System.out.println("Carrosserie : " + voiture.getCarrosserie());
        System.out.println("Moteur : " + voiture.getMoteur());
        System.out.println("Carburant : " + voiture.getCarburant());
    }
}

```

## Corrigé exercice 8.12 (Entrées-sorties binaires/Hôpital)

Non corrigé. S'inspirer de l'exercice précédent.

## Références

- [Cha03] Irène Charon. *Le langage Java 2 - Concepts et pratique*. Hermès, 2 edition, 2003. ISBN 2-7462-0629-3.
- [LTL05] Clark S. Lindsey, Johnny S. Tolliver, and Thomas Lindblad. *JavaTech, an Introduction to Scientific and Technical Computing with Java*. Cambridge University Press, New York, NY, USA, 2005.
- [SM03] Pierre-Yves Saumon and Antoine Mirecourt. *Le guide du développeur Java 2 - Meilleures pratiques avec Ant, Junit et les design patterns*. Eyrolles, 1 edition, 2003. ISBN 2-212-11275-0.