Plan

- Définitions
- Le paquetage java.io
  - Utiliser java.io = utiliser/envelopper/wrapper successivement plusieurs classes
- Quelques classes
  - Accès séquentiel: texte, formaté, sérialisation
  - Accès direct

Définitions

- Besoins
  - Un programme a souvent besoin d'importer de l'information depuis une source externe ou d'exporter une information vers une destination externe
- Où ?
  - Cette information peut etre : dans un fichier, sur un disque, quelquepart sur le réseau, en mémoire, dans un autre programme
- Quoi ?
  - Elle peut être de n'importe quel type : type de base, objects, caractères, image, sons…

Définitions

- Pour *importer* une information, un programme ouvre un *flot* sur une *source d'information* (un fichier, la mémoire, une socket) et *lit* l'information séquentiellement

- Pour *exporter* une information vers une *destination* externe, le programme peut ouvrir un *flot* vers cette destination et y *écrire* l'information séquentiellement

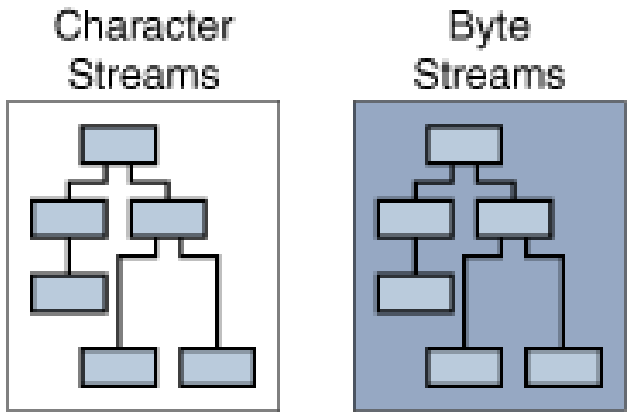
Algorithmes de base

- Lecture
  - Ouvrir un flot
  - Tant qu'il reste des infos à lire
    - Lire une information
  - Fermer le flot
- Ecriture
  - Ouvrir un flot
  - Tant qu'il reste des infos à écrire
    - Ecrire une information
  - Fermer le flot

Un flot (ou flux ou stream en anglais) cache les détails de ce qui arrive aux données

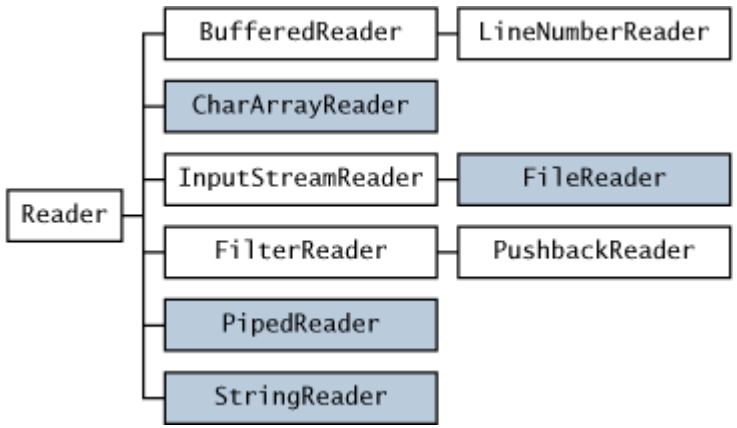
Package Java.io

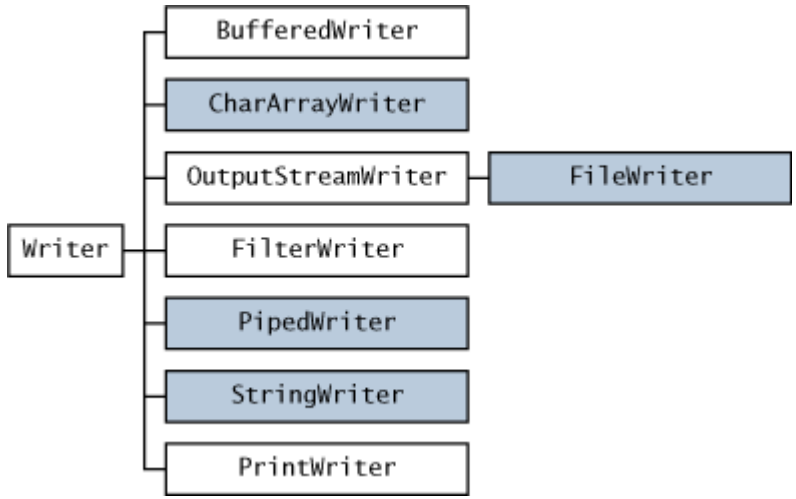
- Deux hiérarchies de classes « historiques »



Flots de caractères (Unicode 16 bits)

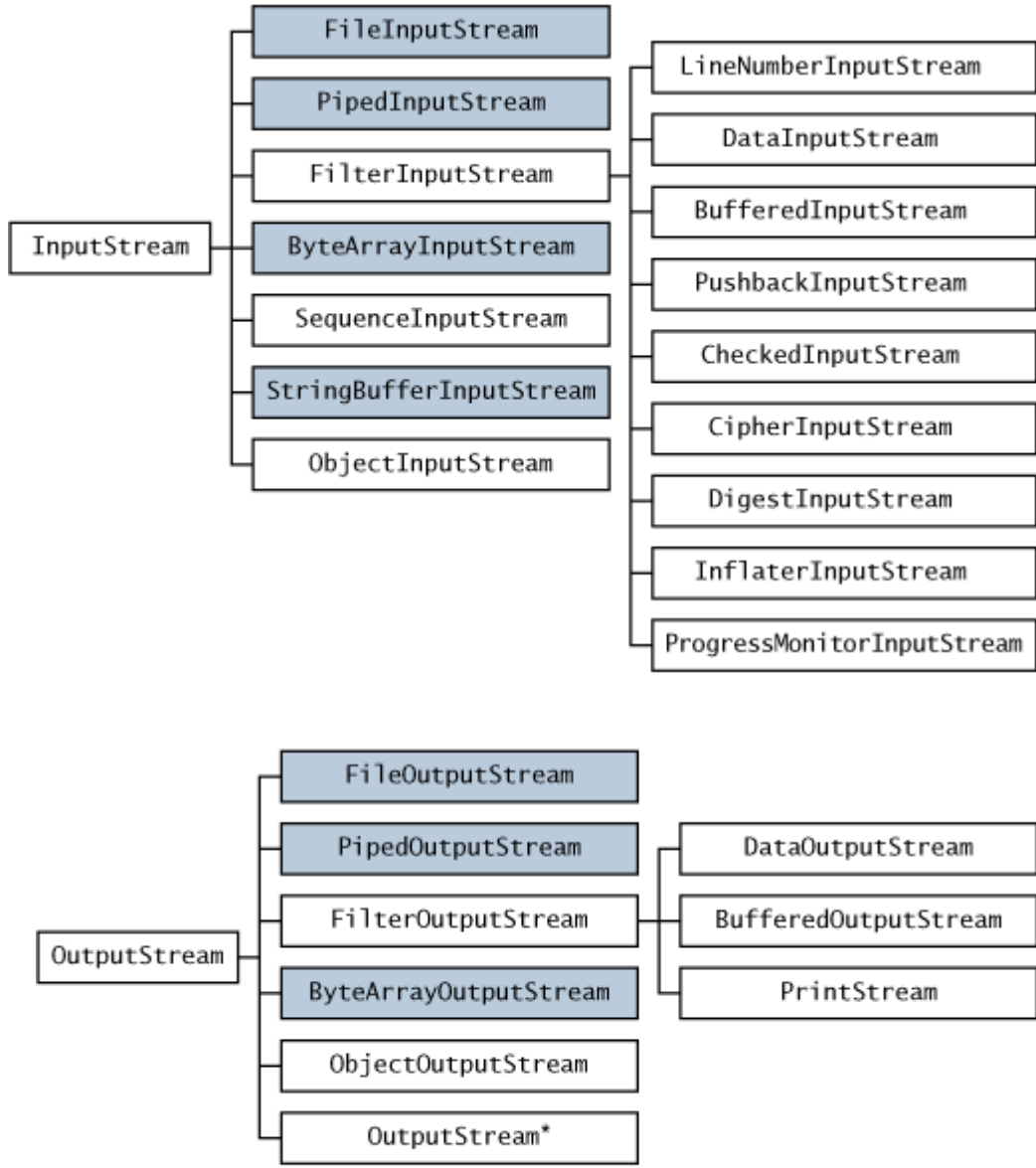
- 2 superclasses abstraites (implémentations partielles) à utiliser pour gérer du texte
- Légende
  - Bleu = lecture et écriture de base
  - Blanc = traitement en plus





Flots d'octets

- A utiliser pour des données binaires (images, sons, objets)



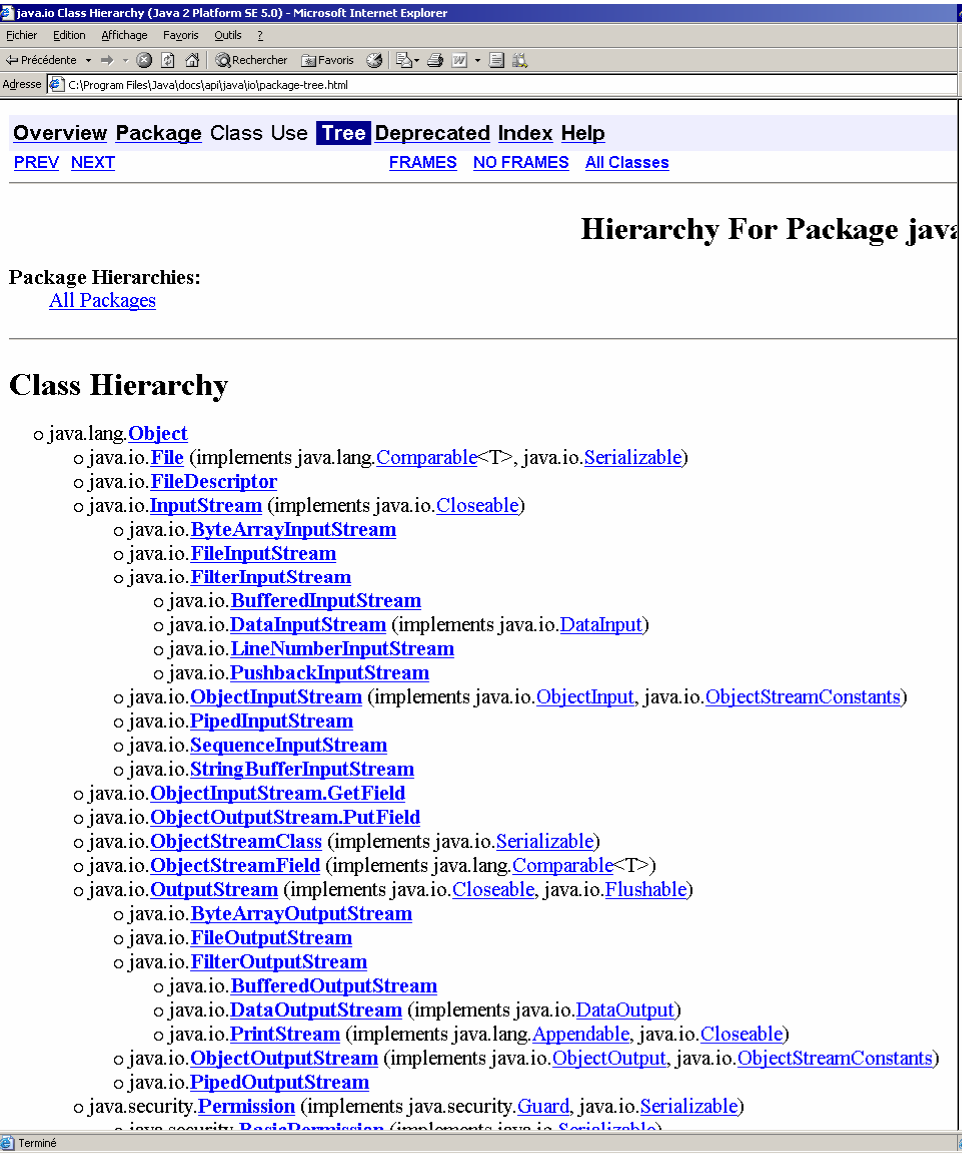
Méthodes de Reader et InputStream

- Méthodes similaires mais pour différents types
- Reader (caractères)
  - int read()
  - int read(char cbuf[])
  - int read(char cbuf[], int offset, int length)
- InputStream (octet)
  - int read()
  - int read(byte cbuf[])
  - int read(byte cbuf[], int offset, int length)
- Méthodes pour marquer une position dans le flot, sauter une information, initialiser la position courante
- On utilise généralement plutôt les méthodes des sous-classes

Méthodes de Writer et OutputStream

- Méthodes similaires mais pour différents types
- Writer (caractères)
  - int write(int c)
  - int write(char cbuf[]) int write(char cbuf[], int offset, int length)
- OutputStream (octet)
  - int write(int c)
  - int write(byte cbuf[])
  - int write(byte cbuf[], int offset, int length)
- Ouverture automatique à la création du flot
- Fermeture par l'appel à la méthode close() pour libérer des ressources système

java.io


java.io Class Hierarchy (Java 2 Platform SE 5.0) - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Précédente Rechercher Favoris
Adresse C:\Program Files\Java\docs\api\java\io\package-tree.html
Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES All Classes
Hierarchy For Package java
Package Hierarchies:
All Packages
Class Hierarchy
java.lang.Object
java.io.File (implements java.lang.Comparable<T>, java.io.Serializable)
java.io.FileDescriptor
java.io.InputStream (implements java.io.Closeable)
java.io.ByteArrayInputStream
java.io.FileInputStream
java.io.FilterInputStream
java.io.BufferedInputStream
java.io.DataInputStream (implements java.io.DataInput)
java.io.LineNumberInputStream
java.io.PushbackInputStream
java.io.ObjectInputStream (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
java.io.PipedInputStream
java.io.SequenceInputStream
java.io.StringBufferInputStream
java.io.ObjectInputStream.GetField
java.io.ObjectOutputStream.PutField
java.io.ObjectStreamClass (implements java.io.Serializable)
java.io.ObjectStreamField (implements java.lang.Comparable<T>)
java.io.OutputStream (implements java.io.Closeable, java.io.Flushable)
java.io.ByteArrayOutputStream
java.io.FileOutputStream
java.io.FilterOutputStream
java.io.BufferedOutputStream
java.io.DataOutputStream (implements java.io.DataOutput)
java.io.PrintStream (implements java.lang.Appendable, java.io.Closeable)
java.io.ObjectOutputStream (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
java.io.PipedOutputStream
java.security.Permission (implements java.security.Guard, java.io.Serializable)
Terminé


java.io Class Hierarchy (Java 2 Platform SE 5.0) - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Précédente Rechercher Favoris
Adresse C:\Program Files\Java\docs\api\java\io\package-tree.html
java.io.PrintStream (implements java.lang.Appendable, java.io.Closeable)
java.io.ObjectOutputStream (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
java.io.PipedOutputStream
java.security.Permission (implements java.security.Guard, java.io.Serializable)
java.security.BasicPermission (implements java.io.Serializable)
java.io.SerializablePermission
java.io.FilePermission (implements java.io.Serializable)
java.io.RandomAccessFile (implements java.io.Closeable, java.io.DataInput, java.io.DataOutput)
java.io.Reader (implements java.io.Closeable, java.lang.Readable)
java.io.BufferedReader
java.io.LineNumberReader
java.io.CharArrayReader
java.io.FilterReader
java.io.PushbackReader
java.io.InputStreamReader
java.io.FileReader
java.io.PipedReader
java.io.StringReader
java.io.StreamTokenizer
java.lang.Throwable (implements java.io.Serializable)
java.lang.Exception
java.io.IOException
java.io.CharConversionException
java.io.EOFException
java.io.FileNotFoundException
java.io.InterruptedIOException
java.io.ObjectStreamException
java.io.InvalidClassException
java.io.InvalidObjectException
java.io.NotActiveException
java.io.NotSerializableException
java.io.OptionalDataException
java.io.StreamCorruptedException
java.io.WriteAbortedException
java.io.SyncFailedException
java.io.UnsupportedEncodingException
java.io.UTFDataFormatException
java.io.Writer (implements java.lang.Appendable, java.io.Closeable, java.io.Flushable)
java.io.BufferedWriter
java.io.CharArrayWriter
java.io.FilterWriter
java.io.OutputStreamWriter
java.io.FileWriter

Tester l’existence d’un fichier
File

```
import java.io.*;
public class TestFile {
    public static void main (String argv[]) {
        File f = new File ("bonjour.txt");

        if (f.exists())
            System.out.println ("Le fichier bonjour.txt existe");
        else
            System.out.println ("Le fichier bonjour.txt n'existe pas");
    }
}
```

Informations sur les fichiers
File

```
import java.io.*;
public class InfosFile {
   public static void main (String argv[]) {
        File f = new File ("Infos.txt");

        if (f.exists()) {
          System.out.println ("Informations sur le fichier Infos.txt");
          System.out.println ("isFile : " + f.isFile());
          System.out.println ("isDirectory : " + f.isDirectory());
          System.out.println ("getAbsolutePath : " + f.getAbsolutePath());
          System.out.println ("length : " + f.length());
          System.out.println ("canRead : " + f.canRead());
          System.out.println ("canWrite : " + f.canWrite());
        }
   }
}
```

Lister les fichiers d'un dossier
File

```
import java.io.*;
public class ListFile {
	public static void main (String argv[]) {
		File f = new File (".");
		String[] liste = f.list() ;
		System.out.println ("Il y a " + liste.length + " fichiers")
		for (int i = 0; i < liste.length; i++)
			System.out.println (liste[i]);
	}
}
```

File permet aussi de créer, renommer ou supprimer des fichiers.

Copie d’un fichier texte
File + FileReader

```
import java.io.*;

        public class Copy {
            public static void main(String[] args) throws IOException {
                File inputFile = new File("bonjour.txt");
                File outputFile = new File ("bonjour2.txt");

                FileReader in = new FileReader(inputFile);
                FileWriter out = new FileWriter(outputFile);
                int c;

                while ((c = in.read()) != -1)
                   out.write(c);

                in.close();
                out.close();
            }
        }
```

Besoin de gérer les erreurs

- Exceptions
- Try …
- Catch …

La classe System

Lecture au clavier
La classe Console !

```
public class Console
{
 public static String readLine()
   {  int ch;
      String r = "";
      boolean done = false;
      while (!done)
      {  try
         {  ch = System.in.read();
            if (ch < 0 || (char)ch == '\n')
               done = true;
            else
               r = r + (char) ch;
         }
         catch(java.io.IOException e)
         {  done = true;
         }
      }
      return r;
   }
```

Lecture au clavier DataInputStream

```
import java.io.*;
public class Echo {
   public static void main (String argv[]) {
         DataInputStream in = new DataInputStream (System.in);
         String s ;
         try {
                  s = in.readLine();        // Lecture d'une ligne
                  System.out.println(s);    // Affichage de la ligne lue
         }
         catch (IOException e) {
                  e.printStackTrace ();
         }
   }
}
```

Ecriture formatée DataOutputStream

```
import java.io.*;

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

        // write the data out
        DataOutputStream out = new DataOutputStream(new
                                FileOutputStream("invoice1.dat"));

        double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
        int[] units = { 12, 8, 13, 29, 50 };
        String[] descs = { "Java T-shirt",
                    "Java Mug",
                    "Duke Juggling Dolls",
                    "Java Pin",
                    "Java Key Chain" };

        for (int i = 0; i < prices.length; i ++) {
            out.writeDouble(prices[i]);
            out.writeChar('\t');
            out.writeInt(units[i]);
            out.writeChar('\t');
            out.writeChars(descs[i]);
            out.writeChar('\n');
        }
        out.close();
```

Lecture formatée DataInputStream

```
 // read it in again
        DataInputStream in = new DataInputStream(new
                             FileInputStream("invoice1.dat"));

        double price;
        int unit;
        StringBuffer desc;
        double total = 0.0;

        String lineSepString = System.getProperty("line.separator");
        char lineSep = lineSepString.charAt(lineSepString.length()-1);
…
```

Lecture formatée DataInputStream

```
…
try {
            while (true) {
                price = in.readDouble();
                in.readChar();       // throws out the tab
                unit = in.readInt();
                in.readChar();       // throws out the tab

                 char chr;
                 desc = new StringBuffer(20);
                 while ((chr = in.readChar()) != lineSep)
                         desc.append(chr);

                System.out.println("You've ordered " +
                             unit + " units of " +
                             desc + " at $" + price);

                total = total + unit * price;
            }
        } catch (EOFException e) {
        }

        System.out.println("For a TOTAL of: $" + total);
        in.close();
    }
}
```

Affichage formaté System.out.format

```
public class Racine {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);

        System.out.println("La racine carree de "
                                                + i + " est " + r);
    }
}


public class Racine2 {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);

        System.out.format("La racine carree de %d est %4.2f", i, r);
    }
}
```

Sérialisation d’objets

- *Sérialiser un objet* = le transformer en une série d’octet
- Les données deviennent *persistantes*
- Tout objet implémentant l'interface *Serializable* peut être sérialisé
- Ces octets peuvent être stockés ou transmis
- L’objet peut être reconstruit à partir de ces octets

Sérialisation d’objets

- Ecriture :
  - créer un OutputStream
  - le wrapper dans un ObjectOutputStream
  - appeler writeObject()
- Lecture :
  - créer un InputStream
  - le wrapper dans un ObjectInputStream
  - appeler readObject()
  - rend une référence sur un object : faire un cast
- Des classes prédéfinies sont sérialisables

Sérialisation d’objets : exemple

```
import java.io.*;

class Personne implements Serializable {
    String nom, prenom ;
    Personne (String nom, String prenom) {
          this.nom = nom ;
          this.prenom = prenom ;
    }
    public String toString() {
          return (nom + " ," + prenom);
    }
    public static void main (String argv[]) {
          try {
                    // Enregistrement
                    ObjectOutputStream out =
                              new  ObjectOutputStream(
                                        new FileOutputStream ("personnes.dat"));
                    out.writeObject ("Sauvegarde d'une personne");
                    out.writeObject (new Personne ("Aubert", "Jean"));
                    out.close();
...
```

Sérialisation d’objets : exemple

```
            // Lecture
            ObjectInputStream in =
                        new  ObjectInputStream(
                                    new FileInputStream ("personnes.dat"));

            String s = (String) in.readObject ();
            System.out.println (s);

            Personne p = (Personne) in.readObject ();
            System.out.println (p);
            in.close();
      }
      catch (Exception e) {
            e.printStackTrace ();
      }
    }
}
```

```
D:\Martin\Java\Programmes\Serialisation>java Personne
Sauvegarde d'une personne
Aubert ,Jean

D:\Martin\Java\Programmes\Serialisation>
```

Sérialisation

- La sérialisation considère aussi les références contenues dans l’objet et sérialise les objets correspondants (copie en profondeur)
- Utile pour les tableaux, listes chaînées...

Sérialisation : exemple

```
import java.io.*;

class Compte implements Serializable {
    Personne        client ;
    int             numero ;
    double          solde ;

    Compte (String nom, String prenom, int numero, double solde) {
          client=new Personne (nom, prenom);
          this.numero = numero ;
          this.solde = solde ;
    }
    public String toString() {
          return (" compte no " + numero + " - solde " + solde + " - client : " + client);
    }
    public static void main (String argv[]) {
          try {
                    // Enregistrement
                    ObjectOutputStream out =
                              new  ObjectOutputStream(
                                        new FileOutputStream ("comptes.dat"));
                    out.writeObject ("Sauvegarde d'un compte");
                    out.writeObject (new Compte ("Aubert", "Jean", 78, 1598.50));
                    out.close();
...
```

Sérialisation : exemple

```
...
                // Lecture
                ObjectInputStream in =
                        new  ObjectInputStream(
                                new FileInputStream ("comptes.dat"));
                String s = (String) in.readObject ();
                System.out.println (s);
                Compte c = (Compte) in.readObject ();
                System.out.println (c);
                in.close();
        }
        catch (Exception e) {
                e.printStackTrace ();
        }
   }
}
```

EXECUTION :

```
D:\Martin\Java\Programmes\Serialisation>java Compte
Sauvegarde d'un compte
 compte no 78 - solde 1598.5 - client : Aubert ,Jean
D:\Martin\Java\Programmes\Serialisation>
```

Sérialisation d'un tableau

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

public class Dates implements Serializable {
     Date [] dates ;
     int nbDates ;

     Dates (int nbDates) {
          this.nbDates = nbDates ;
          dates = new Date [nbDates];
     }
```

Sérialisation d'un tableau

```
public static void main (String argv[]) {
      Dates d = new Dates (2);
      d.dates [0] = new Date ();
      d.dates [1] = new Date ();
      try {
            // Enregistrement
            ObjectOutputStream out =
              new  ObjectOutputStream(
                  new FileOutputStream ("dates.dat"));
            out.writeObject (d);
            out.close();
```

Sérialisation d'un tableau

```
public String toString() {
      String s = new String();
      for (int i = 0; i< nbDates; i++)
            s+= dates[i].toString() + "\n";
      return s ;
}
```

```
                    // Lecture
                    ObjectInputStream in =
                           new  ObjectInputStream(
                                  new FileInputStream ("dates.dat"))

                    Dates d2 = (Dates) in.readObject ();
                    System.out.println (d2);
                    in.close();
    }
    catch (Exception e) {
       e.printStackTrace ();
    }
  }
}
```

```
D:\martin\programmes\tmp>java Dates
Sat Oct 02 10:13:05 CEST 1999
Sat Oct 02 10:13:05 CEST 1999


D:\martin\programmes\tmp>
```

Personaliser la sérialisation

```
private void writeObject(ObjectOutputStream s)
throws IOException {
   s.defaultWriteObject();
   // contrôle de la sérialisation, ajout d'infos
}
private void readObject(ObjectInputStream s)
   throws IOException {
   s.defaultReadObject();

   // mise à jour de l'objet après sa lecture
}
```

La classe RandomAccessFile

```
import java.io.*;
public class Raf {
   public static void main (String argv[]) {
         try {
                  // Ecriture avec accès séquentiel
                  RandomAccessFile f = new RandomAccessFile ("chiffres.dat", "rw");
                  for (int i = 0; i < 10 ; i ++ )
                           f.writeInt (i);
                  f.close ();

                  // Ecriture avec accès direct
                  RandomAccessFile f2 = new RandomAccessFile ("chiffres.dat","rw");
                  f2.seek (5*4);
                  f2.writeInt (0);
                  f2.close ();
...
```

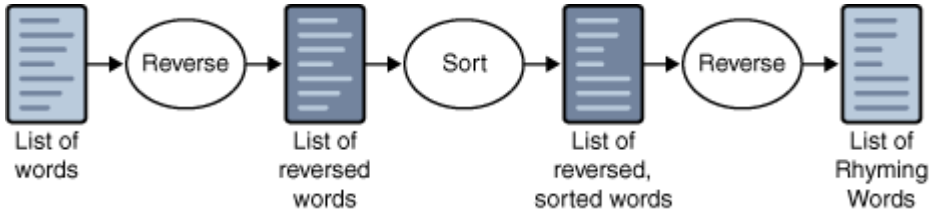
La classe RandomAccessFile

```
...
        // Lecture avec accès séquentiel
        RandomAccessFile f3 = new RandomAccessFile ("chiffres.dat", "r");
        for (int i = 0; i < 10 ; i ++ )
          System.out.println ("Valeur " + i + " : " + f3.readInt());
        f3.close ();
  }
  catch (IOException e) {
        e.printStackTrace ();
  }
 }
}
```

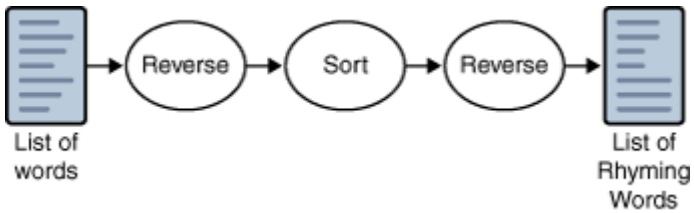
```
EXECUTION :
Valeur 0 : 0
Valeur 1 : 1
Valeur 2 : 2
Valeur 3 : 3
Valeur 4 : 4
Valeur 5 : 0
Valeur 6 : 6
Valeur 7 : 7
Valeur 8 : 8
Valeur 9 : 9
```

Autres classes
Tuyaux (Pipe)

- Sans pipe
  - Besoin de stocker les résultats intermédiaires



- Avec pipe
  - Pas besoin de stocker les résultats intermédiaires



Autres classes et méthodes

- Réorienter l'E/S standard
- Compression zip, archives jar
- Récupérer la classe d'un objet sérialisé
  - objet.getClass()
- Empêcher certaines infos d'être sérialisées (Externalizable ou transient)
- Tokenizer pour « découper »

Bibliographie

- Penser en Java version française ftp://ftp-developpez.com/java/penserenjava.zip
- http://java.sun.com/docs/books/tutorial/essential/io/index.html

PROGRAMMATION JAVA

Les Entrées-Sorties

Jean-Claude MARTIN

Filtrer les noms de fichiers

Créer un filtre de noms de fichiers

```
import java.io.* ;

public class MonFiltre implements FilenameFilter {
        String pattern ;
        public MonFiltre (String pattern) {
                this.pattern = pattern ;
        }

        public boolean accept (File dir, String name) {
                String f = new File (name).getName();
                return f.indexOf (pattern) != -1 ;
        }
}
```

Utiliser un filtre de noms de fichiers

```
import java.io.*;
public class ListJavaFile {
        public static void main (String argv[]) {
                File f = new File (".");
                MonFiltre ef = new MonFiltre (".java");
                String[] liste = f.list(ef) ;
                System.out.println ("Il y a " + liste.length +
                                        " fichiers contenant  .java");

                for (int i = 0; i < liste.length; i++)
                        System.out.println (liste[i]);
        }
}
```

La classe AWT.FileDialog

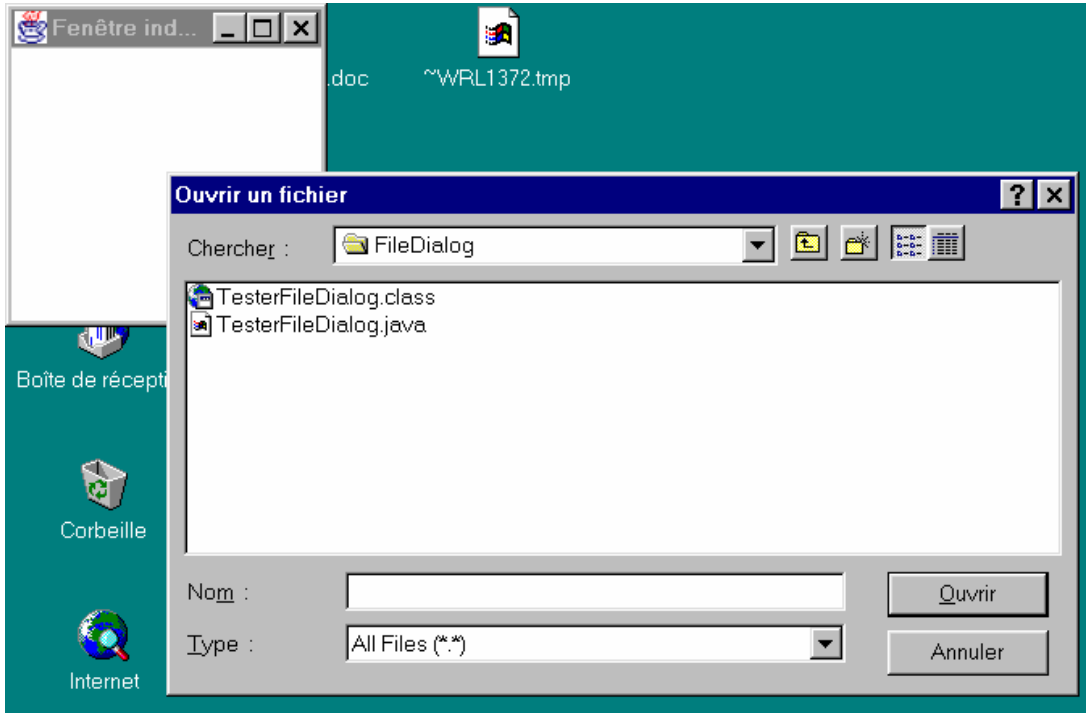
```
import java.io.*;
import java.awt.* ;

public class TesterFileDialog {
  public static void main (String argv[]) {
    Frame maFenetre = new Frame ("Fenêtre indépendante");
    maFenetre.setSize(200, 200);
    maFenetre.show();

    FileDialog f = new FileDialog(maFenetre, "Ouvrir un fichier", FileDialog.LOAD);
    f.show();
    String Nom = f.getFile();
    String Dir = f.getDirectory();
    System.out.println ("Dossier: " + Dir );
    System.out.println ("Nom: " + Nom);
   }
}
```

D:\Martin\Java\Programmes\FileDialog>java TesterFileDialog
Dossier: D:\Martin\Java\Programmes\FileDialog
Nom: TesterFileDialog.java

Exécution FileDialog




```
D:\Martin\Java\Programmes\FileDialog>java TesterFileDialog
Dossier: D:\Martin\Java\Programmes\FileDialog\
Nom: TesterFileDialog.java
```