Le monde des Entrées/Sorties

-1-

Des OS différents

- Windows NT
- Windows XP
- Windows 2000
- Windows 2003 and so on…
- Macintosh OS X 10.2 onwards…
- GNU Linux
- Red Hat Linux
- BSD

-2-

Des fichiers différents

fichiers normaux
* texte : courrier, sources des programmes, scripts, configuration ...
* exécutables : programmes en code binaire

fichiers répertoires
ce sont des fichiers conteneurs qui contiennent des références à d'autres fichiers.

fichiers spéciaux (style unix)
situés dans **/dev**, ce sont les points d'accès préparés par le système aux périphériques. Le montage va réaliser une correspondance de ces fichiers spéciaux vers leur répertoire "point de montage".
par exemple, le fichier /dev/hda permet l'accès et le chargement du 1er disque IDE

fichiers liens symboliques
Ce sont des fichiers qui ne contiennent qu'une référence (un pointeur) à un autre fichier.
Cela permet d'utiliser un même fichier sous plusieurs noms sans avoir à le dupliquer sur le disque.

-3-

Des entrées différentes

– Lecture d'un fichier depuis disque.

– Réception d'une page web depuis un serveur distant.

– Réception d'un signal au travers d'un réseau

– Scanner, video camera, ...

– souris, clavier, joystick

-4-

Des sorties différentes

– Ecriture d'un fichier sur disque local.

– Envoi d'une requète vers un serveur.

– Envoi d'une commande vers un robot.

– Impression d'un document vers un fax.

– Affichage graphique sur un écran.

-5-

Des codages différents

ISO 8859-1	Latin-1	Western European languages (French, German, Italian, etc.)
ISO 8859-2	Latin-2	Eastern European languages (Czech, Hungarian, Polish, etc.)
ISO 8859-3	Latin-3	Southern European languages (Maltese & Turkish)
ISO 8859-4	Latin-4	Northern European languages (Latvian, Lithuanian, etc.)
ISO 8859-5	Cyrillic	Russian, Bulgarian, Ukrainian, Belarusian, Serbian, Macedonian
ISO 8859-6	Arabic	Arabic
ISO 8859-7	Greek	Greek
ISO 8859-8	Hebrew	Hebrew
ISO 8859-9	Latin-5	Turkish (replaces Latin-3)
ISO 8859-10	Latin-6	Northern European (unifies Latin-1 and Latin-4)
ISO 8859-11	Thai	Thai
ISO 8859-13	Latin-7	Baltic languages (replaces Latin-4, supplements Latin-6)
ISO 8859-14	Latin-8	Celtic languages
ISO 8859-15	Latin-9	Western European languages (replaces Latin-1)
ISO 8859-16	Latin-10	Eastern European languages (replaces Latin-2)

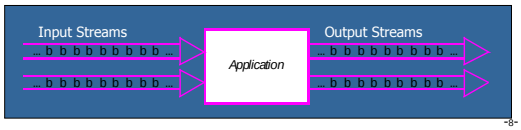
-6-

Abstraction E/S

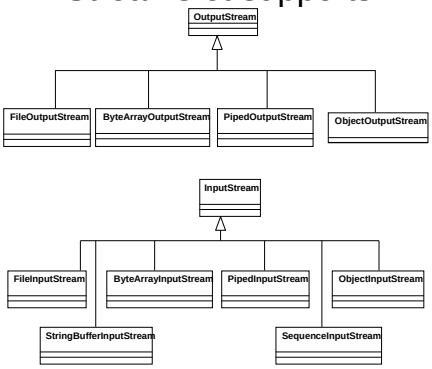
- Réaliser une interface commune pour lire et écrire tous les types de données
- Cette interface sera implémentée par la notion de flots d'E/S
  - input et output ***streams***.
- Toute E/S peut être représentée par une suite de bits.
- Cette suite est découpée en octet/Byte.

Input/Output streams

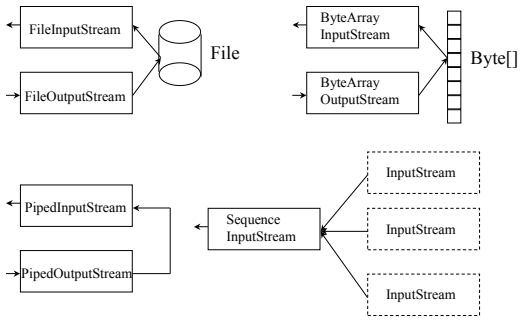
- Un flot d'E/S est donc une suite d'octets attachée à une source en entrée ou en sortie.
- On peut lire ou écrire des données séquentiellement.
- Un octet ou plusieurs octets à la fois.



Streams et supports



Input et Output



Autres sources de flots

Il existe aussi

StringBufferInputStream (1.02-Deprecated)

ObjectInputStream et ObjectOutputStream

et des streams privés tels que SocketInputStream et SocketOutputStream de java.net, seulement accessibles par le biais d'opérations Sockets

InputStream et OutputStream

<abstract>

InputStream
read () : int read (b : byte[]) : int read (b : byte[], off : int, len : int) : int mark (readlimit : int) : void skip (n : long) : long close () : void reset () : void available () : int markSupported () : boolean

OutputStream
close () : void flush () : void write (b : int) : void write (b : byte[]) : void write (b : byte[], off : int, len : int) : void

Class InputStream

```
public abstract int read()
                    throws IOException
```

Lit le prochain octet du flot.
Retourne -1 si la fin du flot est atteinte.

```
public int read(byte[] b)
                    throws IOException
```

Lit jusqu'à b.length octets depuis ce flot vers le tableau b.
Retourne le nombre d'octets lus.

```
public void close() throws IOException
```

Ferme ce flot en entrée et libère les ressources associées

Lecture ascii

```
import java.io.*;

public class SimpleIn {

  static public void main (String[] args) throws IOException {
    int charRead;
    System.out.println(System.in.getClass());
    while ((charRead = System.in.read()) >= 0)
        System.out.printlncharRead);
   }
}
```


Class OutputStream

```
public abstract void write(int b)
                    throws IOException
```

Ecrit l'octet (fournit comme int) sur le flot en sortie

```
public void write(byte[] b)
                    throws IOException
```

Ecrit b.length octets depuis le tableau d'octets spécifié vers le flot en sortie.

```
public void flush() throws IOException
```

Force l'écriture du contenu d'éventuels buffers

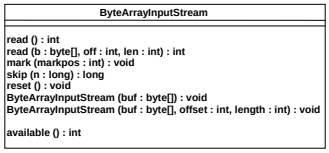
```
public void close() throws IOException
```

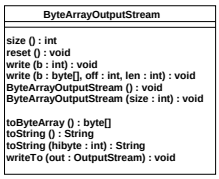
Ferme ce flot en sortie et libère les ressources associées

Ecriture Ascii

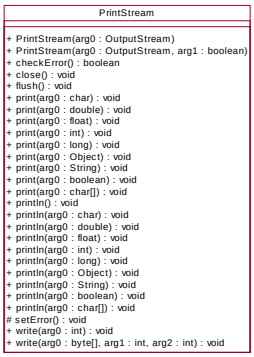
```
import java.io.*;

public class SimpleOut {
  public static void main (String[] args) throws IOException {
    for (int i = 0; i < args.length; ++ i) {
      println (args[i]);
    }
  }
  public static void println (String msg) throws IOException {
      for (int i = 0; i < msg.length (); ++ i)
        System.out.write (msg.charAt (i) & 0xff);
      System.out.write ('\n');
      System.out.flush ();
  }
}
```


ByteArray(In/Out)putStream





PrintStream



Exemple

```
public class ByteArrayOutputTest {
  static public void main (String[] args) throws IOException {
    ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream ();
    byte[] buffer = new byte[16];
    int numberRead;

    while ((numberRead = System.in.read (buffer)) > -1)
       byteArrayOut.write (buffer, 0, numberRead);
    System.out.println ("Read " + byteArrayOut.size () + " bytes.");
    System.out.write (byteArrayOut.toByteArray ());
    PrintStream printStream = new PrintStream (byteArrayOut);
    for (int i = 0; i < args.length; ++ i) {
       printStream.println ("Written to " + args[i] + ".");
       FileOutputStream fileOut = new FileOutputStream (args[i]);
       byteArrayOut.writeTo (fileOut);
       fileOut.close ();
    }
  }
}
```

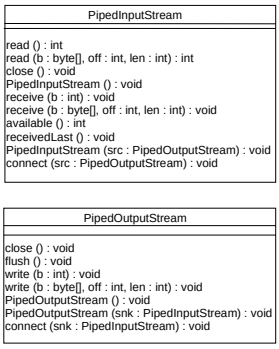

Exemple

Ecrire un programme qui lit des caractères au clavier et les insère dans une chaîne passée en paramètres avant que d'afficher à nouveau l'ensemble. (Sans utiliser la classe PrintStream pas encore étudiée)

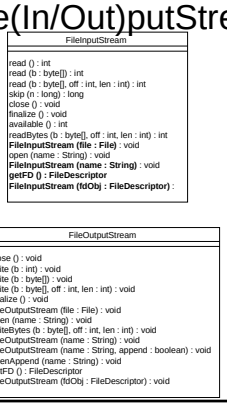
Corrigé

```
import java.io.*;

public class InputOutput {
   byte c ;
   ByteArrayOutputStream intermediaire=new ByteArrayOutputStream();
   String chaine;

   public static void main(String args[]) throws IOException{
         new InputOutput().run(args);
   }
   void run(String args[]) throws IOException{
         int i=0;
         chaine = args[0];
         while((c=(byte)System.in.read())>=0){
            intermediaire.write(c);
            if(i<chaine.length()) intermediaire.write(chaine.charAt(i++));
            }
         intermediaire.writeTo(System.out);
   }
}
```


Piped(In/Out)putStream



File(In/Out)putStream



Exemple

Ecrire un programme qui recopie un fichier texte dans un autre.

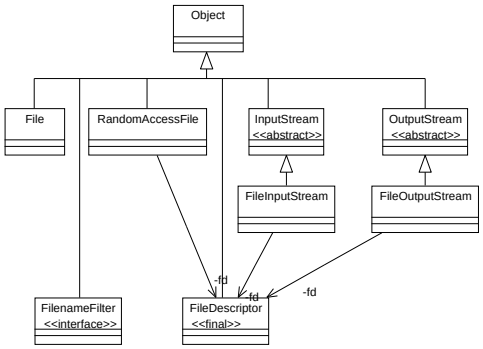
Les deux noms de fichiers seront passés en paramètres.

Corrigé

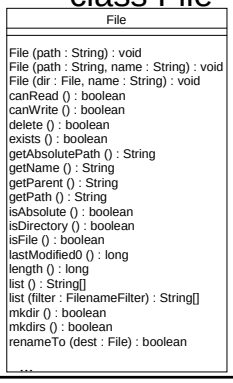
```
import java.io.*;

public class Files {

   public static void main(String args[]) throws IOException{
         FileInputStream in = new FileInputStream(args[0]);
         FileOutputStream out = new FileOutputStream(args[1]);
         byte buffer[] = new byte[256];
         int n;
         while((n=in.read(buffer))>=0) out.write(buffer,0,n);
         out.close();
         in.close();
   }
}
```


- Le constructeur **`FileOutputStream(String nom, boolean append)`** permet d'ajouter à la fin du fichier
- Sinon, le contenu du fichier est effacé à la création du flot

Streams et fichiers



class File



Exemple

Créer un programme qui prend en argument un nom de répertoire et édite ses fichiers, leur nature et les droits associés.

Corrigé 1

```
import java.io.*;  // d'après Java programming Explorer

class ListDir {

  public static void main (String args[]) {
    int MAXFILES = 100; // maximun files per directory
    if (args.length < 1) { System.out.println("Usage: Java listDir dirName");
      return;
    }
    File dirName = new File(args[0]);
    if (!dirName.exists()) {  System.out.println(args[0] + " does not exist on file system");
      return; }

    if (!dirName.isDirectory()) {System.out.println(args[0] + " is not a directory");
      return; }
    System.out.print("Files in directory: " + dirName.getAbsolutePath() + "\n\n\n");
    String fileArr[] = new String[MAXFILES];

     ....
  }
}
```

Corrigé (suite)

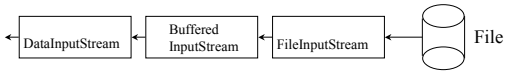
```
import java.io.*;

class ListDir {

  public static void main (String args[]) {
   ......

  try {    fileArr = dirName.list();}
    catch (Exception e) { System.out.println("Error listing contents of " + dirName); System.exit(0);
    }
    for (int i = 0; i < fileArr.length; i++) {
      File filename = new File(args[0] + System.getProperty("file.separator") + fileArr[i]);
      System.out.println("  Filename: " + fileArr[i]);
      if (filename.exists()) {
             if (filename.isFile()) { System.out.println("    Is a file");}
             if (filename.isDirectory()) {System.out.println("    Is a directory"); }
             if (filename.canWrite()) { System.out.println("    You have write permission"); }
             if (filename.canRead()) { System.out.println("    You have read permission"); }
      }
       else {System.out.println("    Does not exist");}
     System.out.println("");
     }
   }
}
```


Filtres : Etendre les Streams

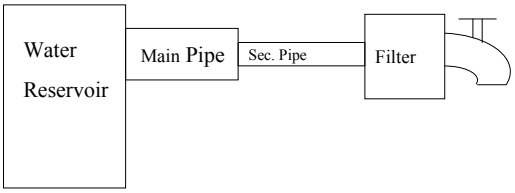
Les fonctionnalités des streams sont minimun

On peut les étendre par héritage, mais alors il faut choisir d'étendre tel ou tel des Streams ou bien de rajouter une classe pour chacun.

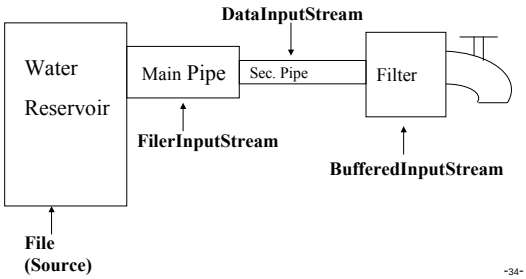
Java utilise une technique plus astucieuse : les filtres



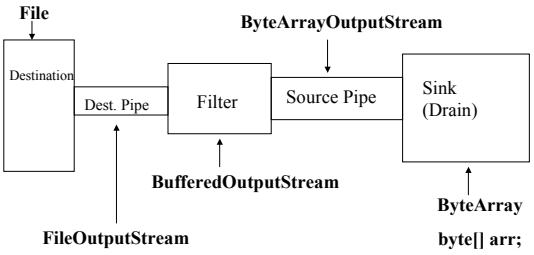
Plomberie de base !



Lecture au fil de l'eau

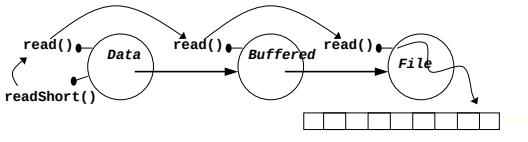


Dans l'autre sens



Chaîner les Streams

```
try {
   DataInputStream input = new DataInputStream(
      new BufferedInputStream(
         new FileInputStream(args[0])));
} catch (FileNotFoundException fnfe) {
   // ...
}
```



Pattern Decorateur

- idée:
  - Toutes les classes implémentent les mêmes fonctions de base
  - Chaque décorateur ajoute une fonctionnalité supplémentaire (“decorate”)

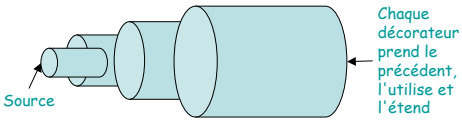
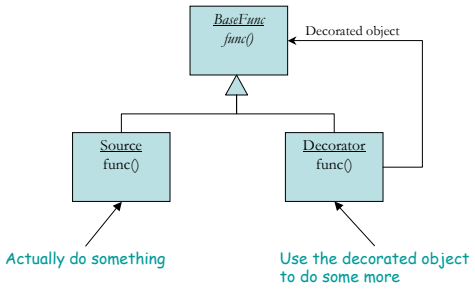


diagramme Decorateur

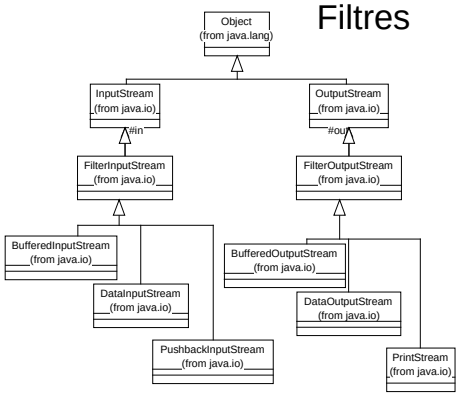


I/O decorators

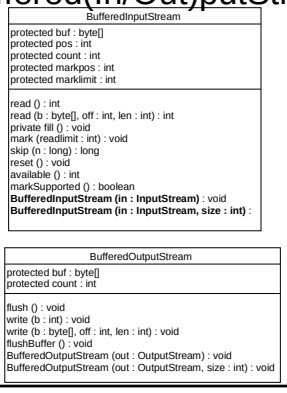
- La fonction de base est la lecture/écriture d'octets
- Sources sont les périphériques ou d'autres sources comme :
  - Files
  - Sockets
  - Byte arrays
  - Pipes
  - Strings

- Les décorations sont:
  - Reading/writing primitive (int, char, float…)
  - Reading/writing objects
  - Compressing / decompressing
  - Buffering
    - Reading lines of text
  - Printing
    - Writing lines
  - Filtering

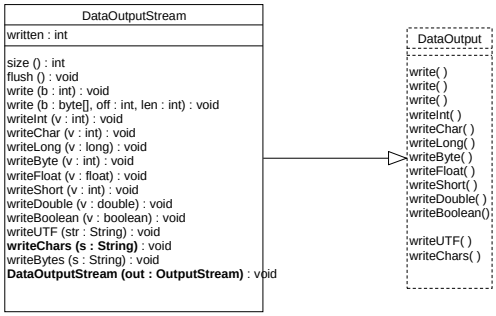
Filtres



Buffered(In/Out)putStream

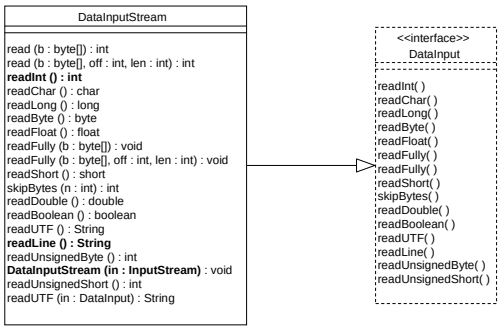


DataOutputStream



```
DataOutputStream dos =
  new DataOutputStream(
    new BufferedOutputStream(
      new FileOutputStream("fichier")));
dos.writeDouble(12.5);
dos.writeUTF("Dupond");
dos.writeInt(1254);
dos.close();
```

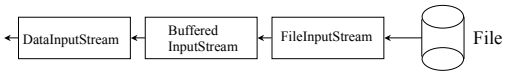

DataInputStream



PushbackInputStream

PushbackInputStream
protected buf : byte[] protected pos : int
read () : int read (b : byte[], off : int, len : int) : int **unread (b : int)** : void unread (b : byte[]) : void unread (b : byte[], off : int, len : int) : void available () : int markSupported () : boolean PushbackInputStream (in : InputStream) : void PushbackInputStream (in : InputStream, size : int) : void

Déclaration des filtres




```
DataInputStream f = new DataInputStream(new BufferedInputStream(new FileInputStream(NomFich)));

String s = f.readLine();
```


Exemple

Sélectionner un fichier texte le lire ligne à ligne et mettre celles ci à la fin d'un autre fichier.

Corrigé

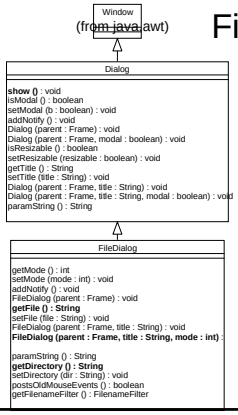
```
import java.io.*;
import java.awt.*;

public class Random extends Frame{

   public static void main(String args[]) throws IOException{
            new Random().run(args);
   }
   public void run(String args[])throws IOException{
            FileDialog fd= new FileDialog(this);
            fd.setMode(FileDialog.LOAD);
            fd.show();
            String nom=fd.getFile();
            DataInputStream iFile = new DataInputStream(new BufferedInputStream(new FileInputStream(nom)));
            RandomAccessFile oFile = new RandomAccessFile(args[0],"rw");
            oFile.seek(oFile.length());
            String ligne;
            while((ligne=iFile.readLine())!=null){
               oFile.writeBytes(ligne);
               oFile.writeByte('\n');
            }
            oFile.close();
   }
}
```

Autre filtre : Zip

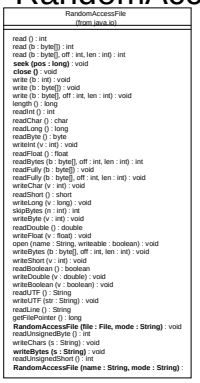
java.util.zip.ZipInputStream pour lire des fichier “zippés”
(inverse : **java.util.zip.**ZipOutputStream)

```
ZipInputStream zin = new ZipInputStream
          (new FileInputStream(“names.zip”));
ZipEntry ze = zin.getNextEntry();
if (ze.getNextEntry() != null) {
    DataInputStream dzin = new
DataInputStream(zin);
    float s = dzin.readFloat();
}
```

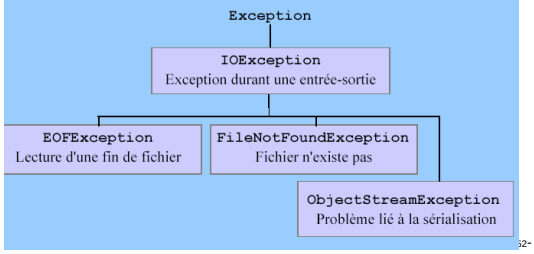

FileDialog



RandomAccessFile



Principales exceptions liées aux entrées-sorties



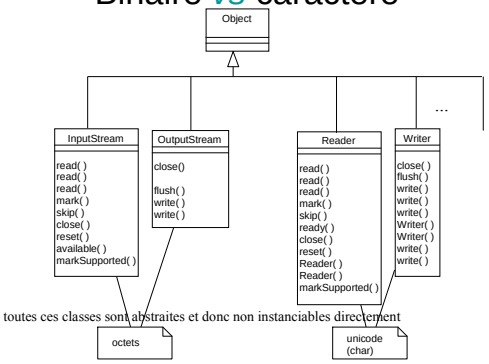

```
DataInputStream dis;
try {
  dis =
    new DataInputStream(new FileInputStream("fichier"));
  while (true) {
    double d = dis.readDouble();
    . . .
  }
}
catch(FileNotFoundException e) { . . }
catch(EOFException e) {}
catch(IOException e) { . . . }
finally {
  if (dis != null)
    try {dis.close();} catch (IOException) {…}
}
```

1 seul bloc `try`

Texte *vs* données binaires

- On distingue souvent les données textuelles des autres types de données
- Les sources sont dites 'fichiers texte' ou 'fichiers binaires'
- Les fichiers binaires utilisent des formats différents et sont affichés par des plugins spécifiques.
- La structure des fichiers texte est plus simple. Elle utilise cependant des encodages différents.

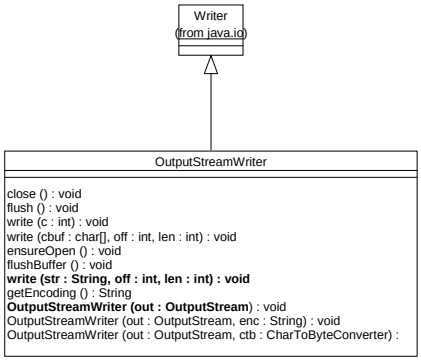
Binaire *vs* caractère



Octets et Caractères

OutputStream	Writer
InputStream	Reader
FileOutputStream	FileWriter
FileInputStream	FileReader
ByteArrayOutputStream	CharArrayWriter
ByteArrayInputStream	CharArrayReader
	StringWriter
	StringReader
BufferedOutputStream	BufferedWriter
BufferedInputStream	BufferedReader
	PrintWriter
DataOutputStream	
DataInputStream	
	OutputStreamWriter
	OutputStreamReader
ObjectInputStream	
ObjectOutputStream	

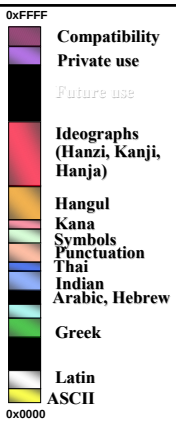
Reader et InputStream Writer et OutputStream



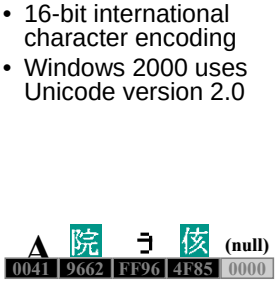
Encodage des caractères

US-ASCII	!	33
IBM-EBCDIC	!	90
Iso Latin1	é	232
UTF8	é	195 168

Unicode™ / ISO 10646

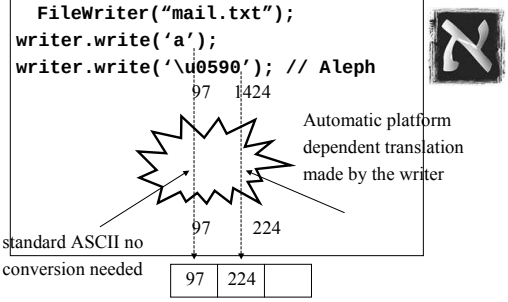


- 16-bit international character encoding
- Windows 2000 uses Unicode version 2.0



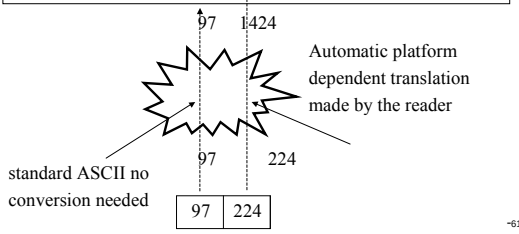
Readers & Writers

```
Writer writer = new
  FileWriter("mail.txt");
writer.write('a');
writer.write('\u0590'); // Aleph
```



Readers & Writers

```
Reader reader = new FileReader("mail.txt");
char c = reader.read(); // c = '\u0590'
c = reader.read();      // c = 'a'
```



Conversion de codage

```
public class Convert {
  public static void main (String[] args) throws IOException {
    if (args.length != 4) throw new IllegalArgumentException
                    ("Convert <srcEnc> <source> <dstEnc> <dest>");
    FileInputStream fileIn = new FileInputStream (args[1]);
    FileOutputStream fileOut = new FileOutputStream (args[3]);
    InputStreamReader inputStreamReader= new InputStreamReader (fileIn, args[0]);
    OutputStreamWriter outputStreamWriter=new OutputStreamWriter(fileOut,args[2]);
    char[] buffer = new char[16];
    int numberRead;
    while ((numberRead = inputStreamReader.read (buffer)) > -1)
          outputStreamWriter.write (buffer, 0, numberRead);
    outputStreamWriter.close ();
    inputStreamReader.close ();
  }
}
>Java Convert  Unicode   fichier.un latin1 fichier.deux
```


Class java.io.Reader

`public abstract int read(char[] buffer, int offset, int length) throws IOException`

Reads up to 'length' characters into 'buffer' starting from 'offset', returns the number of characters read.

`public void close() throws IOException`

Closes the reader.

- Few additional methods (look up in the API)

Class java.io.Writer

`public void write(int c) throws IOException`

Writes a single character given as an int.

`public void write (char[] buffer) throws IOException`

Writes a given char array.

Example: Reader Writer

```
import java.io.*;

// This class reads a text file and writes it into
// another text file after converting all letters to
// uppercase.
// Usage: java ToUpper <source> <target>
class ToUpper {

    public static void main(String[] args) {
        if (args.length!=2) {
            System.err.println("Invalid usage.");
            return;
        }
        String sourceName = args[0];
        String targetName = args[1];
```


Example (cont.)

```
 try {
        Reader reader = new FileReader(sourceName);
        Writer writer = new FileWriter(targetName);
        int c;

        while ((c=reader.read())!=-1) {
            c = Character.toUpperCase((char)c);
            writer.write(c);
        }
    } catch (IOException ioe) {
        System.err.println("Copying failed.");
    }
}
```

Persistance et Sérialisation

On a vu comment écrire ou lire des caratères, des types de base ou des chaînes de caractères. L'idéal des E/S en environnement objet consisterait à généraliser ces opérations aux objets quelconques. C'est le but de la "Serialization" qui assure ainsi la persistance de l'état d'un objet hors environnement d'exécution.

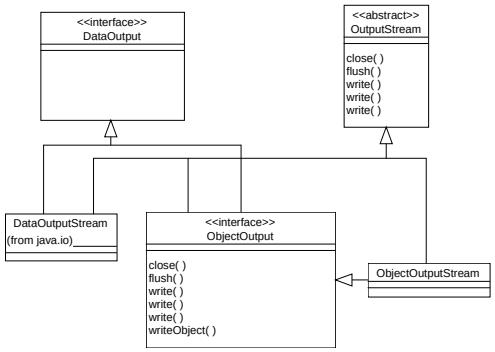
La persistance nécessite

le moyen de transformer la représentation interne en représentation sérialisée sous forme de byte-stream

le moyen de restaurer l'objet à l'identique

Généralement les objets sont restaurés avec la même classe que celle utilisée pour la sauvegarde mais la notion est étendue à une version près sous certaines réserves.

Serialisation



Sauvegarde d'un objet sur fichier

```
FileOutputStream fout = new FileOutputStream(fileName);
ObjectOutputStream out = new ObjectOutputStream(fout);

out.writeObject(obj);
```


Exemple

```
class Enregistrement implements Serializable{
   String nom;
   int note;

   public String toString(){
       return "nom :"+nom+"  note: "+note;
   }
}
```


Sauvegarde d'un objet dans un tableau d'octets

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(bout);

out.writeObject(obj);

Byte[] barray = bout.toByteArray();
```


Classes non sérialisables

Les objets dont l'état dépend de celui de l'environnement d'exécution ne sont pas sérialisables tels quel exemples :identificateur de thread ou de process, descripteur de fichiers, socket réseau, ...

Lors de la restauration rien n'indique par exemple que l'on retrouvera le fichier dans le même état.

Ce type d'objet peut justifier une implantation spécifique des procédures de sérialisation pour stocker par exemple le chemin d'accès et toute autre information utile.

ObjectOutputStream permet d'écrire aussi bien des types primitifs que des objets

Seuls des classes implémentant Serializable ou Externalizable permettent la sérialisation de leurs objets. Sont ainsi sauvegardés: le nom de cette classe, sa signature ainsi que la valeur de tous les champs non 'static' et non 'transient' , ceci récursivement pour tous les objets sérialisables définis dans les attributs. Les références multiples ou circulaires ne sont sauvegardées qu'une seule fois à l'aide d'un mécanisme de références partagées. Rappelons que les tableaux et les chaînes sont des objets et sont donc traitées également ainsi.

Les classes qui souhaitent définir un mécanisme particulier lors de la sérialisation doivent implémenter les méthodes

private void readObject(ObjectOutputStream stream)) throws IOException, ClassNotFoundException

private void writeObject(ObjectOutputStream stream)) throws IOException

Cette dernière méthode n'a pas à se préoccuper de l'écriture de ses super classes sauf si celles-ci n'implémente pas Serializable. Dans ce cas il faut sauver l'état de celle ci et elle doit offrir un constructeur sans argument pour la restauration.

Une classe qui est sérializable par héritage peut s'en défaire en implémentant ces mêmes méthodes avec levée d'une exception NotSerializableException.

Enfin si une classe souhaite la complète gestion de la sérialisation elle doit implémenter l'interface Externalizable.

interface Externalizable

hérite de Serializable

permet de contrôler la totalité de la sauvegarde, y compris de celle des super classes (seule l'identification de la classe de l'objet est traitée automatiquement)

void readExternal(ObjectInput oi);

void writeExternal(ObjectOutput oo);

Cette interface ne supporte pas automatiquement lla gestion des versions de classes.

Transmission sur sockets

```
Socket s = new Socket(remoteHost, remotePort);
OutputStream os = s.getOutputStream();
ObjectOutputStream out = new ObjectOutputStream(os);

out.writeObject(obj);
```


Résumé: Data Processing Streams

Process	Character Streams	Byte Streams
Buffering	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Filtering	FilterReader, FilterWriter	FilterInputStream, FilterOutputStream
Converting between bytes and characters	InputStreamReader OutputStreamReader	
Concatenation		SequenceInputStream
Object Serialization		ObjectInputStream, ObjectOutputStream
Data Conversion		DataInputStream, DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking Ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

Java New I/O - NIO

Neo

- New, innovative I/O framework
  - Located at its own package: `java.nio`
  - Caused changes throughout the entire library
- Based on buffers and channels
  - Like streams, but read in big chunks
- Used mainly for large-scale operations, like
  - File mapping to memory
  - Text charset decoding regular-expressions parsing
  - Channels gathering and scattering…