•Flux de données
•Utilisation des fichiers

Un flux de données est un object qui représente une suite d'octets d'un programme pour une certaine destination ou
issus d'une source pour un programme

flux d'entrée (input stream): données vers le programme
flux de sortie (output stream): données issues du programme

•flux d'entrée à partir du clavier

```
System.in
Scanner keyboard = new Scanner(System.in);
```

•flux de sortie vers l'écran ou un fichier

```
System.out
System.out.println("Output stream");
```

•Classe **`PrintWriter` est un output stream**

pour écrire sur un fichier

méthodes print, println similaires de la classe System.out

```
import java.io.PrintWriter;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;

PrintWriter outputStreamName;
outputStreamName = new PrintWriter(new
                FileOutputStream(FileName));
```

The class FileOutputStream takes a string representing
the file name as its argument
The class PrintWriter takes the anonymous FileOutputStream object
as its argument

```
outputStreamName =
 new PrintWriter(new FileOutputStream(FileName, true)); //concaténer
```


Classe Scanner pour lire à partir d'un fichier texte

```
Scanner StreamObject =
  new Scanner(new FileInputStream(FileName));
```

Les mêmes méthodes que la classe Scanner
nextInt , nextLine, nextDouble, nextLong....

Classe BufferedReader pour lire à partir d'un fihier
méthodes: read and readLine

import statements:
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

BufferedReader readerObject;
readerObject = new BufferedReader(new
FileReader(FileName));

readln retourne un String
read retourne un entier

```
char next = (char)(readerObject.read());
```


Display 10.7 **Reading Input from a Text File Using BufferedReader**

```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.FileNotFoundException;
4  import java.io.IOException;

5  public class TextFileInputDemo
6  {
7      public static void main(String[] args)
8      {
9         try
10        {
11            BufferedReader inputStream =
12               new BufferedReader(new FileReader("morestuff2.txt"));

13            String line = inputStream.readLine();
14            System.out.println(
15                           "The first line read from the file is:");
16            System.out.println(line);
```

(continued)

Display 10.7 Reading Input from a Text File Using BufferedReader

```
17
18              line = inputStream.readLine();
19              System.out.println(
20                              "The second line read from the file is:");
21              System.out.println(line);
22              inputStream.close();
23          }
24          catch(FileNotFoundException e)
25          {
26              System.out.println("File morestuff2.txt was not found");
27              System.out.println("or could not be opened.");
28          }
29          catch(IOException e)
30          {
31              System.out.println("Error reading from morestuff2.txt.");
32          }
33      }
34  }
```

(continued)

Display 10.7 Reading Input from a Text File Using BufferedReader

File morestuff2.txt

```
1 2 3
Jack jump over
the candle stick.
```

This file could have been made with a text editor or by another Java program.

SCREEN OUTPUT

```
The first line read from the file is:
1 2 3
The second line read from the file is:
Jack jump over
```

Display 10.12 Some Methods in the Class `File`

`File` is in the `java.io` package.

`public File(String File_Name)`

Constructor. *File_Name* can be either a full or a relative path name (which includes the case of a simple file name). *File_Name* is referred to as the **abstract path name**.

`public boolean exists()`

Tests whether there is a file with the abstract path name.

`public boolean canRead()`

Tests whether the program can read from the file. Returns `true` if the file named by the abstract path name exists and is readable by the program; otherwise returns `false`.

(continued)

Display 10.12 Some Methods in the Class `File`

`public boolean setReadOnly()`

Sets the file represented by the abstract path name to be read only. Returns `true` if successful; otherwise returns `false`.

`public boolean canWrite()`

Tests whether the program can write to the file. Returns `true` if the file named by the abstract path name exists and is writable by the program; otherwise returns `false`.

`public boolean delete()`

Tries to delete the file or directory named by the abstract path name. A directory must be empty to be removed. Returns `true` if it was able to delete the file or directory. Returns `false` if it was unable to delete the file or directory.

(continued)

Display 10.12 **Some Methods in the Class `File`**

`public boolean createNewFile() throws IOException`

Creates a new empty file named by the abstract path name, provided that a file of that name does not already exist. Returns `true` if successful, and returns `false` otherwise.

`public String getName()`

Returns the last name in the abstract path name (that is, the simple file name). Returns the empty string if the abstract path name is the empty string.

`public String getPath()`

Returns the abstract path name as a `String` value.

`public boolean renameTo(File` *New_Name*`)`

Renames the file represented by the abstract path name to *New_Name*. Returns `true` if successful; otherwise returns `false`. *New_Name* can be a relative or absolute path name. This may require moving the file. Whether or not the file can be moved is system dependent.

(continued)

Display 10.12 **Some Methods in the Class `File`**

`public boolean isFile()`

Returns `true` if a file exists that is named by the abstract path name and the file is a normal file; otherwise returns `false`. The meaning of *normal* is system dependent. Any file created by a Java program is guaranteed to be normal.

`public boolean isDirectory()`

Returns `true` if a directory (folder) exists that is named by the abstract path name; otherwise returns `false`.

(continued)

Display 10.12 **Some Methods in the Class `File`**

`public boolean mkdir()`

Makes a directory named by the abstract path name. Will not create parent directories. See `mkdirs`. Returns `true` if successful; otherwise returns `false`.

`public boolean mkdirs()`

Makes a directory named by the abstract path name. Will create any necessary but nonexistent parent directories. Returns `true` if successful; otherwise returns `false`. Note that if it fails, then some of the parent directories may have been created.

`public long length()`

Returns the length in bytes of the file named by the abstract path name. If the file does not exist or the abstract path name names a directory, then the value returned is not specified and may be anything.

```
import java.io.*;

public class CopyBytes {
   public static void main(String[] args) throws IOException {
      File inputFile = new File("farrago.txt");
      File outputFile = new File("outagain.txt");

      FileInputStream in = new FileInputStream(inputFile);
      FileOutputStream out = new FileOutputStream(outputFile);
      int c;
      while ((c = in.read()) != -1)
        out.write(c);

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