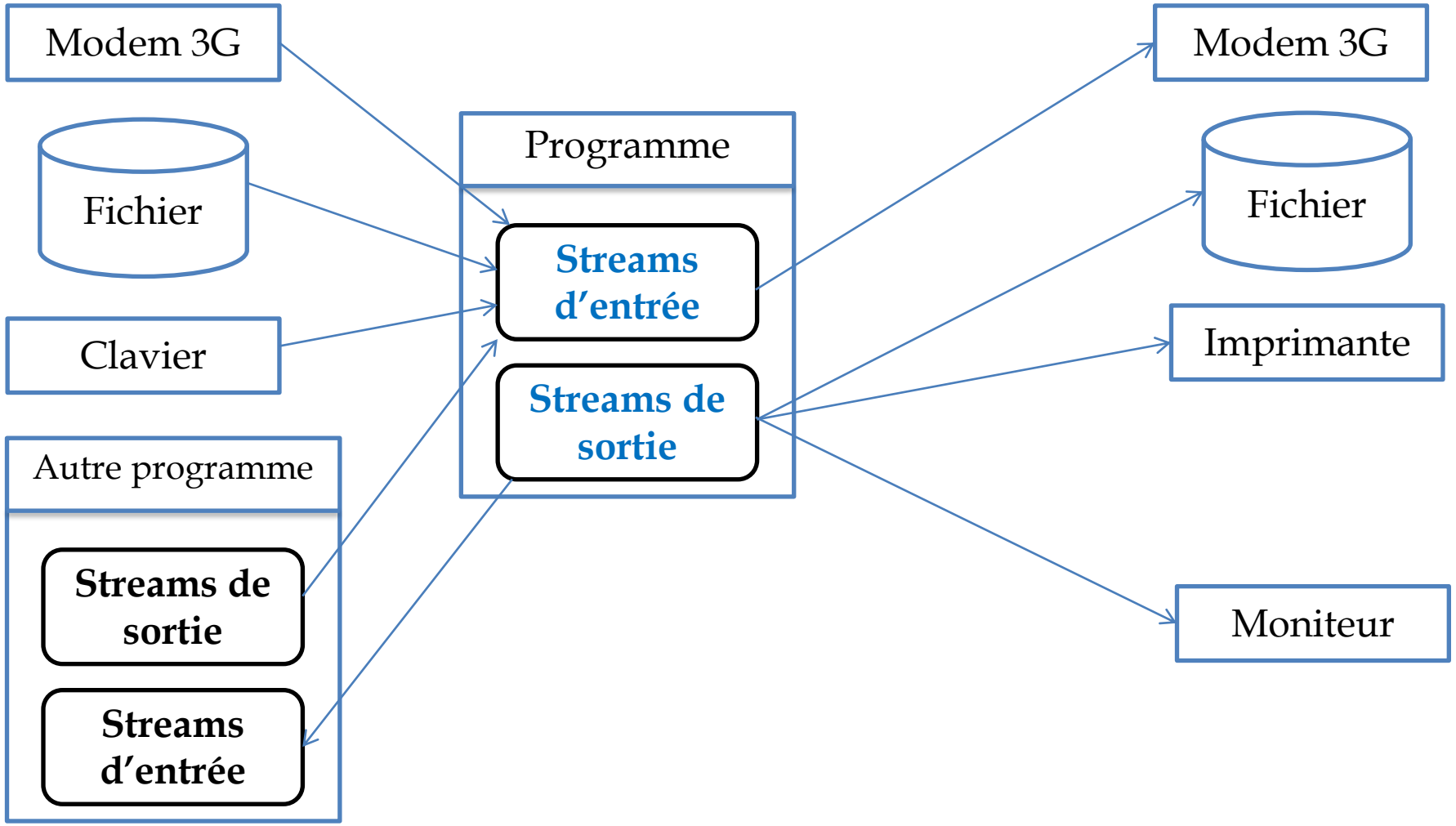
Programmation orientée objet

Chapitre9: Les fichiers (entrées/sorties)

Entées/Sorties



Principes d'entrées/sorties

- Pour effectuer une entrée ou une sortie de données en Java, le principe est simple et se résume aux opérations suivantes :
  - Ouverture d'un moyen de communication.
  - Écriture ou lecture des données.
  - Fermeture du moyen de communication.
- En Java, les moyens de communication sont représentés par des objets particuliers appelés (en anglais) ***stream***. Ce mot, qui signifie ***courant***, ou ***flot***

Les flux en java

Le Package **java.io** offre une véritable collection de classes permettant la gestion des Entrées/Sorties. On énumère alors les classes suivantes:

BufferedInputStream
BufferedOutputStream
BufferedReader
BufferedWriter
ByteArrayInputStream
ByteArrayOutputStream
CharArrayReader
CharArrayWriter
DataInputStream
DataOutputStream
File
FileDescriptor
FileInputStream
FileOutputStream
FilePermission
FileReader
FileWriter
FilterInputStream
FilterOutputStream
FilterReader
FilterWriter
InputStream
InputStreamReader
LineNumberInputStream
LineNumberReader
ObjectInputStream
ObjectInputStream.GetField
ObjectOutputStream
ObjectOutputStream.PutField
ObjectStreamClass
ObjectStreamField
OutputStream
OutputStreamWriter
PipedInputStream
PipedOutputStream
PipedReader
PipedWriter
PrintStream
PrintWriter
PushbackInputStream
PushbackReader
RandomAccessFile
Reader
SequenceInputStream
SerializablePermission
StreamTokenizer
StringBufferInputStream
StringReader
StringWriter
Writer

Les classes utilisées

Pour une manipulation assez exhaustive des fichiers nous avons choisis les 13 classes suivantes :

File
FileReader
FileWriter
RandomAccessFile
FileInputStream
FileOutputStream
InputStreamReader
OutputStreamWriter
BufferedReader
BufferedWriter
PrintWriter
ObjectInputStream
ObjectOutputStream

La classe File

La classe **File** permet de donner des informations sur un fichier ou un répertoire
La création d'un objet de la classe File peut se faire de différentes manières :

```
File f1=new File("c:/projet/fichier.ext");
File f2=new File("c:/projet", "fihier.ext");
File f3=new File("c:/projet");
```

Exemple d'utilisation de la classe File

- Afficher le contenu d'un répertoire en affichant si les éléments de ce répertoire sont des fichier ou des répertoires.
- Dans le cas ou il s'agit d'un fichier afficher la capacité physique du fichier.

```
public class Application1 {
 public static void main(String[] args) {
  String rep="c:/"; File f=new File(rep);
  if(f.exists()){
   String[] contenu=f.list();
   for(int i=0;i<contenu.length;i++){
    File f2=new File(rep,contenu[i]);
    if(f2.isDirectory())
      System.out.println("REP:"+contenu[i]);
    else
      System.out.println("Fichier :"+contenu[i]+"Size:"+f2.length());
   }
  }
   else{
     System.out.println(rep+" n'existe pas");
  }
}}
```

Les fichiers textes

Fonctions de lecture : classe FileReader

Constructeur :

FileReader (String NomDeFichier)

FileReader f = **new FileReader**("**test.txt**");

Si le fichier n'existe pas l'ouverture déclenche une exception qui peut être attrapée de la manière suivante :

```
try {
f = new FileReader("test2.dat");
}
catch (Exception e) {
System.out.println("Erreur d'ouverture du fichier");
System.out.println(e.getMessage());
}
```

FileReader: Méthodes

`int read()`	Lit un caractère. Retourne –1 si fin de fichier.
`int read(char[] cbuf)`	Lit un tableau de caractères et retourne le nombre de caractères lus,
`int read(char[] Tchar, int debut,int Nombre)`	Lit une séquence de caractères dans une portion de tableau et retourne le nombre de caractères lus.
`skip``(long n)`	Saut de n caractères. n doit être un nombre positif
`close``()`	Fermeture du fichier

FileWriter

Fonctions d'écriture: classe FileWriter

Constructeur :

FileWriter (String NomDeFichier)

FileWriter f = **new FileWriter**("**test.dat**");

- Écrasement du contenu du fichier si existant
- Les erreurs d'ouverture peuvent aussi être détectées par l'intermédiaire de la clause: try … catch

FileWriter: Méthodes

`void write(int c)`	Écrit un caractère dans le fichier.
`void write(char[] Tchar)`	Écrit un tableau de caractères.
`void write(char[] Tchar, int Debut, int Nombre)`	Écrit une portion d'un tableau de caractères.
`void write(String str)`	Écrit une chaîne de caractères.
`void write(String str, int Debut, int Nombre)`	Écrit une portion d'une chaîne de caractères.
`flush``()`	vider le tampon d'écriture dans le fichier
`close``()`	Fermeture du fichier

Les fichiers Binaires

Pour l'ajout, la modification, accès direct etc… on utilise la classe « **RandomAccessFile** ». Celle-ci peut cependant être utilisée pour gérer aussi bien des fichiers **textes** que des fichiers **binaires**

Constructeur:

RandomAccessFile (String NomDeFichier, String Mode) avec **Mode** une chaîne de caractères qui précise le mode d'ouverture. Elle peut prendre deux valeurs possibles:

"r"	Ouverture en mode lecture. Si le fichier n'existe pas une exception est déclenchée.
"rw"	ouverture en mode mise à jour : Lecture+Ecriture+Ajout+Accès Direct. Si le fichier existe son contenu n'est pas écrasé et le pointeur de fichier est placé à la fin du fichier. Sinon, il est alors créé.

RandomAccessFile

Exemple:

```
public class App{
public static void main(String args[]) throws IOException {
RandomAccessFile f;
f = new RandomAccessFile("test.dat","rw");
f.seek(f.length());
f.writeBytes("\r\nceci est une nouvelle ligne");
f.write(20);
f.writeBytes("#");
int x = 65;
f.write(x);
f.writeBytes(" 30.58");
f.close();}
}
```

RandomAccessFile: Méthodes

Méthodes de lecture:

```
int read()
int read(byte[] cbuf)
int read(byte[] Tchar, int debut, int
Nombre)
boolean readBoolean()
byte readByte()
char readChar()
double readDouble()
float readFloat()
void readFully(byte[] b)
void readFully(byte[] b,int off,int len)
int readInt()
String readLine()
long readLong()
short readShort()
int readUnsignedByte()
int readUnsignedShort()
String readUTF()
```

Méthodes d'écriture :

```
void write(int b)
void write(byte[] Tbyte)
void write(byte[] Tbyte, int
Debut, int Nombre)
void writeBoolean(boolean v)
void writeByte(int v)
void writeBytes(String s)
void writeChar(int v)
void writeChars(String s)
void writeDouble(double v)
void writeFloat(float v)
void writeInt(int v)
void writeLong(long v)
void writeShort(int v)
void writeUTF(String str)
```

FileInputStream et FileOutputStream

Permettent de lire et écrire sur un fichier binaire

Exemple:

```
File f1=new File("D:/image1.jpg");
FileInputStream fis=new FileInputStream(f1);
File f2=new File("D:/image2.jpg");
FileOutputStream fos=new FileOutputStream(f2);
int c;
while((c=fis.read())!=-1){
fos.write(c);
}
fis.close();fos.close();
```

InputStreamReader et OutputStreamReader

Permettent de lire et écrire sur un fichier texte

Exemple:

```
InputStream is=new FileInputStream("D:/Fichier1.txt");
OutputStream os=new FileOutputStream("D:/Fichier2.txt");
InputStreamReader isr=new InputStreamReader(is);
OutputStreamWriter osr=new OutputStreamWriter(os);
int c;
while((c=isr.read())!=-1){
        osr.write(c);
}
isr.close();osr.close();
```

BufferedReader et BufferedWriter

Permettent de lire et écrire sur un fichier texte ligne par ligne

Exemple:

```
File f1=new File("D:/Fichier1.txt");
FileReader fr=new FileReader(f1);
BufferedReader br=new BufferedReader(fr);
File f2=new File("D:/Fichier2.txt");
FileWriter fw=new FileWriter(f2);
BufferedWriter bw=new BufferedWriter(fw);
String s;
while((s=br.readLine())!=null){
bw.write(s);bw.newLine();
}
br.close();bw.close();
```

PrintWriter

On peut écrire sur un fichier texte ligne par ligne via la classe PrintWriter

Exemple:

```
File f1=new File("D:/Fichier1.txt");
FileReader fr=new FileReader(f1);
BufferedReader br=new BufferedReader(fr);
File f2=new File("D:/Fichier2.txt");
PrintWriter pw=new PrintWriter(f2);
String s;
while((s=br.readLine())!=null){
pw.println(s);
}
br.close();pw.close();
```

Les fichiers Binaires d'enregistrement/persistance des objets

Classes utilisées :

ObjectInputStream : pour la lecture

ObjectOutputStream : pour l'écriture

Constructeur :

ObjectInputStream (**new FileInputStream(**NomDeFichier))

ObjectOutputStream (**new FileOutputStream(**NomDeFichier))

L'objet à lire ou à écrire doit appartenir à un objet **sérialisable**. La classe de l'objet doit alors implémenter l'interface **Serializable**.

Sérialisation: Exemple

```
public class Produit implements Serializable {
    private int num;
    private String designation;
    private float prix;
    public Produit(int num, String designation, float prix) {
    this.num=num;
    this.designation=designation;
    this.prix=prix;
}
    @Override
    public String toString() {
        return num+" -- "+designation+" -- "+prix;
    }
}
```

Sérialisation: Exemple

```
public class App8 {
    public static void main(String[] args) {
        try {
        ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream("produit.dat"));
        ObjectInputStream ois=new ObjectInputStream(new FileInputStream("produit.dat"));
        Produit p=new Produit(10, "Ordinateur", 5600);
        oos.writeObject(p);
        Produit p2=(Produit)ois.readObject();
        System.out.println(p2);
        oos.close(); ois.close();
        } catch (FileNotFoundException e) {
        e.printStackTrace();
        } catch (IOException e) {
        e.printStackTrace();
        } catch (ClassNotFoundException e) {
        e.printStackTrace();}
    }
}
```