Introduction à la programmation en `Java`

Cours 6

Stéphane Airiau

Université Paris-Dauphine

Entrée et sortie

Entrée / sortie

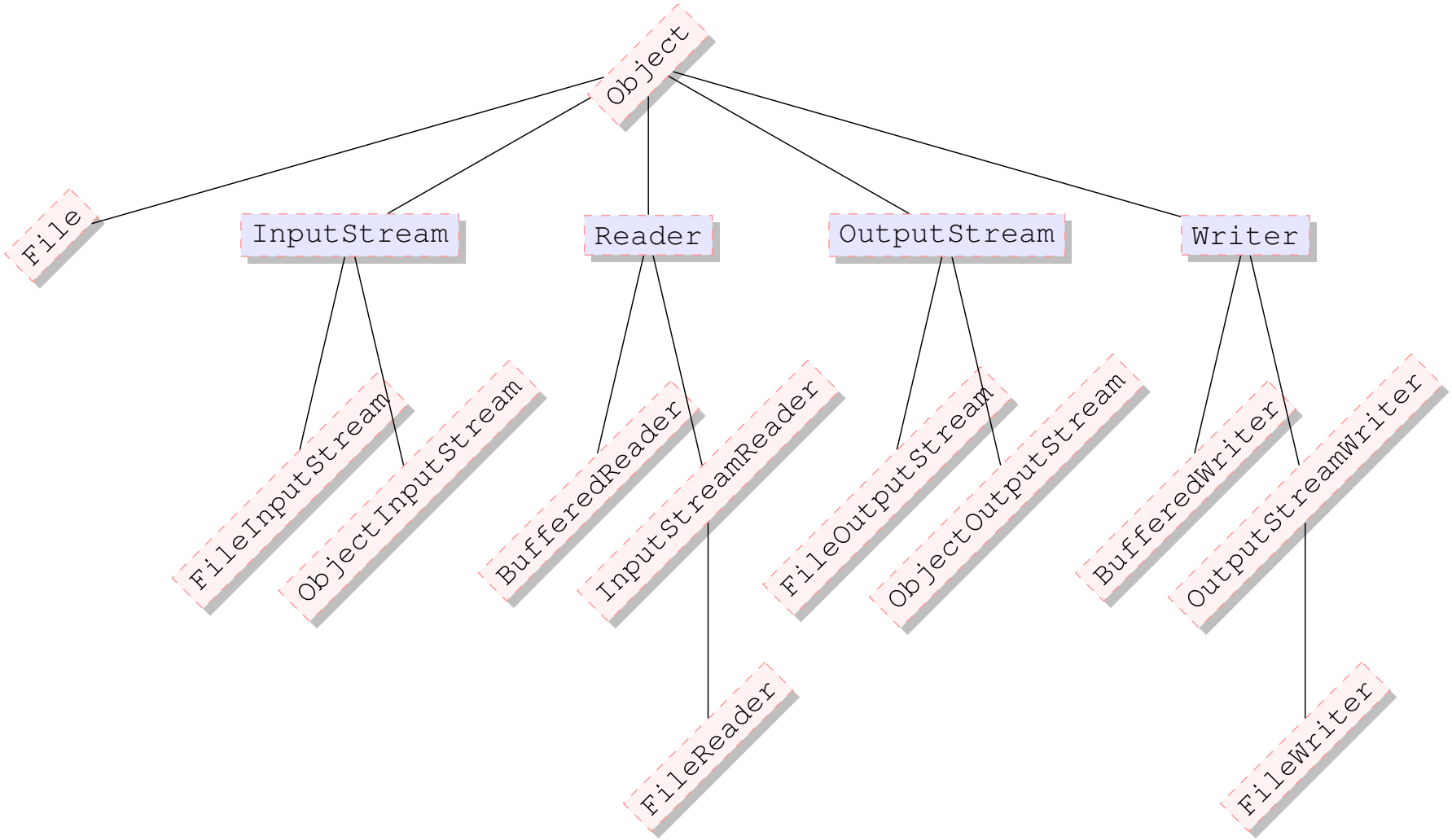
Entrée/sortie : échange de données entre le programme et une source :

- entrée : au clavier, lecture d'un fichier, communication réseau
- sortie : sur la console, écriture d'un fichier, envoi sur le réseau

⇨ `Java` utilise des flux (stream en anglais) pour abstraire toutes ses opérations.
de manière générale, on observera trois phases :

1- ouverture du flux
2- lecture/écriture du flux
3- fermeture du flux

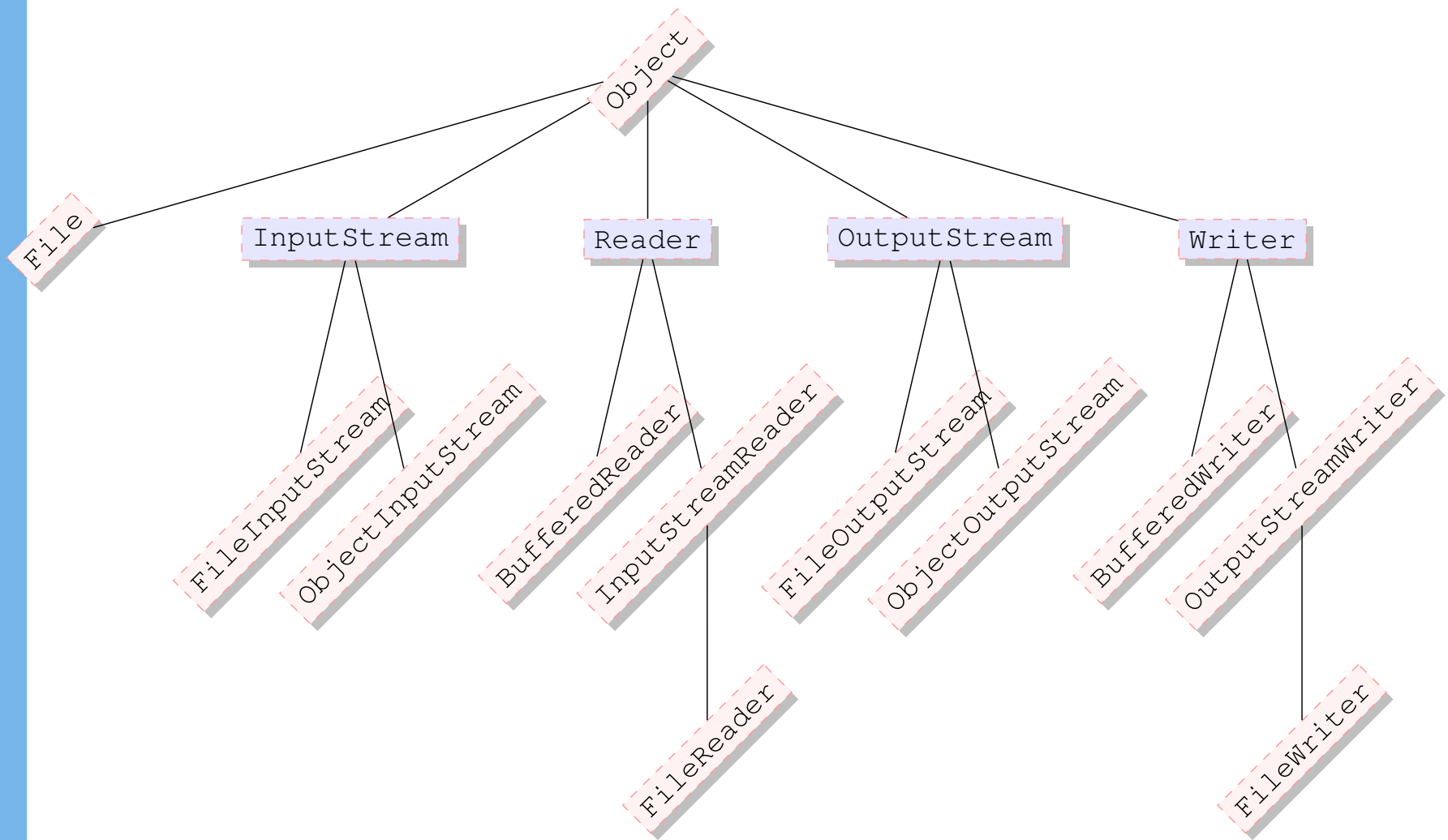
le package `java.io` (fragment)



La classe `File`

La classe `File` permet d'obetenir des informations sur les fichiers

- nom, chemin absolu, répertoire parent
- s'il existe un fichier d'un nom donné en paramètre
- droit : l'utilisateur a-t-il le droit de lire ou d'écrire dans le fichier
- la nature de l'objet (fichier, répertoire)
- la taille du fichier
- obtenir la liste des fichiers
- effacer un fichier
- créer un répertoire
- accéder au fichier pour le lire ou l'écrire


Object
File
InputStream
Reader
OutputStream
Writer
FileInputStream
ObjectInputStream
BufferedReader
InputStreamReader
FileReader
FileOutputStream
ObjectOutputStream
BufferedWriter
OutputStreamWriter
FileWriter

Flux

Les flux transportent des `bytes` ou des `char`.

Direction du Flux :

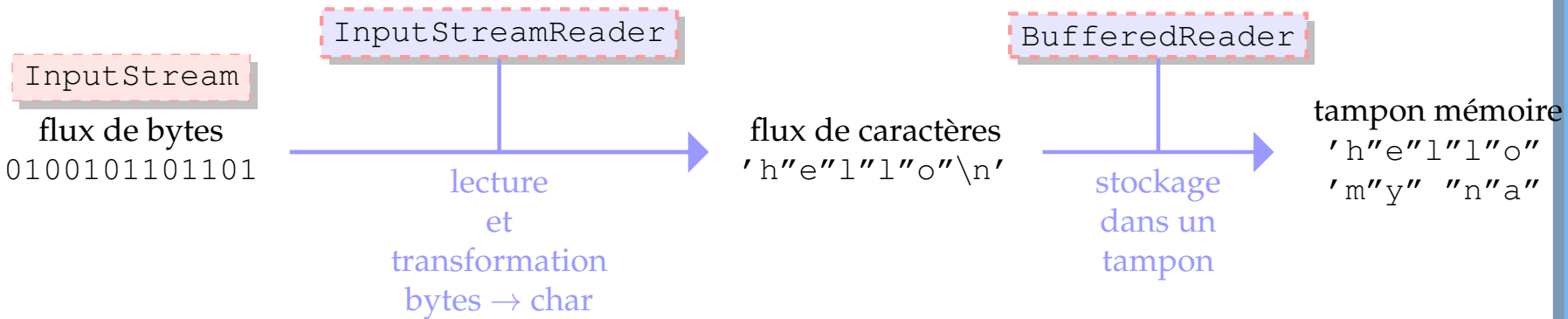
- objets qui gèrent des flux d'entrée : **`in`**
  - ➪ **`In`**`putStream`, `File`**`In`**`putStream`, `File`**`In`**`putStream`
- objets qui gèrent des flux de sortie : **`out`**
  - ➪ **`Out`**`putStream`, `File`**`Out`**`putStream`, `File`**`Out`**`putStream`

Source du flux :

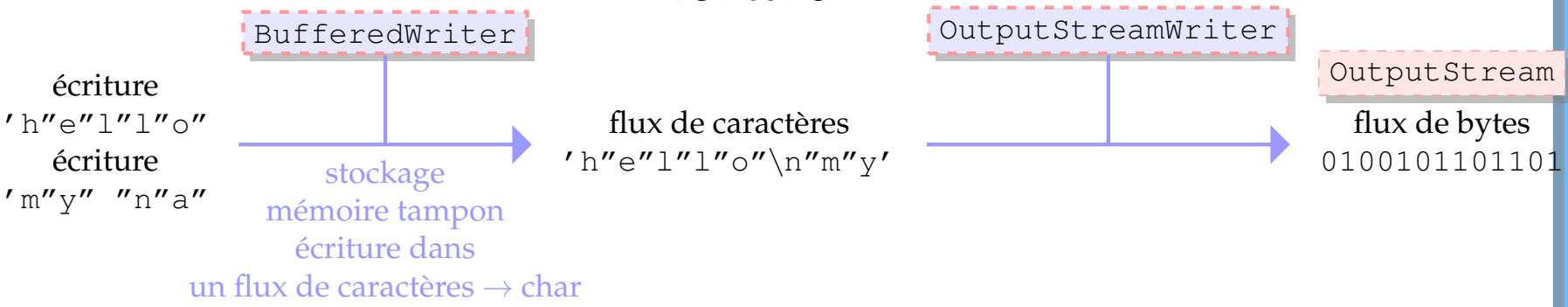
- **fichiers** : on pourra avoir des flux vers ou à partir de fichiers
  - ➪ **`File`**`InputStream` et **`File`**`OutputStream`
- **objets** : on pourra envoyer/recevoir un objet via un flux
  - ➪ **`Object`**`InputStream` et **`Object`**`OutputStream`

Processus de lecture et d'écriture

Lecture



Ecriture



Selon le type de la source ou de la destination (fichier, objet), on utilisera

- `FileReader` à la place de `InputStreamReader`
- `FileOutputStream` ou `ObjectOutputStream` comme implémentation de la classe abstraite `OutputStream`

Exemple : Lecture d'un fichier

Lecture du premier octet d'un fichier

```
1 FileInputStream fis =
2                 new FileInputStream(new File("ex.txt"));
3 byte[] huitLettres = new byte[8];
4 int nbLettreLues = fis.read(huitLettres);
5 for(int i=0;i<8;i++)
6    System.out.println(Byte.toString(huitLettres[i]));
```

Affiche un fichier sur la console

```
1 BufferedReader reader =
2    new BufferedReader(new FileReader(new File("ex.txt")));
3 String line = reader.readLine();
4 while(line != null){
5    System.out.println(line);
6    line = reader.readLine();
7 }
8 reader.close();
```

N.B. Les codes ne sont pas corrects (gestion des exceptions)

Envoi d'objets

But : envoyer toute l'information d'un objet
⇨ mécanisme de « sérialisation »

- la classe doit implémenter l'interface `Serializable`
- l'interface `Serializable` **n'a pas** de méthodes : c'est juste un marqueur.
- `Java` transforme l'objet automatiquement en un code pas lisible pour les humains

NB : Si un attribut de la classe est un objet d'une classe `MaClasse`

- `MaClasse` est « sérialisable » : ✔
- `MaClasse` n'est pas « sérialisable » : on peut utiliser le mot-clé `transient` pour indiquer de ne pas enregistrer cet attribut

Exemple

```
1  IrreductibleGaulois panoramix =
2          new IrreductibleGaulois("Panoramix", 1.75);
3
4  ObjectOutputStream oos =
5          new ObjectOutputStream(
6            new FileOutputStream(
7              new File("panoramix.txt")));
8
9  oos.writeObject(panoramix);
10 oos.close();
11
12 ObjectInputStream ois =
13         new ObjectInputStream(
14           new FileInputStream(
15             new File("panoramix.txt")));
16
17 IrreductibleGaulois copyPanoramix =
18         (IrreductibleGaulois) ois.readObject();
19 System.out.println(copyPanoramix.nom);
20 ois.close();
```

N.B. Le code n'est pas correct (gestion des exceptions)

Lire depuis la console, afficher sur la console

- `System.in` :
  - entrée « standard »
  - objet de type `InputStream`
- `System.out` :
  - sortie « standard »
  - objet de type `PrintStream` qui hérite de `OutputStream`

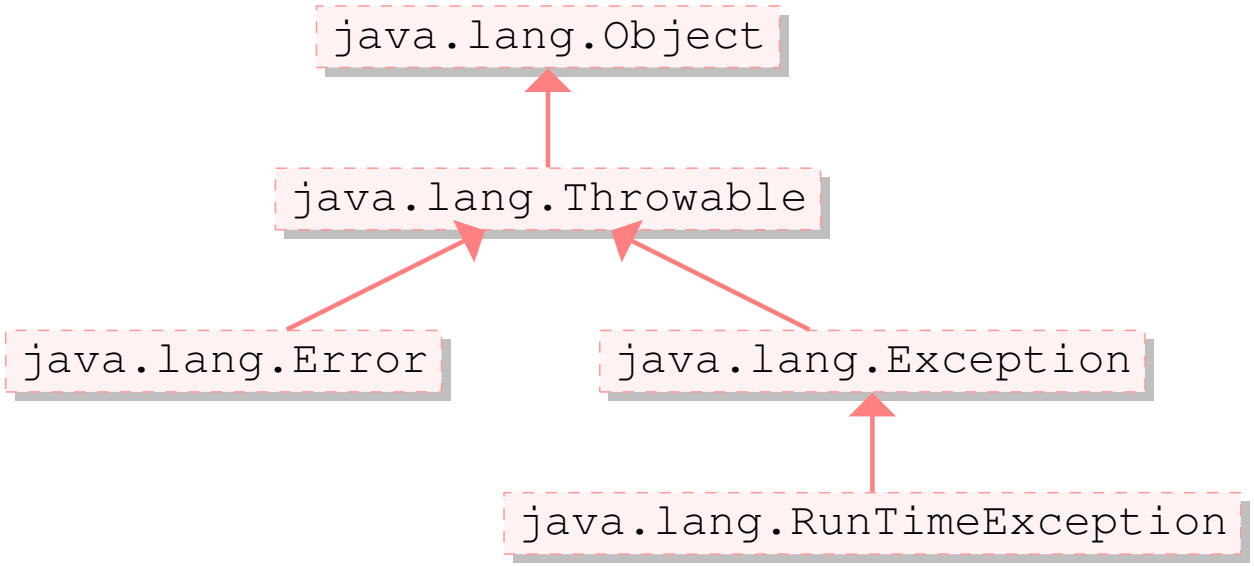
La classe `Scanner` permet de récupérer ce que vous tapez

```
1 Scanner scan = new Scanner(System.in);
2 int n = scan.nextInt();
3 double x = scan.nextDouble();
4 String s = scan.nextLine();
```

Gestion des Exceptions

Gestion des erreurs

`Java` possède un mécanisme de gestion des erreurs, ce qui permet de renforcer la sécurité du code. On peut avoir différents niveaux de problèmes :



- `Error` représente une erreur grave intervenue dans la machine virtuelle (par exemple `OutOfMemory`)
- La classe `Exception` représente des erreurs moins grave

➥ le développeur a la possibilité de **gérer** de telles erreurs et **éviter** que l'application ne se termine

Levée d'exception

Lors de la détection d'une erreur

- un objet qui hérite de la classe `Exception` est créé
- ⇨ ce qui s'appelle **lever une exception**
- l'exception est propagé à travers la pile d'exécution jusqu'à ce qu'elle soit traitée.

```
1 int[] tab = new int[5];
2 tab[5]=0;
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
at Personnage.main(Personnage.java:2)
```

```
1 int d=10,t1=5,t2=5;
2 System.out.println("vitesse:" + d / (t2-t1));
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at Personnage.main(Personnage.java:21)
```

Le bloc `try ... catch`

- bloc `try` : le code qui est susceptible de produire des erreurs
- on récupère l'exception créée avec le `catch`.
- on peut avoir plusieurs blocs `catch` pour capturer des erreurs de types différentes.
- en option, on peut ajouter un bloc `finally` qui sera toujours exécuté (qu'une exception ait été levée ou non)

Lorsqu'une erreur survient dans le bloc `try`,

- la suite des instructions du bloc est abandonnée
- les clauses `catch` sont testés séquentiellement
- le premier bloc `catch` correspondant à l'erreur est exécuté.

Exemple

```
1  int d=10,t1=5,t2=5;
2  try{
3    System.out.println("vitesse:" + d / (t2-t1));
4  }
5  catch(ArithmeticException e){
6     System.out.println(" vitesse non valide ");
7  }
8  catch(Exception e){
9     e.printStackTrace();
10 }
```

Créer sa propre exception

- La classe `MyException` hérite de la classe `Exception`.
- Une méthode qui risque lever une exception de type `MyException` l'indique à l'aide de **throws**

```
1  public class PotionMagiqueException extends Exception {
2      public PotionMagiqueException(){
3        super();
4      }
5      public PotionMagiqueException(String s){
6        super(s);
7      }
8  }
9
10 public class GourdePotionMagique {
11    private int quantite, gorgee=2,contenance=20;
12    public GourdePotionMagique(){ quantite=0;}
13
14    public boolean bois() throws PotionMagiqueException {
15      if (quantite-gorgee <0)
16        throw new PotionMagiqueException
17                  (" pas assez de potion magique!");
18    }
19 }
```

Exceptions et entrée/sortie

```
1  try {
2    FileInputStream fis = new FileInputStream(new File("test.txt"));
3    byte[] buf = new byte[8];
4    int nbRead = fis.read(buf);
5    System.out.println("nb bytes read: " + nbRead);
6    for (int i=0;i<8;i++)
7      System.out.println(Byte.toString(buf[i]));
8    fis.close();
9
10   BufferedReader reader =
11       new BufferedReader(new FileReader(new File("test.txt")));
12   String line = reader.readLine();
13   while (line!= null){
14     System.out.println(line);
15     line = reader.readLine();
16   }
17   reader.close();
18 } catch (FileNotFoundException e) {
19   e.printStackTrace();
20 }
21 catch(IOException e){
22   e.printStackTrace();
23 }
```

Ce dont on n'a pas parlé

- générer de la documentation avec `javadoc`
- **packages** et **imports**
- **programmation multi-thread** parallélisme, concurrence
- **Interfaces graphiques**, gestion des évènements
- Communication avec une **base de données**
- Tester le code `JUnit`