CHAPITRE 7 :

LES EXCEPTIONS

Objectifs spécifiques

1. Introduire la notion d'exception, les règles d'utilisation et de définition des exceptions.
2. Distinguer entre les exceptions prédéfinies et les exceptions définies par l'utilisateur.

Eléments de contenu

I. Définition

II. Les exceptions prédéfinies

III. Les exceptions définies par l'utilisateur

IV. Exemple

Volume Horaire :

Cours : 1 heure 30

Travaux Dirigés : 1 heure

7.1 Définitions

La notion d'exception est offerte aux programmeurs Java pour résoudre de manière efficace et simple le problème de la gestion des erreurs émises lors de l'exécution d'un programme.

Une exception est un signal qui indique qu'un événement anormal est survenu dans un programme

- La récupération (le traitement) de l'exception permet au programme de continuer son exécution.
- Une exception est un signal déclenché par une instruction et traité par une autre
    - Il faut qu'un objet soit capable de signaler ou lever (throw) une exception à un autre objet
    - Il faut que l'autre objet puisse saisir (catch) une exception afin de la traiter

Lorsque l'exception se produit le contrôle est transféré à un *gestionnaire d'exceptions*

- Séparation de l'exécution normale de l'exécution en cas de condition anormale

Exemple

Tester cet exemple en modifiant la valeur de j par **0**

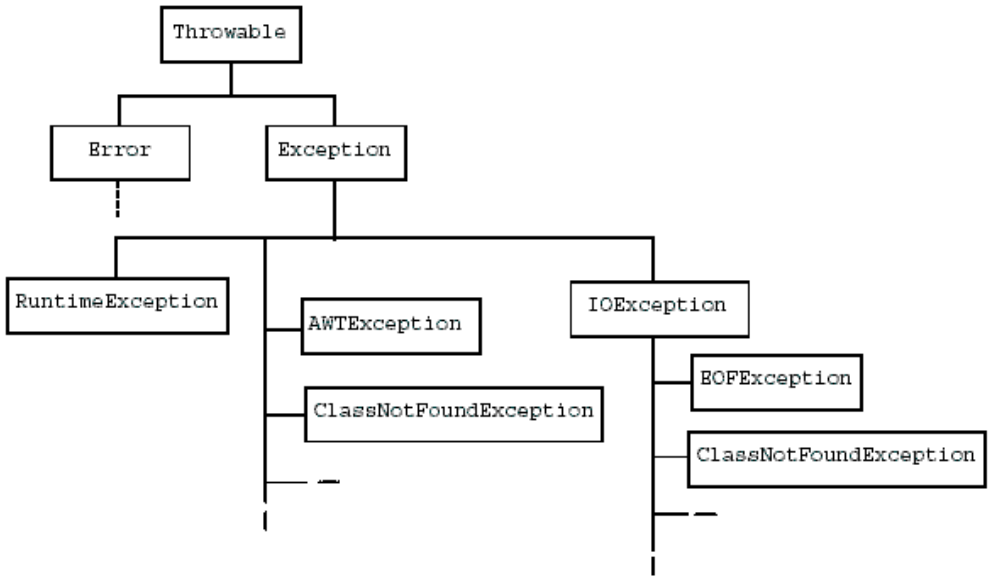
```
public class TestException {
  public static void main(java.lang.String[] args) {

     int i = 3;
     int j = 1;
```

```
      try {
        System.out.println("résultat = " + (i / j));
       }
      catch (ArithmeticException e) {
         System.out.println("erreur division par zéro");

     }
  }
}
```

Une méthode qui désire gérer ce genre de problèmes peut indiquer qu'elle est disposée à intercepter l'exception (instruction **throws**).

7.2 Les exceptions prédéfinies

- En Java, les exceptions sont de véritables objets.
- Ce sont des instances de classes qui héritent de la classe **Throwable**.
- Lorsqu'une exception est levée, une instance de la classe **Throwable** est créée.Voici un aperçu de la hiérarchie des classes pour les exceptions**.**



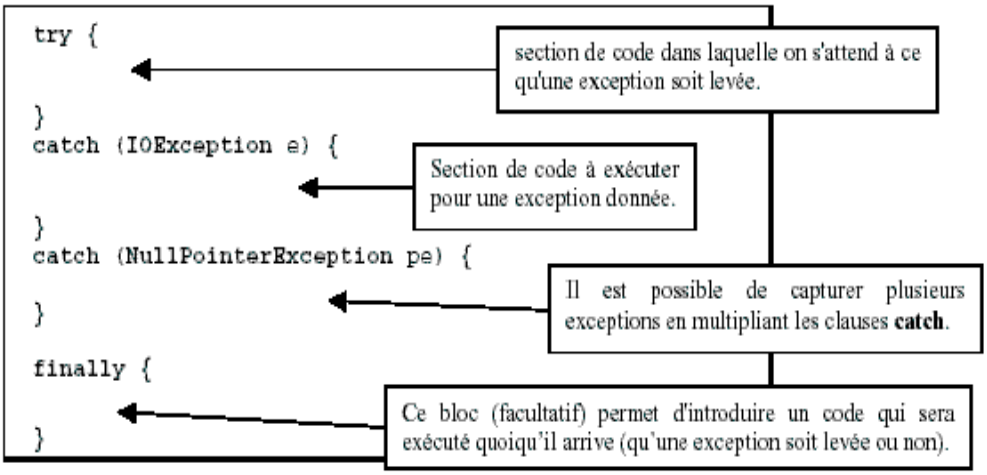
Erreurs et Exceptions

• Erreur: indications de problèmes irrécupérables dus au matériel ou au système d'exploitation

• Exception: erreurs résiduelles dans le programme qui peuvent être traitées par une sortie propre ou une récupération (arithmétique, pointeurs, index, i/o, etc.)

- Les instances de la classe **Error** sont des erreurs internes à la machine virtuelle Java. Elles sont rares et fatales.
- Les sous-classes de la classe Exception sont réparties en deux catégories :
- Les exceptions d'exécution (runtime) sont souvent l'effet du manque de robustesse du code. Par exemple l'exception NullPointerException est levée lorsque l'on manipule un objet non instancié (oubli de l'instruction new) ;
- Les autres exceptions correspondent à des événements anormaux échappant au contrôle du programme. Par exemple, l'exception EOFException est levée si on essaie de lire au-delà d'un fichier.

➔ Traiter les exceptions levées : les mots clés **try, catch et finally :**



➔ Intercepter une exception : le mot clé **throws** : Si une méthode est susceptible de lever une exception et si elle ne peut pas la traiter, elle se doit de prévenir le système qu'elle relaye cette tâche. Pour ce faire, on utilise le mot clé **throws** dans la définition de la méthode. Ce mot clé permet d'avertir le système qu'une certaine catégorie d'exception ne sera pas traitée en local (dans l'exemple suivant, l'ensemble des exceptions liées aux entrées/sorties).

```
public void ma_methode (int x) throws IOEXception {
…
}
```

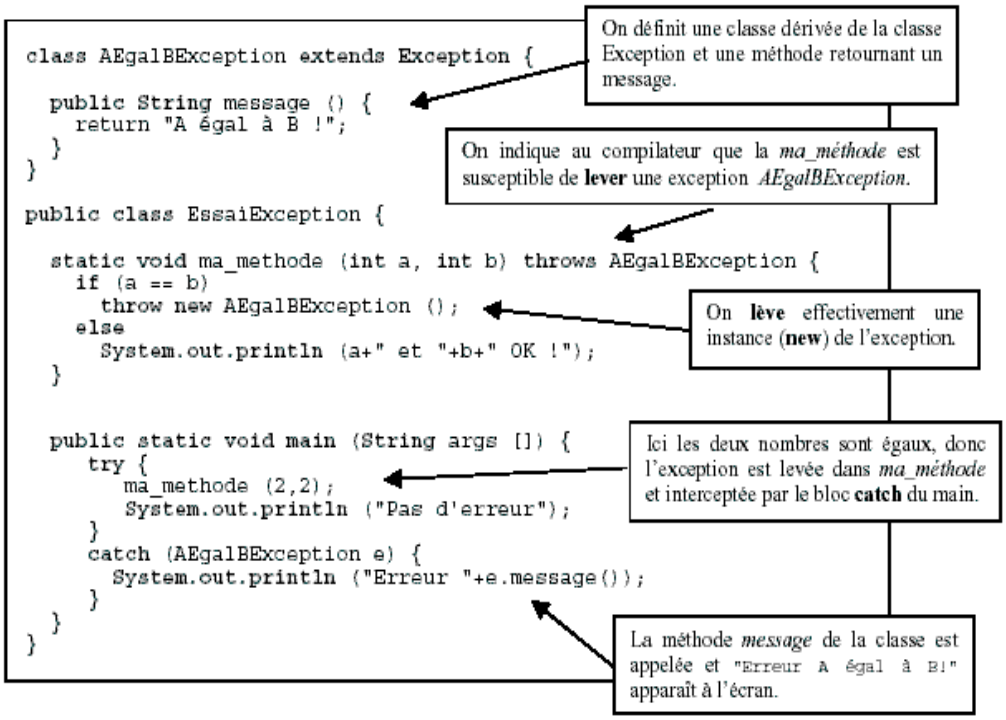
➔ Il est également possible de désigner l'interception de plusieurs exceptions :

```
public void ma_methode (int x) throws IOException, EOFException{
…
}
```

7.3 Les Exceptions définies par l'utilisateur

Jusqu'à présent on parlé des exceptions prédéfinies qui se déclenchent toutes seules. Java offre au programmeur la possibilité de définir ses propres exceptions. Ces exceptions doivent hériter d'une autre exception de la hiérarchie des classes Java. Le programmeur doit lui-même lever ses exceptions.

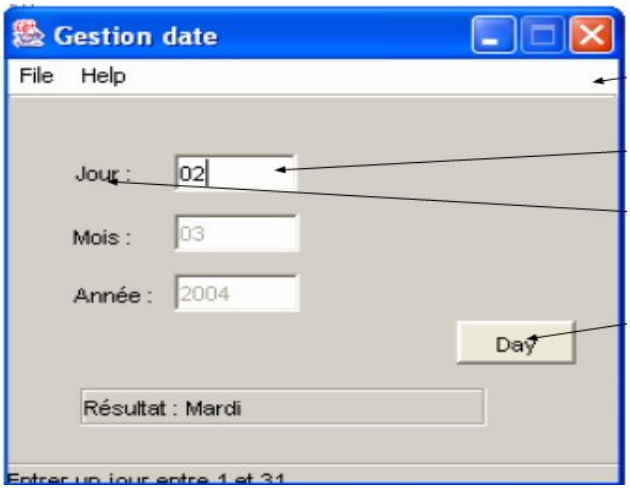
Pour se faire Java met à sa disposition le mot-clé **throw** (à ne pas confondre avec **throws**). Pour le reste (**try, catch, finally**) le mécanisme est identique**.**




```
class AEgalBException extends Exception {

  public String message () {
    return "A égal à B !";
  }
}

public class EssaiException {

  static void ma_methode (int a, int b) throws AEgalBException {
    if (a == b)
      throw new AEgalBException ();
    else
      System.out.println (a+" et "+b+" OK !");
  }

  public static void main (String args []) {
     try {
        ma_methode (2,2);
        System.out.println ("Pas d'erreur");
     }
     catch (AEgalBException e) {
       System.out.println ("Erreur "+e.message());
     }
  }
}
```

7.4 Exemple

Voici un exemple avancé réalisé par l'éditeur Jbuilder, c'est en fait une interface utilisateur qui permet l'insertion d'une date, en faisant les contrôles nécessaires pour afficher l'intitulé du jour.



Appel à l'exception

```
public static void controlmois(int jour,int mois) throws   SaisieDateException {
    if (mois<1||mois>12)
     { throw new SaisieDateException("Mois invalid.");   }

          else{
              if((mois==4||mois==6||mois==9||mois==11)&&(jour==31)){
                throw new SaisieDateException("le jour du mois est invalid.");
                 }
              else if(mois==2&&(jour==31||jour==30)){
                throw new SaisieDateException("le jour du mois est invalid.");

                }
}
```

Déclaration de l'exception

```
public class SaisieDateException extends Exception {
 public SaisieDateException(String s) {
 super(s);
 }
```

Exercice

Ecrire une méthode lireNombre() demandant un nombre à l'utilisateur, le lisant sous forme de chaîne de caractère et le transformant ensuite en entier. La méthode doit recommander un nombre tant que l'entrée est invalide.

NB : Les exceptions susceptibles d'être renvoyées par readline() est IOException et par parseInt() est NumberFormatException

Solution :

```
import java.io.*;
public class Clavier {
//Méthode qui permet de lire une chaine de caractères
public static String lireString ()
{String ligne_lue = null;
try
{InputStreamReader lecteur = new InputStreamReader(System.in);
BufferedReader entree = new BufferedReader(lecteur);
ligne_lue = entree.readLine();
}
catch(IOException err)
{System.exit(0);
}
return ligne_lue;}

//Méthode qui permet de lire un entier
public static int lireInt()
{int x=0;
try
{String ligne_lue = lireString();
x=Integer.parseInt(ligne_lue);
}
catch (NumberFormatException err)
{System.out.println ("***Erreur de données***");
System.exit(0);
}
return x;
}
}

public class test {
 public static void main(String[] argv) {
   int x= clavier.lireInt();
    System.out.println(">>>>>>>   "+ x);
                                }         }
```