Chapitre 2

Les Exceptions

Le mécanisme des exceptions en Java
L'utilisation des exceptions prédéfinis
La création de ses propres exceptions

1. Introduction

Les exceptions est un mécanisme particulier que tous les langages ne possèdent pas.

Ce mécanisme essaye de donner une solution efficace de gestion des cas d'erreur que tout programme informatique peut provoquer.

JAVA intègre un mécanisme d'exception basé sur les principes des langages orientées objet :

- une exception **est un objet**, c'est-à-dire une instance d'une classe des APIs JAVA ou d'une classe développée
- une classe d'exception hérite toujours d'une autre classe d'exception
- les caractéristiques d'une exception sont gérées par des méthodes de la classe d'exception

Avant de d'aller plus loin, il ne faut définir ce que l'on entend par un cas d'erreur car bien que les exceptions soient utilisées pour gérer les cas d'erreur, toute exception n'est pas nécessairement représentative d'un cas d'erreur. Nous verrons que l'exception est un moyen nominal de "retourner" une information.

2. La gestion des cas d'erreur

En informatique un cas d'erreur est un comportement "anormal" d'un traitement.
Cela est tout à fait subjectif car une anormalité peut être vue comme un cas particulier du traitement ou comme un cas d'erreur.
On définit généralement un cas d'erreur comme un comportement non nominal d'un traitement.
Exemple, ajouter un élément dans un tableau alors que le tableau est plein peut être vu comme un cas d'erreur ou comme un cas attendu et traiter et donc normal.

Deux cas d'erreur :

- le traitement en erreur est capable de signaler son erreur (Cas 1)
- le traitement en erreur n'est pas capable de signaler son erreur (Cas 2)

2.1. Cas 1 (Cas JAVA)

C'est l'appelant, d'un traitement en erreur, qui décidera si cela est une erreur ou pas. Tout dépendra de ce que fera l'appelant.

L'appelant a alors deux alternatives :

- soit il **traite** l'erreur remontée par le traitement et alors l'erreur est traitée
- soit il ne sait pas quoi faire et alors il **remonte** à son tour une erreur à son appelant

2.2. Cas 2

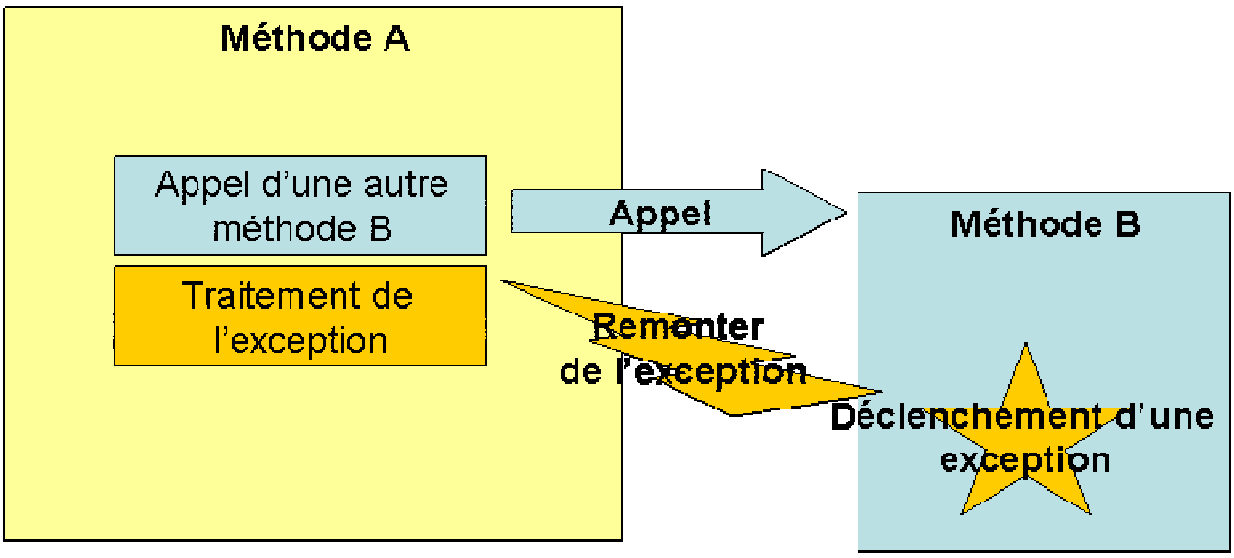
Dans les langages informatiques en général, il existe ce cas d'erreur.
C'est quand le programme s'arrête brutalement sans indications particulière (core) comme cela est le cas dans les langages C ou C++ par exemple.
Dans ce cas de figure, on utilise le debugeur pour localiser l'erreur ou l'interrogation de la pile d'exécution.
En Java, ce type de cas d'erreur n'existe pas car une méthode Java remonte toujours une exception.

De plus, dans les langages qui ne possèdent pas de mécanismes d'exception, il faut donc programmer soi-même le traitement et la remontée des erreurs en utilisant une valeur de retour du traitement afin d'indiquer l'erreur. On parle de retour du code d'erreur.

3. Le mécanisme d'exception en Java

3.1. Principe général

Le principe général est le suivant :
- une méthode A appelle une méthode B
- le traitement de la méthode B ne peut pas se faire correctement, la méthode B retourne une exception
- la méthode A **récupère** alors l'exception retournée afin de traiter le cas d'erreur.



Pour réaliser ce mécanisme, le langage Java utilise une syntaxe particulière :
le **try-catch.**

3.2. Récupérer une exception prédéfinie de Java

Dans la documentation JAVA est indiqué pour chaque méthode quels sont les exceptions que la méthode peut déclencher.

Exemple pour la méthode Integer.parseInt que l'on connait.

parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value or an ASCII plus sign '+' ('\u002B') to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the parseInt(java.lang.String, int) method.

Parameters:

s - a String containing the int representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

NumberFormatException - if the string does not contain a parsable integer.

Exemple :
Tester si une chaîne de caractère est un entier.
L'instruction Java **Integer.parseInt** permet de convertir une chaîne en entier. Si la chaîne n'est pas un entier alors elle déclenche une exception : **NumberFormatException**

```
String str;
int n;
boolean estUnEntier;

str = Terminal.lireString();

estUnEntier = true;
try{
   n = Integer.parseInt(str);
}catch (NumberFormatException ex)
{
   estUnEntier = false;
}

if (estUnEntier)
   Terminal.ecrireStringln("OK");
else
   Terminal.ecrireStringln("NOK");
```

Quand on ne sait pas quel est exactement l'exception retournée par une méthode prédéfinie (ou par "flemme"), on peut utiliser l'exception : **Exception**.

Le code précédent devient :

```
String str;
int n;
boolean estUnEntier;

str = Terminal.lireString();

estUnEntier = true;
try{
   n = Integer.parseInt(str);
}catch (Exception ex)
{
   estUnEntier = false;
}

if (estUnEntier)
   Terminal.ecrireStringln("OK");
else
   Terminal.ecrireStringln("NOK");
```

Et cela marche dans tout les cas d'erreur !!! Donc on ne s'en prive pas.

Cela est possible car **NumberFormatException** hérite de **Exception** :
Toutes les exceptions héritent de Exception.

java.lang

Class NumberFormatException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IllegalArgumentException
          java.lang.NumberFormatException

Exemple :

Faire une méthode qui saisie un entier avec un texte d'invitation, et redemande la saisie si la valeur saisie n'est pas un entier.

Tester la méthode dans un programme.

```
public class ExceptionSaisirEntier
{
    public static void main(String a_args[])
    {
        int n = saisirEntier("Entrez la valeur de n : ");
        Terminal.ecrireStringln("La valeur saisie est : " + n);
    }

    static int saisirEntier(String texte)
    {
        int res=0; // On initialise res sinon erreur de compilation
        boolean ok;

        do{
            ok = true;
            Terminal.ecrireString(texte);
            String chaine = Terminal.lireString();
            try{
                res = Integer.parseInt(chaine);
            }catch(Exception ex)
                {
                  Terminal.ecrireStringln("Erreur. Saisissez un
entier.");
                  ok=false;
                }
        }while(! ok);

        return(res);
    }
}
```

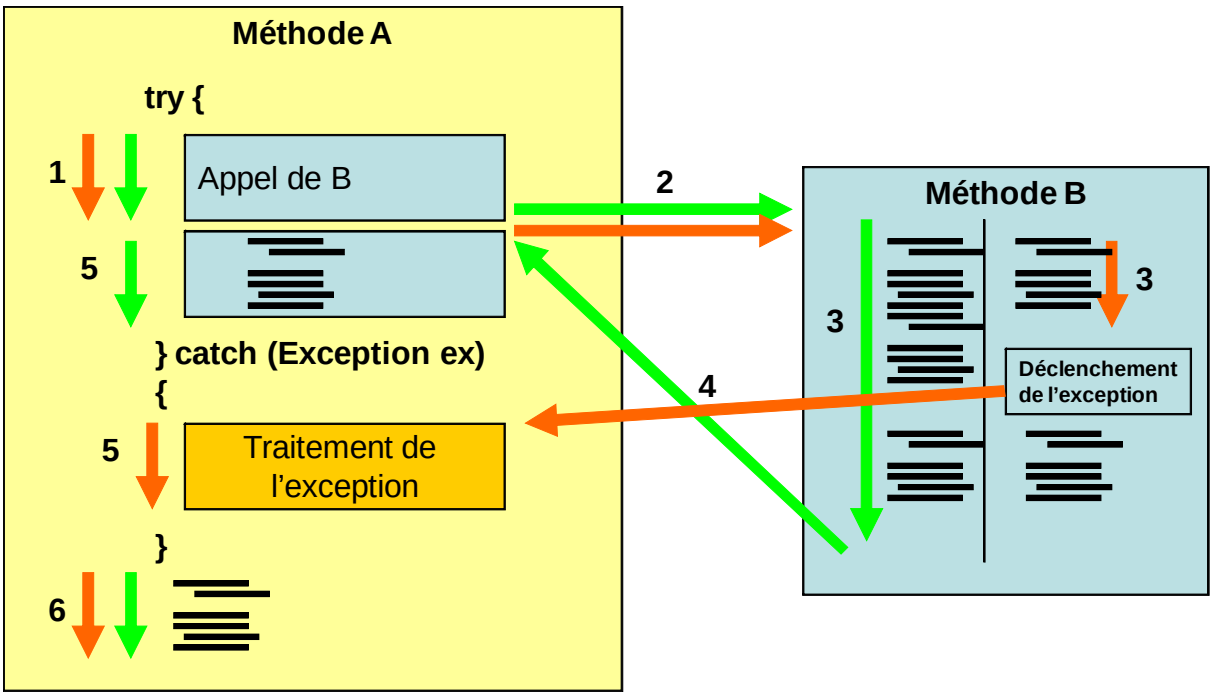
Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple06_SaisirEntier**

3.3. Le flux d'exécution en cas d'exception

Le bloc d'instruction try{ }catch est un bloc d'instruction comme un autre.

Il peut donc contenir plusieurs lignes d'instruction.

Dans le cas où il y a des lignes de code après l'appel de la méthode qui déclenche une exception, ces lignes ne sont pas exécutées.



Lorsque la méthode B ne déclenche pas d'exception, le code exécuté est celui indiqué en vert. Dans ce cas le traitement d'exception n'est pas exécuté.

Lorsque la méthode B déclenche une exception, le code exécuté est celui indiqué en rouge. Dans ce le traitement se trouvant après l'appel de la méthode B n'est pas exécuté et le traitement d'exception est exécuté.

Exemple :
Faire un programme qui prend en entrée une série de double et calcule la moyenne.

```
  public class Test2
  {
      public static void main(String args[])
      {
          double tab[];
          double somme = 0.0;
          double moyenne;

          try{
              tab = new double[args.length];
              for(int i=0;i<args.length;i++)
                  tab[i] = Double.parseDouble(args[i]);  // Exception si
pas double

              for(int i=0;i<tab.length;i++)
                  somme = somme + tab[i];

              moyenne = somme / tab.length;

              Terminal.ecrireStringln("Moyenne = " + moyenne);
```

```
        }
        catch(Exception ex)
            {
                Terminal.ecrireStringln("Erreur de parametre");
            }
    }
}

java Test2 21 2.5 31 2 3
Moyenne = 11.9

java Test2 2 4 x 3
Erreur de paramètre
```

3.4. Récupérer plusieurs exception prédéfinies

Dans le cas où un bloc d'instruction est susceptible de retourner plusieurs exceptions, on peut capturer les différentes exceptions en écrivant plusieurs zones de catch.

Ces différentes zones de catch sont pris en compte dans l'ordre (de bas en haut). Le premier catch qui correspond à l'exception déclenchée est utilisé. Les zones de catch qui suivent, ne sont plus pris en compte.

Voir sur le site http://jacques.laforgue.free.fr cours NFA 032
l'exemple **Exemple39_Exceptions**

```
public class Exemple39
{
 public static void main(String[] args)
    {
        for(int i=0;i<3;i++)
            {
                System.out.println("Cas : "+i);
                String str;
                int k;
                Integer entier;

                if (i==0) str="toto";
                else str="123";

                if (i==1) k=10;
                else k=1;

                if (i==2) entier=null;
                else entier = new Integer(123);


                try{
                    // Conversion d'un entier
                    int x = Integer.parseInt(str);
                    System.out.println("x="+x);
                    // Affectation d'un tableau
                    int[] tab = new int[3];
                    tab[k] = 12;
                    System.out.println("tab["+k+"]="+tab[k]);
                    //Acces a un objet
                    int y = entier.intValue();
                    System.out.println("entier: "+y);
                }
                catch(NumberFormatException ex)
                    {
                        System.out.println("Catch NumberFormatException");
                        System.out.println(ex);
                    }
                catch(NullPointerException ex)
                    {
                        System.out.println("Catch NullPointerException");
                        System.out.println(ex);
                    }
                catch(Exception ex)
                    {
                        System.out.println("Catch Exception");
                        System.out.println(ex);
                    }

                System.out.println("-------------------------");
            }
    }
}
```

Exécution :

```
Cas : 0
Catch NumberFormatException
java.lang.NumberFormatException: For input string: "toto"
--------------------------
Cas : 1
x=123
Catch Exception
java.lang.ArrayIndexOutOfBoundsException: 10
--------------------------
Cas : 2
x=123
tab[1]=12
Catch NullPointerException
java.lang.NullPointerException
--------------------------
```

3.5. La classe Exception et la pile de trace

Une exception est un objet dont la classe d'appartenance est la classe Exception ou une classe qui hérite de la classe Exception.

Les méthodes que l'on peut utiliser sur une exception sont principalement les suivantes :

- String **toString**() :

  Retourne l'exception sous la forme d'une chaine de caractère

- String **getMessage**()

  Retourne le message de l'exception

- void **printStackTrace**()

  Affiche dans le flots de sortie standard la pile de l'exception

- void **printStackTrace**(PrintWriter p)

  Ecrit la pile d'exception dans un PrintWriter. Cela permet de récupérer dans une chaine String, la pile de l'exception

- StackTraceElement[] **getStackTrace**()

  Retourne les éléments de la pile de l'exception

Voir sur le site http://jacques.laforgue.free.fr cours NFA 032
l'exemple **Exemple40_ExceptionsMethodes**

Comme on le voit dans cet exemple, une exception qui remonte garde en mémoire chaque endroit du code qui remonte l'exception ou une nouvelle exception :

```
java.lang.NumberFormatException: For input string: "toto"
 at java.lang.NumberFormatException.forInputString(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at Exemple2.convEntier(Exemple40.java:61)
 at Exemple1.traitement(Exemple40.java:54)
 at Exemple40.main(Exemple40.java:16)
```

Cela est très important car l'analyse de cette pile de trace permet d'identifier le lieu exact de déclenchement de l'exception mais aussi les différents appelants ce qui permet d'identifier le vrai responsable de l'erreur.

Si l'exception n'est jamais capturée alors elle remonte jusqu'à la méthode main et la pile de trace est affichée par la JVM dans la console de lancement du programme Java.

3.6. *L 'exception* Exception

Quand on écrit une méthode, il est important de penser à ses propres cas d'erreur.
Pour cela, on a besoin de pouvoir déclencher une exception :
l'instruction **throw.**

Il existe deux constructeurs de la classe Exception :

```
throw new Exception();
```

ou

```
throw new Exception("message d'erreur");
```

La méthode qui déclenche une exception **doit préciser qu'il peut remonter une exception** :
l'instruction : **throws**.

```
public class Exemple
{
    private String
    public void traitement() throws Exception
    {
       […]
         throw new Exception("Message d'erreur");
    }
}
```

Exemple :
Reprenons l'exemple sur la Bibliotheque de MultiMedia et utilisons les exceptions pour traiter les cas d'erreur de :

- erreur de fichier inconnu
- erreur de paramètre en entrée du programme
- erreur de décodage du fichier contenant les medias.

Voir sur le site http://jacques.laforgue.free.fr cours NFA 032
l'exemple **Exemple16_Bibliotheque**

Commentaires en cours.

3.7. L'exception RuntimeException

La classe RuntimeException hérite de la classe Exception.
C'est donc une exception.

Mais, elle a un rôle particulier : elle surcharge la classe Exception afin que le développeur ne soit pas obligé d'indiquer par l'instruction throws que la méthode peut retourner une telle exception.

Cela explique pourquoi dans nos premiers développement nous n'avons pas eu besoin d'ajouter l'instruction" throws Exception" à chacune de nos méthodes pour les erreurs "communes" de Java : IndexOutOfBoundsException, NullPointerException, …

Un développeur peut donc lui aussi créer une erreur dans ce cas de figure :

```
throw new RuntimeException()
```

```
throw new RuntimeException("Message d'erreur");
```

4. Créer ses propres exceptions

Nous avons vu que nous pouvions créer son propre cas d'erreur en utilisant la classe Exception en faisant : throw new Exception("Message d'erreur").

Mais cette exception n'est pas reconnaissable parmi les autres. S'il faut réaliser un traitement particulier lors de la capture de nos exceptions, il faut créer ses propres classes d'exception.

Cela se fait simplement, il suffit de créer une classe d'exception qui hérite de Exception ou de RuntimeException.
L'exception étant une classe et comme elle doit être publique pour être utilisée par tous, il faut faire autant de fichier .java que de classe d'exception.

On crée un fichier de nom MonException.java dans le code est :

```
public class MonException extends Exception {}
```

On peut alors déclencher son exception est écrivant :

```
throw new MonException();
```

OU

```
public class MonException extends Exception {

   public MonException(String texte)
   {
      super(texte);
   }
}
```

On peut alors déclencher son exception est écrivant :

```
throw new MonException("Message d'erreur");
```

Une classe d'exception étant une classe comme une autre, on peut rendre plus complexe le codage de sa classe d'exception et notamment faire remonter avec le message d'erreur des objets applicatives afin de remonter le maximum d'information.

```
public class MediaException extends Exception
{
     private Media objet;
     public MediaException(String message, Media m)
      {
         super(message);
         objet = m;
       }
     public Media getMedia(return objet;}
}
```

On déclenche l'exception ainsi :

```
throw new MediaException("Impossible d'emprunter",m)
```

On capture cette exception ainsi :

```
try{
```

```
    un traitement d'emprunt de la biblliotheque
}
catch(MediaException ex){
     String s = ex.getMessage();
     Media m = ex.getMedia();
     […]
}
catch(Exception ex){
    pour traiter toute autre exception
}
```

Si on veut par exemple écrire dans un fichier log toutes les exceptions déclenchées dans un programme, il faut par exemple :

- créer une classe racine d'exception, exemple BibliothequeException qui gère le fichier avec une méthode protected qui écrit le message d'erreur dans le fichier
- créer toutes ses classes d'exception qui hérite de la classe BibliothequeException et qui utilise au passage la méthode pour écrire dans le fichier.

Exercice : modifier l'exemple Exemple16_Bibliotheque pour implémenter ce mécanisme.

Vous ferez cet exercice une fois le cours sur les entrée-sorties acquis.