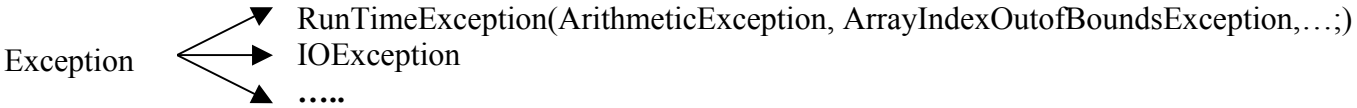
La gestion des exceptions

Une *exception* est une interruption de l'exécution d'un programme suite à une erreur. Par exemple, une division par zéro provoque une exception de type `ArithmeticException`. Java permet non seulement la gestion des exceptions, mais aussi la création d'exceptions spécifiques.
Les exceptions Java sont des objets à part entière qui peuvent porter de l'information. Ces objets doivent être instances de la classe `Throwable` ou d'une de ses sous-classes.

Classe Throwable
- Constructeur
  public Throwable();
  public Throwable(String message);
- Méthodes
  public String getMessage(); //renvoie le message détaillé associé à l'objet
  public void printStackTrace();// imprime sur la sortie standard le nom de l'exception et la trace de l'exception
  public Throwable fillInSteackTrace();// réinitialise la trace de la pile
  public String toString(); // méthode héritée de la classe Object, renvoyant une description //sommaire de l'exception

La classe java;lang.Throwable possède 2 sous classes dérivées
  Classe java.lang.Exception (pour des événements inattendus, qui seront souvent traités de sorte qu'elle ne provoque pas l'arrêt du programme)
  Classe java.lang.Error (pour des erreurs graves, qui devront généralement conduire à l'arrêt du programme)




```
class Test {
   public static void main (String argv[] ) {
   int []a ={2,3,5};
   System.out.println(a[3]);
   }
}
```

La Compilation se fait mais erreur d'exécution
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: **3** at **Test.main**(Test.java:**4**)

```
class TaperTouche {
   public static void main (String argv[] ) {
   System.out.println("Taper une touché pour terminer");
   System.in.read();
   }
}
```

La compilation est impossible

```
TaperTouche.java :6 unreported exception java.io.IOException, must be caught
or declared to be thrown
```

En effet la methode read() de l'objet in est susceptible de lever une exception d'entree/sortie (IOException)

Comment corriger?

```
import java.io.IOException;
class TaperTouche {
   public static void main (String argv[] ) {
     System.out.println("Taper une touché pour terminer");
     try{
        System.in.read();
     }
     catch (IOException e) {
        System.out.println("Erreur IOException detecte");
     }
   }
}
```

1. Utilisation des exceptions

Avant de voir comment traiter les exceptions, regardons à quoi elles correspondent concrètement, en prenant un exemple très simple où on génère une exception de division par zéro :

```
// DivParZero.java
//
class DivParZero {
   public static void main (String argv[] ) {
   int val=0;
   val = 1997/val;
   System.out.println("Fin du programme");
   }
}
```

En exécutant ce programme, on obtient l'affichage suivant : (message par défaut)

```
java.lang.ArithmeticException: / by zero
at DivParZero.main(DivParZero.java:6)
```

On distingue:

- le nom complet de l'exception qui a été *levée*,
- un message précisant la cause de cette erreur est envoyé par l'objet de type exception (`/by zero`),
- l'indication de la classe, de la méthode et du numéro de ligne où s'est produite cette exception.

Nous avons vu plusieurs fois, dans les chapitres précédents, des constructions du type **try** et **catch()**. Ces mots clés permettent de gérer (*intercepter*) des exceptions afin de provoquer l'exécution de certaines instructions spécifiques. Examinons de plus près comment s'utilisent ces mots clés :

```
try {
  // zone contenant des instructions
  //pouvant lever des exceptions
}
catch (NomException e) {
  // traitement de l'exception
}
```

Un bloc **try{ }** contient donc un ensemble d'instructions qui peuvent lever des exceptions durant leur exécution. On indique donc que l'on va gérer tout ou partie de ces exceptions. Cette gestion s'effectue dans un ou plusieurs blocs **catch() { }**. Dans un tel bloc, on *attrape* une exception précisée dans l'instruction **catch()**. Il est important de noter qu'une exception est également un objet en Java. Nous verrons plus loin comment créer un objet de ce type.

On peut donc définir autant de bloc **catch()** qu'il y a d'exceptions susceptibles d'être levées. Reprenons notre exemple de division par zéro en implémentant une gestion de l'exception **ArithmeticException** :

```
// DivParZero.java
//

class DivParZero {
  public static void main (String argv[] ) {
    int val=0;
    try {
      val = 1997/val;
    }
    catch (ArithmeticException e ) {
      System.out.println("Une exception arithmétique a été levée");
      System.out.println("Message : " + e.getMessage());
      System.out.println("Pile :");
      e.printStackTrace();
   }
   System.out.println("Fin du programme");
  }//fin de main
}//fin de la classe
```

A l'exécution, on obtient :

```
Une exception arithmétique a été levée
Message : / by zero
Pile :
java.lang.ArithmeticException: / by zero
at DivParZero.main(DivParZero.java:5)
Fin du programme
```

Nous avons donc ici installé un *gestionnaire d'exceptions* autour de l'affectation provoquant une division par zéro. Un bloc **catch()** interceptant une exception arithmétique a été défini. Dans ce bloc, nous signalons quelle exception a été levée, puis nous affichons le message associé grâce à la méthode **getMessage()** appliquée à l'objet e du type ArithmeticException. Enfin, nous appelons la méthode **printStackTrace()** qui va nous renvoyer le message affiché par défaut.
Il faut également savoir qu'il existe un dernier mot clé, **finally.** Il définit un bloc dont les instructions sont systématiquement exécutées, qu'une exception soit levée ou non. Ce bloc vient après le bloc catch().

```
finally {
System.out.println("Fin du calcul");
}
```

Si on exécute le programme ainsi modifié, voilà ce que l'on obtient :

```
Une exception arithmétique a été levée
Message : / by zero
Pile :
java.lang.ArithmeticException: / by zero
at DivParZero.main(DivParZero.java:8)
Fin du calcul
Fin du programme
```

Nous constatons que non seulement l'instruction ajoutée dans le bloc **finally** a été exécutée, mais qu'en plus le programme a repris son cours **après** ce bloc. Nous avons donc pu non seulement intercepter l'exception, mais également poursuivre l'exécution du programme.

Le mécanisme d'exception permet donc de gérer des erreurs d'exécution, sans nécessité de vérifier systématiquement la cohérence de certaines données ou de regarder la valeur de retour d'une fonction. Cela simplifie donc la programmation et augmente la lisibilité du code.

En consultant la documentation de l'API Java livrée avec le JDK, vous trouverez que, pour chaque package, il est indiqué une liste d'exceptions susceptibles d'être levées par les différentes méthodes des classes contenues dans ces packages. Avant de voir comment créer vos propres exceptions, signalons quelques points importants :

- une seule exception peut être levée au même moment,
- en utilisant un bloc catch du type **catch (Exception e)**, on intercepte toutes les exceptions pouvant se produire.

```
class DivParZero {
  public static void main (String argv[] ) {
    int val=0;
    try {
          val = 1997/val;
    }
    catch (Exception e ) {
             System.out.println("Pile :");
       }
    catch (ArithmeticException e ) {
     System.out.println("Une exception arithmétique a été levée");
     System.out.println("Message : " + e.getMessage());
     System.out.println("Pile :");
     e.printStackTrace();
   }
  }
}
```

Un tel segment de code compile-t-il ? Précisez.

Non, car le deuxième catch capture une exception plus spécialisée que la première : le code est inatteignable

2. Déclaration d'exceptions

Avant de voir comment déclarer une exception dans une méthode, il faut savoir comment en lever une. Java définit pour ce faire un mot clé **throw** qui permet de lever une exception. Par exemple, si nous ajoutons dans notre programme la ligne suivante juste avant le calcul provoquant l'exception arithmétique (dans le bloc **try**):

```
if (val==0) throw new ArithmeticException ("Division par zéro");
```

On obtient :

```
Une exception arithmétique a été levée
Message : Division par zéro
Pile :
java.lang.ArithmeticException: Division par zéro
at DivParZero.main(DivParZero.java:8)
```

Cette fois-ci, nous avons nous même levé cette exception, en indiquant un message précisant la nature exacte de l'erreur, redéfinissant l'exception par défaut.

Regardons maintenant comment définir notre propre exception. En fait, puisqu'une exception est un objet en Java, il va falloir créer une classe spécifique, héritée de la classe **Exception**. Ensuite, nous allons créer une fonction qui va lever notre exception. Pour ce faire, il est obligatoire de préciser dans la signature de la fonction la liste des exceptions susceptibles d'être levées. Voici le source de notre exemple :

```
//  NouvelleException.java
class NouvelleException extends Exception {
    NouvelleException () {}
    NouvelleException (String msg) {
        super(msg);
    }
}
```

```
// CreeException.java
public class MonApplication {
```

```
    static void fnct (double val) throws NouvelleException {
     if (val<0) throw new NouvelleException ("Racine d'un nombre negatif");
     double racineCarre =  Math.sqrt(val);
     System.out.println("la racine carre est : " + racineCarre);
    }
    public static void main (String argv[]) {
      try {
          // On appelle la fonction levant l'exception
          fnct(-5.0);
      }
      catch (NouvelleException e) {
        // On attrape cette exception
        System.out.println("Pile:");
        e.printStackTrace();
      }
  }
}
```

Analysons cet exemple. Tout d'abord, nous déclarons notre nouvelle exception en créant une classe dédiée, héritée de la classe **Exception**. On crée deux constructeurs, l'un sans argument qui ne fait rien, l'autre prenant un message en argument. Ce message est transmis au constructeur de la classe parente, grâce au mot clé **super()**.

Ensuite, nous définissons une classe, dans laquelle on crée une méthode levant l'exception que nous venons de définir. Comme nous l'avons dit précédemment, on spécifie quelles exceptions peuvent être levées dans la signature de la fonction, soit ici:

```
static void fnct () throws NouvelleException {
```

Cette fonction se contente de lever l'exception `NouvelleException`.

Enfin, dans la méthode **main()**, nous faisons appel à la fonction précédente en installant un gestionnaire d'exceptions. Le bloc **catch()** attrape notre exception et affiche à l'écran le contenu de la pile d'appel des fonctions :

```
Pile :
NouvelleException: Racine d'un nombre négatif
at CreeException.fnct(CreeException.java:13)
at CreeException.main(CreeException.java:18)
```

Remarquons que notre message spécifié lors de la levée de l'exception a été reproduit.

On note également que la pile d'appel contient deux fonctions, celle dans laquelle on lève l'exception, et celle appelant cette dernière, avec indication des numéros de lignes respectifs.