Les exceptions en java

C'est quoi une exception?

Une exception est un événement (une erreur) qui se produit lors de l'exécution d'un programme, et qui va provoquer un fonctionnement anormal (par exemple l'arrêt du programme) de ce dernier.

Dans le programme suivant vu que la variable b était nulle, le programme va générer une erreur lors de l'exécution, car une division par zéro vient d'avoir lieu.

```
 1 public class Div1 {
 2         public static int divint (int x, int y) {
 3             return (x/y);
 4   }
 5         static void main (String [] args) {
 6             int c=0,a=1,b=0;
 7             c= divint(a,b);
 8             System.out.println("res: " + c);
 9             System.exit(0);
10         }
11 }
```

Le système affiche l'erreur suivante:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Div.divint(Div.java:3)
        at Div.main(Div.java:7)
```

Une erreur s'est produite à la ligne 7 de la méthode main (la méthode divint), plus exactement à la ligne 3 (la division de x/y) ; elle correspond à une division par zéro.

Des erreurs peuvent se produire dans d'autres circonstances, comme une ouverture d'un fichier inexistant, l'index d'un tableau qui déborde etc.

Gestion des exceptions

La gestion des exceptions se substitue en quelque sorte à l'algorithmique permettant la gestion des erreurs. Dans l'exemple précédent, si nous avons voulu anticiper sur la gestion de l'erreur, il fallait prévoir un traitement adéquat pour contrer la division par zéro:

```
if (y!=0)
      return(x/y);
else
      // traitement de l'erreur.
```

Le traitement de l'erreur pourrait consister à retourner une valeur: -1 pour une division par zéro, -2 pour un index qui déborde etc. Ce traitement devient fastidieux à la longue!

```
traitement A
if (!traitement_A)
      méthodes de correction
else
      traitement_B
      if (!traitement_B)
            méthodes de correction
      else
            etc.
```

Le langage Java offre un mécanisme très souple pour la gestion des erreurs. Ce mécanisme permet d'isoler d'une part la partie du code générant l'erreur du reste du programme, et d'autre part de dissocier les opérations de détection et de traitement de cette erreur. Par ailleurs, le langage Java utilise des objets pour représenter les erreurs (exceptions) et l'héritage pour hiérarchiser les différents types d'exception.

La gestion des erreurs consiste donc à définir le bloc pouvant provoquer l'erreur (le bloc try), ainsi ce bloc devient isolé du reste du programme ; à lancer (throw) ou attraper (catch) les "objets" représentant les exceptions générées.

```
public class Div2 {
        public static int divint (int x, int y) {
            return (x/y);
        }
        static void main (String [] args) {
            int c=0,a=1,b=0;
            try {
                  c= divint(a,b);
            }
            catch (ArithmeticException e) {
                  System.out.println("Erreur a été capturée");
            }
            System.out.println("res: " + c);
            System.exit(0);
        }
}
```

La liste des exceptions possibles est fournie dans la documentation technique de Java. En effet cette documentation technique fournit pour chaque méthode, en plus de son nom, sa signature et son type de retour ainsi que les exceptions pouvant être générées en cas d'erreur lors de l'utilisation de cette méthode.

Mécanisme du traitement d'exception

Lorsqu'une exception est levée dans une méthode donnée, les instructions qui suivent le lancement de l'exception, se trouvant dans cette méthode, sont ignorées. L'exception peut-être attrapée par un bloc catch (s'il existe un try) se trouvant dans cette méthode. Si l'exception n'a pas été capturée, le traitement de cette exception remonte vers la méthode appelante, jusqu'à être attrapée ou bien on est arrivé à la fin du programme (ici nous nous trouvons au niveau de la méthode main).

1- Une exception est levée …

en dehors d'un bloc try

Dans ce cas, on remonte immédiatement dans la méthode appelante pour lui donner la possibilité de traiter cette exception. Si elle ne peut pas, il y a arrêt du programme.

dans un bloc try

Un bloc try est défini et contient quelques instructions qui risqueraient de lever une ou plusieurs exceptions.

```
try {
        c= divint(a,b);
        System.out.println ("la suite du programme!");
}
```

Si une instruction donnée de ce bloc try (par exemple c=divint(a,b)) ne génère pas une exception, le programme passe au traitement de l'instruction suivante dans ce bloc la (dans le précédent exemple: System.out.println ….).

Par contre, si une des instructions du bloc try provoque une exception (par exemple: c=divint(a,b)), les instructions suivantes du bloc try ne seront pas exécutées (dans le précédent exemple: System.out.println …. ne sera pas exécutée. On sort complètement du bloc try ; et la main sera donnée à un bloc catch …

Dans une méthode, si nécessaire, il peut y avoir plusieurs bloc try.

2- … capturons la …

Un bloc catch sert à capturer un type d'exception donné, par exemple ArithmeticException, IndexOutOfBoundsException etc. et de réaliser des opérations comme:

corriger l'erreur qui a provoqué l'exception, proposer un traitement alternatif, retourner une valeur particulière, sortir de l'application, relancer la même exception, faire qu'une partie du traitement et la méthode appelante fera le reste etc.

Un bloc catch, s'il existe dans un programme, il suit immédiatement un bloc try (donc contigu au bloc try). Il n'est pas indispensable (nous allons voir cela plus loin dans le cours).

Dans le cas où il n'y pas eu d'exception levée par aucune des instructions du bloc try, l'exécution du programme se poursuit après le dernier bloc catch. Comme si aucun bloc catch n'a été défini dans le programme.
Par contre, si une exception est levée, deux cas se présentent:

- s'il existe un bloc catch qui peut capturer cette exception, il sera exécuté en premier, puis le programme poursuit après le dernier bloc catch.

- si aucun bloc catch ne peut capturer cette exception, la main est donnée à la méthode appelante. À elle de voir si elle peut traiter cette exception, sinon on remonte de méthode en méthode à la recherche d'un bloc catch adéquat, jusqu'à terminer le programme (donc niveau main).

- catch peut avoir les deux formats d'appel suivants:

  * **catch (T info)**: capturer une exception du type T, et info est un objet du type T. Cet objet sert à extraire des informations supplémentaires.

-3 … mais peut-on la lever?

Une exception est donc levée si une opération illégale risquerait d'avoir lieu. Pour le faire, nous utilisons la commande **throw** mais il faudra faire attention aux points suivants:

- Comme déjà mentionné au début de ces notes de cours, les exceptions sont représentées en quelque sorte par des "objets", instanciés donc à partir de classes ; de ce fait, chaque exception fait référence à une classe. Pour lever une exception, nous devons créer une instance de la classe où réside cette exception.


```
public class Div3 {
        public static int divint (int x, int y) {
            if (y==0) throw new ArithmeticException();
            return (x/y);
        }
        static void main (String [] args) {
            int c=0,a=1,b=0;
            try {
                  c= divint(a,b);
            }
            catch (ArithmeticException e) {
                  System.out.println("Erreur a été capturée");
            }
            System.out.println("res: " + c);
            System.exit(0);
        }
}
```

- Dans l'exemple précédent, nous avons une exception déjà connue par le mécanisme d'exception. Nous pouvons lever aussi de nouvelles exceptions définies par le programmeur. Il faut savoir, que la catégorie d'exception qui nous concerne le plus (nous allons les voir en détails quand nous allons exposer la hiérarchie des exceptions) porte le nom de **Exception**, à partir d'elle doivent dériver toutes les nouvelles exceptions introduites par le programmeur.

```
class MaArithmeticException extends Exception {
      // quelque chose …
}


public class Div4 {
        public static int divint (int x, int y)
                                    throws MaArithmeticException {

            if (y==0) throw new MaArithmeticException();
            return (x/y);
        }
        static void main (String [] args) {
            int c=0,a=1,b=0;
            try {
                  c= divint(a,b);
            }
            catch (MaArithmeticException e) {
                  System.out.println("Erreur a été capturée");
            }
            System.out.println("res: " + c);
            System.exit(0);
        }
}
```

Si nous avons affaire à une méthode susceptible de lever une exception qu'elle ne traite pas localement, cette méthode doit mentionner son type dans son en-tête en utilisant pour

cela, le mot réservé **throws**. Soit l'exception est traitée localement, soit elle est propagée par throws.

- on peut lever une seconde fois une exception qui a été déjà levée! Comment?

```
class MaArithmeticException extends Exception {
                 // quelque chose
}

public class Div5 {
          public static int divint (int x, int y)
                          throws MaArithmeticException {
                if (y==0) throw new MaArithmeticException();// 1e fois.
                return (x/y);
           }

         public static int madivision(int x, int y)
                          throws MaArithmeticException {
                int z=0;

                try {
                        z=divint(x,y);
                }
                catch (MaArithmeticException e) {
                    System.out.println("Erreur a été capturée" +
                                                    " dans divint");
                    throw e; // on lève une 2nde fois l'exception.
                }
                return z;
        }

        static void main (String [] args) {
            int c=0,a=1,b=0;
            try {
                c= madivision(a,b);
            }
            catch (MaArithmeticException e) {
                System.out.println("Erreur a été capturée dans main");
            }
            System.out.println("res: " + c);
            System.exit(0);
        }
}
```

Très intéressante comme technique si l'on veut continuer à traiter cette exception dans la méthode appelante.

Transmission d'information au gestionnaire d'exception

On peut transmettre de l'information au gestionnaire d'exception soit par le l'objet ayant levé l'exception via throw, sinon par le constructeur de la classe Exception.

- à partir de throw

```
class MaArithmeticException extends Exception {
        public void message(){
               System.out.println("Erreur a été capturée" +
                                    " dans MonArithmeticException");
        }
}

public class Div6 {

          public static int divint (int x, int y)
                                    throws MaArithmeticException {

                if (y==0) throw new MaArithmeticException();
                return (x/y);

         }

        static void main (String [] args) {

            int c=0,a=1,b=0;

            try {
                c= divint(a,b);
            }

            catch (MaArithmeticException e) {
                  e.message();
            }

            System.out.println("res: " + c);
            System.exit(0);
        }
}
```

- à partir de la classe Exception

Exception dérive de la classe Throwable. Un des constructeurs de cette classe accepte une chaîne de caractères qui sert à décrire le problème. Le message associé à ce problème est accessible par la méthode getMessage de la classe Throwable. Ainsi l'exemple précédent devient comme suit:

```
class MaArithmeticException extends Exception {

      MaArithmeticException(String message){
             super(message);
      }

}

public class Div7 {
      public static int divint (int x, int y)
                              throws MaArithmeticException {

             if (y==0) throw new MaArithmeticException("Erreur a été" +
                                " capturée dans MonArithmeticException");
                return (x/y);
      }

      static void main (String [] args) {

            int c=0,a=1,b=0;

            try {
                c= divint(a,b);
            }

            catch (MaArithmeticException e) {
                System.out.println(e.getMessage());
            }

            System.out.println("res: " + c);
            System.exit(0);

        }
}
```

utilisation de finally

Le bloc finally sert plus particulièrement à faire le nettoyage en cours de route (fermer un fichier par exemple). Il est exécuté peu importe l'état des autres blocs try et catch. Il sera donc toujours exécuté, sauf si l'instruction qui le précède est System.exit() (cette dernière permet de sortir complètement du programme).

- Un bloc try peut être suivi d'un seul bloc, ou plusieurs blocs, catch. Le dernier bloc catch peut-être suivi d'un bloc finally. Le bloc try peut-être est suivi directement par le bloc finally tout court (sans bloc catch).

- Si une exception a été levée: Si elle est capturée par un bloc, le bloc finally est exécuté après ce bloc catch. Si l'exception n'a pas été capturée par aucun des blocs catchs présents dans la méthode, finally sera exécutée avant que la main ne soit donnée à la méthode appelante.

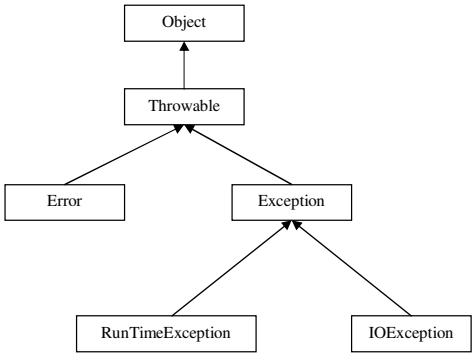
- Si aucune exception n'a été levée, finally est exécutée quand même après la fin du bloc try.

- Si le bloc finally se termine par un return, c'est cette valeur qui sera retournée, peu importe ce qu'il s'est passé dans les blocs try et catch (par exemple, s'il y avait aussi une instruction return dans un bloc try et/ou catch, elle sera ignorée dans ce cas). S'il y a une instruction return que dans le bloc try, finally est quand même exécutée avant que ce "return" soit pris en compte.

- Si dans un bloc try, il y a une commande break, finally sera toujours exécutée, par la suite ca sera le tour des instructions après le bloc finally d'être exécutées.

```
public class Div8 {

      public static int divint (int x, int y) {
         return (x/y);
      }

      static void main (String [] args) {

         int c=0,a=1,b=0;

         try {
             c= divint(a,b);
         }

         catch (ArithmeticException e) {
             System.out.println("Erreur a été capturée");
         }

         finally {
             System.out.println("on est arrivé à la fin du cours!");
         }

         System.out.println("res: " + c);
         System.exit(0);

      }
}
```

Hiérarchie des exceptions



Throwable est la classe de base, à partir de laquelle vont dériver toutes les exceptions.

Error: Elle gère les erreurs liées à la machine virtuelle (LinkageError, ThreadDeath etc.)

Exception: contient l'ensemble des exceptions gérées par le programmeur (ArithmeticException etc.).

RunTimeException: regroupe les erreurs de base (ArithmeticException etc.)

IOException: regroupe les erreurs entrée/sortie.

Error et RunTimeException appartiennent à la catégorie "unchaked" donc "throws" n'est pas nécessaire à coté de la méthode qui lance une exception de cette catégorie là.

Toutes les autres excéptions (y compris donc celles créées par le programmeur) appartiennent à la catégorie des "checked" où l'utilisation de "throws" est exigée.

Le compilateur génère quand même une erreur s'il trouve que l'instruction throws a été omise pour une exception de la catégorie checked. L'erreur générée est comme suit:


```
class MaArithmeticException extends Exception {
      // quelque chose …
}
```

// omission de throws …

```
public class Div9 {
        public static int divint (int x, int y) {

            if (y==0) throw new MaArithmeticException();
            return (x/y);
        }
        static void main (String [] args) {
            int c=0,a=1,b=0;
            try {
                  c= divint(a,b);
            }
            catch (MaArithmeticException e) {
                  System.out.println("Erreur a été capturée");
            }
            System.out.println("res: " + c);
            System.exit(0);
        }
}
```

```
Div9.java:10: unreported exception MaArithmeticException; must be
caught or declared to be thrown
               if (y==0) throw new MaArithmeticException();
```

L'erreur doit être capturée (donc il faudra utiliser un catch) ou bien elle doit être propagée (donc il faudra utiliser throws)