Exceptions

Une exception survient lors d'un cas de figure anormal. Elle interrompt le flot normal d'exécution et exécute les éventuelles routines de traitement. Par exemple, s'il on essaye d'ouvrir en lecture un fichier qui n'existe pas, une exception **IOException** sera générée.

En Java, il existe une classe **Exception**. Si l'on désire créer un nouveau "type" d'exception, il suffit d'étendre la classe, comme présenté ci-dessous :

```
class divisionParZeroException extends Exception
{
}
```

Remarquons que **Exception** et **Error** sont des sous-classes de **Throwable**. Les objets de la classe Error sont des erreurs internes à la machine virtuelle Java, c'est pourquoi il faut être très vigilant si l'on désire les récupérer. En effet, il ne serait pas très judicieux d'ignorer une exception de type "Plus de mémoire".

Si dans un bloc (une méthode), on fait appel à une méthode qui peut potentiellement générer une exception, on doit soit essayer de la récupérer avec les clauses **try** et **catch**, soit ajouter le mot clé **throws** dans la déclaration du bloc. Si on ne le fait pas, il en résultera une erreur à la compilation. Il n'est toutefois pas requis de traiter les exceptions et erreurs déclarées dans le paquetage java.lang, car celles-ci peuvent se déclencher partout.

Pour générer une exception, il suffit d'utiliser **throw** suivi d'un objet dont la classe est dérivée de Throwable.

```
throw new Exception();
throw new IOException("fichier ABSENT.TXT non trouve");
```

Lorsqu' une méthode contient une instruction (éventuellement à travers l'appel à une autre méthode) susceptible de lancer une exception sans l'attraper (avec les clauses **try-catch**), il faut l'indiquer dans la déclaration de méthode, en utilisant le mot clé **throws.**

Attention à l'orthographe, lorsque l'on indique qu'une méthode peut générer une exception, dans la déclaration de la méthode, on utilise le mot clé throws, qui prend un s. Lorsque l'on génére l'exception, on utilise throw, qui ne prend pas de s.

Quand une exception est lancée, toutes les instructions suivantes sont ignorées et on remonte la pile des appels des méthodes : **l'exception se propage**. Pendant la propagation si une méthode de la pile d'appel a été invoquée à l'intérieur d'une clause **catch** :

suivi de la clause **finally** : les instructions de la clause finally sont nécessairement exécutées et la propagation se poursuit

suivi de la clause **catch**, attrapant les exceptions de la classe de l'exception qui se propage, les instructions de la clause catch sont exécutés puis l'exécution reprend son cours normal avec l'instruction qui suit la clause finally

La clause **finally** est en général utilisé pour effectuer des nettoyages (fermer des fichiers, libérer des ressources...).

```
class Demo
{
	public static void main(String arg[])
	{
		DataInputStream s=new DataputStream(System.in);
		do
		{
			 System.out.print(« age: »);
			System.out.flush ();
			try
			{
				String str = s.readLine();
				int age = Integer.parseInt(str);
				if(age> 130)
					throw new Age Over(age++ « ans trop grand »);
			}
```

```
                catch (IOException ioc)
                {
                        System.out.printin (« erreur d'E/S ») ;
                        System.exit(1) ;
                }
                catch (AgeOver a)
                {
                        System.out.printin (a.getMessage()) ;
                        continue ;
                }
                catch (NumberFormatException nbe)
                {
                        System.out.printin (« il faut saisir une valeur entière ! ») ;
                        continue ;
                }
                finally
                {
                        System.out.printin (« Saisie effectuée ») ;
                }
        }while (true) ;
    }
}
```

La classe **Age0ver** est une sous-classe d' **Exception :**

```
class Age0ver extends Exception
{
    Age0ver (String msg) {super (msg); }
}
```

Une méthode qui lance une exception sans la traiter doit le spécifier dans sa déclaration par **throws**

```
class Demo
{
        public boolean saisie () throws Age0ver
        {
            DataInputStream s = new DataInputStream (System.in);
            String str = s.readline 0;
            int age = Integer.parselnt (str);
            if (age > 130)
                    throw new Age0ver (age+ "ans trop grand");
           }
}
```

Les méthodes des classes d'entrées-sorties lancent des exceptions qui sont des instances de la classe **1Oexception :**

```
            DataInputStream dis = new DataInputStream (System.in);
            String buf = null;
            try
            {
                    buf = dis.readline ();
              }
            catch (IOException ioe)
            {  System.out.println (ioe.getMessage (); }
```

Exemple : Division par zero

```
class divisionParZeroException extends Exception
{
}

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

class divMul
{
        private static int execOperation(int operande1,String operation,int operande2)
                        throws divisionParZeroException, operationInconnueException
        {
                if ("*".equals(operation))
                        return operande1*operande2;
                if ("/".equals(operation))
                {
                        if (operande2 == 0)
                                throw new divisionParZeroException();
                        return operande1/operande2 ;
                }
                else throw new operationInconnueException(operation);
   }

public static void main (String args[])
{
        System.out.println("exemple de traitement d'exceptions");
```

```
        int    operande1[] = {3,  56, 33, 25};
        String operation[] = {"*","/","+","/"};
        int    operande2[] = {5,  4,  5,  0};

        for (int n = 0;n < operande1.length;n++)
        {
                try
                {
                        System.out.println("Calcul de "+ operande1[n] + operation[n] +
                                                                operande2[n]);
                        System.out.println("Resultat"  + execOperation
                                        ( operande1[n], operation[n], operande2[n]));
                }
                catch (divisionParZeroException e)
                { System.out.println("Division par zero."); }
                catch (operationInconnueException e)
                {System.out.println("Operation inconnue " + e.getMessage() ); }
                catch (Exception e)
                { System.out.println("une autre exception est survenue.");}
                 finally
                {  System.out.println("clause finally appellee"); }
        }
    }
}
```

Exécution

```
exemple de traitement d'exceptions
Calcul de 3*5
Resultat = 15
clause finally appellee
Calcul de 56/4
Resultat = 14
clause finally appellee
Calcul de 33+5
Operation inconnue : +
clause finally appellee
Calcul de 25/0
Division par zero.
clause finally appellee
```

Exemple de l'ouvrage Java in Nutshell de chez O'Reilly

Le meilleur que je connaisse concernant la propagation des exceptions sur la Pile des appels.

```
// Here we define some exception types of our own.
// Exception classes generally have constructors but no data or
// other methods.  All these do is to call their superclass constructors.

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

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

class MySubException extends MyException
{
   public MySubException() { super(); }
   public MySubException(String s) { super(s); }
}

// This class demonstrates defining, throwing and handling exceptions.
// Try invoking it in the following ways and try to understand the
// output:
//    java throwtest
//    java throwtest one
//    java throwtest 0
//    java throwtest 1
//    java throwtest 99
```

```
//    java throwtest 2
//    java throwtest 3

public class throwtest
{
   // This is the main() method.  Note that it uses two
   // catch clauses to handle two standard Java exceptions.
   public static void main(String argv[])
   {
      int i;
      // First, convert our argument to an integer
      // Make sure we have an argument and that it is convertible.
      try
      {
         i = Integer.parseInt(argv[0]);
      }
      catch (ArrayIndexOutOfBoundsException e)
      { // argv is empty
         System.out.println("Must specify an argument");
         return;
      }
      catch (NumberFormatException e)
      { // argv[0] isn't an integer
         System.out.println("Must specify an integer argument.");
         return;
      }

      // Now, pass that integer to method a().
      a(i);
   }
```

```
// This method invokes b(), which is declared to throw
// one type of exception.  We handle that one exception.

public static void a(int i)
{
      try
      {
            b(i);
      }

      // A PROGRAMMER AVEC DEUX CLAUSES catch
      catch (MyException e)
      {                       // Point 1.
      // Here we handle MyException and
      // its subclass MySubException
      if (e instanceof MySubException)
         System.out.print("MySubException: ");
      else
         System.out.print("MyException: ");

      System.out.println(e.getMessage());
      System.out.println("Handled at point 1");
   }
}
```

```
    // This method invokes c(), and handles one of the
    // two exception types that that method can throw.  The other
    // exception type is not handled, and is propagated up
    // and declared in this method's throws clause.
    // This method also has a finally clause to finish up
    // the work of its try clause.  Note that the finally clause
    // is executed after a local catch clause, but before a
    // a containing catch clause or one in an invoking procedure.

    public static void b(int i) throws MyException
            // et à fortiori MySubException
{
        int result;
        try
        {
            System.out.print("i = " + i);
            result = c(i);
            System.out.print(" c(i) = " + result);
        }
        catch (MyOtherException e)
        {                       // Point 2
            // Handle MyOtherException exceptions:
            System.out.println("MyOtherException: " + e.getMessage());
            System.out.println("Handled at point 2");
        }
        finally
        {
            // Terminate the output we printed above with a newline.
            System.out.print("\n");
        }
    }
```

```
   // This method computes a value or throws an exception.
   // The throws clause only lists two exceptions, because
   // one of the exceptions thrown is a subclass of another.

  public static int c(int i) throws MyException, MyOtherException
          // et à fortiori MySubException
{
      switch (i) {
         case 0: // processing resumes at point 1 above
            throw new MyException("input too low");
         case 1: // processing resumes at point 1 above
            throw new MySubException("input still too low");
         case 99:// processing resumes at point 2 above
            throw new MyOtherException("input too high");
         default:
            return i*i;
      }
   }
}
```