Chapitre 8

Exceptions

8.1 Introduction

Les exceptions sont un mécanisme qui permet de gérer certaines erreurs, notamment celles qui proviennent des méthodes prédéfinies et qui conduisent à un arrêt brutal du programme. Par exemple, si l'on cherche à accéder à une case de tableau qui n'existe pas, l'exception ArrayIndexOutOfBoundsException est déclenchée et le programme s'arrête. Autre cas : le programme lit un entier avec Terminal.lireInt() et l'utilisateur tape des lettres ou des signes de ponctuation : c'est cette fois l'erreur TerminalException qui est activée.

De façon plus générale, les exceptions sont utilisées quand une erreur est détectée par une méthode (prédéfinie ou pas) et que le traitement de cette erreur n'est pas réalisée à l'endroit où elle est detectée mais à celui où la méthode a été appelée.

Le mécanisme d'exception introduit une perturbation dans l'ordre d'exécution des instructions d'un programme.

Il y a trois temps dans la gestion des exceptions :

— la création de l'exception
— le déclenchement de l'exception à l'endroit où une erreur est détectée
— la désactivation de l'exception à l'endroit où le problème est résolu

La terminologie employée dans le langage Java est la suivante :

— pour le déclenchement de l'exception : on dit que l'exception est lancée (*to throw* en anglais). Dans d'autres langages, c'est le terme *lever* (*to raise*) qui est employé.
— pour la désactivation de l'exception : on dit que l'exception est attrapée (*to catch* en anglais)

Le programmeur ne programme pas toujours les trois temps. Dans le cas des méthodes prédéfinies, la création de l'exception et son déclenchement sont prédéfinis dans la librairie Java et le programmeur n'a plus qu'à résoudre le problème. Nous allons étudier d'abord cette situation avant de voir comment le programmeur peut créer et déclencher ses propres exceptions.

8.2 Traiter les exceptions prédéfinies

Prenons l'exemple d'un tableau créé et rempli d'après des données saisies au clavier. Voyons d'abord le programme nu, sans traitement des erreurs.

```
1 public class Excep_1{
2    public static void main(String[] args){
```

```
3       int[] tab;
4       int taille, indice;
5       Terminal.ecrireString("Taille␣du␣tableau?␣");
6       taille = Terminal.lireInt();
7       tab = new int[taille];
8       for (int i=0; i<taille; i=i+1){
9          Terminal.ecrireString("Entrez␣un␣nombre:␣");
10         tab[i]=Terminal.lireInt();
11      }
12      Terminal.ecrireString("Quelle␣case␣voulez-vous␣doubler?␣");
13      indice = Terminal.lireInt();
14      tab[indice]=tab[indice]*2;
15      for (int i=0; i<taille; i=i+1){
16         Terminal.ecrireString(tab[i]+"␣");
17      }
18      Terminal.sautDeLigne();
19   }
20 }
```

Quand tout va bien, le programme s'exécute jusqu'au bout.

```
> java Excep_1
Taille du tableau? 3
Entrez un nombre: 10
Entrez un nombre: 5
Entrez un nombre: 7
Quelle case voulez-vous doubler? 1
10 10 7
```

Mais un certain nombre d'erreurs peuvent se produire :

— Terminal.lireInt (lignes 6, 10 et 13) : TerminalException si autre chose qu'un entier est entré.
— new int[taille] (ligne 7) : NegativeArraySizeException si taille est négatif
— tab[indice] (ligne 14) : ArrayIndexOutOfBoundsException si indice n'est pas un numéro de case correct.

Exemples de ces erreurs :

```
> java Excep_1
Taille du tableau? azerty
Exception in thread "main" TerminalException
        at Terminal.exceptionHandler(Terminal.java:115)
        at Terminal.lireInt(Terminal.java:25)
        at Excep_1.main(Excep_1.java:6)
> java Excep_1
Taille du tableau? -10
Exception in thread "main" java.lang.NegativeArraySizeException
        at Excep_1.main(Excep_1.java:7)
> java Excep_1
Taille du tableau? 3
```

```
Entrez un nombre: 10
Entrez un nombre: 5
Entrez un nombre: 7
Quelle case voulez-vous doubler? 10
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at Excep_1.main(Excep_1.java:14)
```

8.2.1 Résoudre les problèmes : l'instruction try...catch

L'instruction qui sert à attraper les exceptions a la structure suivante :

```
try{
   instruction-1
   instruction-2
   ...
   instruction-n
}catch(NomDException e){
   autre-instruction-1
   autre-instruction-2
   ...
   autre-instruction-n
}
```

Lors du déroulement normal du programme, les instructions `instruction-1` à `instruction-n` sont exécutées et les instructions situées après le catch (`autre-instruction-1` ...) ne sont pas exécutées.

Si une des instructions `instruction-1` à `instruction-n` lance l'exception NomDException, alors l'exécution de ces instructions-là s'arrête et ce sont les instructions `autre-instruction-1` à `autre-instruction-n` qui sont exécutées.

Supposons que c'est l'instruction `instruction-2` qui lance `NomDException`. Dans ce cas, `instruction-1` est exécutée, puis `instruction-2` est en partie exécutée et ensuite `autre-instruction-1` à `autre-instruction-n` sont exécutées. Dans ce cas, les instructions `instruction-3` à `instruction-n` ne sont pas exécutées.

Le lancement de l'exception fait que le déroulement normal du programme est interrompu et c'est la suite des autres instructions (`autre-instruction-1` à `autre-instruction-n`) qui est exécutée.

Si une des instructions `instruction-1` à `instruction-n` lance une exception qui n'est pas `NomDException`, le programme s'arrête brutalement.

Le même try peut avoir plusieurs catch pour traiter des instructions différentes.

```
try{
   liste-instuctions-1
}catch(Exception1 e){
   liste-instructions-2
}catch(Exception2 e){
   liste-instructions-3
}
```

8.2.2 Traitement d'une exception de l'exemple

On peut commencer par le traitement de l'exception TerminalException pour notre programme. Voici un premuier exemple d'utilisation du try...catch.

```
1  public class Excep_1_1{
2     public static void main(String[] args){
3        int[] tab;
4        int taille, indice;
5        try{
6           Terminal.ecrireString("Taille␣du␣tableau?␣");
7           taille = Terminal.lireInt();
8           tab = new int[taille];
9           for (int i=0; i<taille; i=i+1){
10             Terminal.ecrireString("Entrez␣un␣nombre:␣");
11             tab[i]=Terminal.lireInt();
12          }
13          Terminal.ecrireString("Quelle␣case␣voulez-vous␣doubler?␣");
14          indice = Terminal.lireInt();
15          tab[indice]=tab[indice]*2;
16          for (int i=0; i<taille; i=i+1){
17             Terminal.ecrireString(tab[i]+"␣");
18          }
19          Terminal.sautDeLigne();
20       }catch(TerminalException e){
21          Terminal.ecrireString("On␣vous␣demandait␣un␣nombre␣entier,␣");
22          Terminal.ecrireStringln("Vous␣avez␣tapé␣autre␣chose.");
23          Terminal.ecrireStringln("␣Tant␣pis␣pour␣vous.");
24       }
25    }
26 }
```

Les deux exemples suivants montrent que plus ou moins d'instructions sont exécutées selon le moment où la saisie est incorrecte.

```
> java Excep_1_1
Taille du tableau? aze
On vous demandait un nombre entier, Vous avez tapé autre chose.
Tant pis pour vous.
> java Excep_1_1
Taille du tableau? 3
Entrez un nombre: 1
Entrez un nombre: 2
Entrez un nombre: a
On vous demandait un nombre entier, Vous avez tapé autre chose.
Tant pis pour vous.
```

Mais dans les deux cas d'exception, les dernières instructions de la voie normale (lignes 13 à 19) ne sont pas exécutées.

Grâce au traitement d'exception de ce programme, celui-ci ne plante plus brutalement comme avant, mais le résultat est presque le même : le programme n'a rien fait d'utile. Il vaudrait mieux un programme qui donne une seconde chance après une erreur de saisie.

8.2.3 Meilleur traitement d'un erreur de lecture

```
public class Excep_1_2{
   public static void main(String[] args){
      int[] tab;
      int taille, indice;
      Terminal.ecrireString("Taille du tableau? ");
      try{
         taille = Terminal.lireInt();
      }catch(TerminalException e){
         Terminal.ecrireString("On vous demandait un nombre entier, ");
         Terminal.ecrireString("Veuillez rentrer un nombre entier: ");
         taille = Terminal.lireInt();
      }
      tab = new int[taille];
      for (int i=0; i<taille; i=i+1){
         Terminal.ecrireString("Entrez un nombre: ");
         try{
            tab[i]=Terminal.lireInt();
         }catch(TerminalException e){
            Terminal.ecrireString("On vous demandait un nombre entier, ");
            Terminal.ecrireString("Veuillez rentrer un nombre entier: ");
            tab[i] = Terminal.lireInt();
         }
      }
      Terminal.ecrireString("Quelle case voulez-vous doubler? ");
      try{
         indice = Terminal.lireInt();
      }catch(TerminalException e){
         Terminal.ecrireString("On vous demandait un nombre entier, ");
         Terminal.ecrireString("Veuillez rentrer un nombre entier: ");
         indice = Terminal.lireInt();
      }
      tab[indice]=tab[indice]*2;
      for (int i=0; i<taille; i=i+1){
         Terminal.ecrireString(tab[i]+" ");
      }
      Terminal.sautDeLigne();
   }
}
```

On a été obligé de faire plusieurs try...catch différents pour éviter d'avoir des instructions non exécutées dans le cas où la saisie est erronée.

Un exemple d'exécution :

```
> java Excep_1_2
Taille du tableau? a
On vous demandait un nombre entier, Veuillez rentrer un nombre entier: 3
Entrez un nombre: 5
Entrez un nombre: 1
Entrez un nombre: a
On vous demandait un nombre entier, Veuillez rentrer un nombre entier: 4
Quelle case voulez-vous doubler? a
On vous demandait un nombre entier, Veuillez rentrer un nombre entier: 1
5 2 4
```

Le comportement du programme est meilleur, mais il n'est pas parfait, car s'il y a une erreur sur la deuxième chance de saisie, le programme plante.

```
> java Excep_1_2
Taille du tableau? a
On vous demandait un nombre entier, Veuillez rentrer un nombre entier: a
Exception in thread "main" TerminalException
        at Terminal.exceptionHandler(Terminal.java:115)
        at Terminal.lireInt(Terminal.java:25)
        at Excep_1_2.main(Excep_1_2.java:11)
```

En effet, l'instruction `taille = Terminal.lireInt();` dans la liste de sinstructions du catch n'est pas soumise à une récupération d'exception. Seules les instruction dans la partie `try` sont soumises au `catch`.

8.2.4 Traitement amélioré des lectures

Voici comment on peut améliorer encore le traitement des saisies : il faut proposer en boucle la lecture au clavier du nombre autant de fois que nécessaire et faire en sorte que cette saisie soit toujours à l'intérieur du try.

```
public class LectureProtegee{
   public static void main(String[] args){
      int entier=0;
      boolean correct;
      do{
         try{
            Terminal.ecrireString("Entrez un entier: ");
            entier = Terminal.lireInt();
            correct=true;
         }catch(TerminalException e){
            Terminal.ecrireString("Vous avez tapé autre chose qu'un entier");
            Terminal.ecrireStringln(" Veuillez recommencer");
            correct = false;
         }
      }while(!correct);
```

```
      Terminal.ecrireIntln(entier);
   }
}
```

Exécution :

```
> java LectureProtegee
Entrez un entier: a
Vous avez tapé autre chose qu'un entier Veuillez recommencer
Entrez un entier: 1.2
Vous avez tapé autre chose qu'un entier Veuillez recommencer
Entrez un entier: 12
12
```

Dans notre exemple, nous avons trois instructions Terminal.lireInt. Plutôt que de répéter la boucle trois fois, nous allons créer une méthode pour réaliser la lecture des entiers.

```
public class Excep_1_3{
   public static int lireEntier(String msg){
      int entier=0;
      boolean correct;
      do{
         try{
            Terminal.ecrireString(msg);
            entier = Terminal.lireInt();
            correct=true;
         }catch(TerminalException e){
            Terminal.ecrireString("Vous␣avez␣tapé␣autre␣chose␣qu'un␣entier");
            Terminal.ecrireStringln("␣Veuillez␣recommencer");
            correct = false;
         }
      }while(!correct);
      return entier;
   }
   public static void main(String[] args){
      int[] tab;
      int taille, indice;
      taille = lireEntier("Taille␣du␣tableau?␣");
      tab = new int[taille];
      for (int i=0; i<taille; i=i+1){
         tab[i]=lireEntier("Entrez␣un␣nombre:␣");
      }
      indice=lireEntier("Quelle␣case␣voulez-vous␣doubler?␣");
      tab[indice]=tab[indice]*2;
      for (int i=0; i<taille; i=i+1){
         Terminal.ecrireString(tab[i]+"␣");
      }
      Terminal.sautDeLigne();
```

```
   }
}
```

Exécution :

```
> java Excep_1_3
Taille du tableau? a
Vous avez tapé autre chose qu'un entier Veuillez recommencer
Taille du tableau? 2
Entrez un nombre: 1
Entrez un nombre: 1
Quelle case voulez-vous doubler? a
Vous avez tapé autre chose qu'un entier Veuillez recommencer
Quelle case voulez-vous doubler? 1
1 2
```

8.2.5 Traitement de toutes les exceptions

En appliquant le même principe aux autres exceptions (NegativeArraySizeException et ArrayIndexOutOfBoundsException), on obtient le programme suivant.

```
public class Excep_1_4{
   public static int lireEntier(String msg){
      int entier=0;
      boolean correct;
      do{
         try{
            Terminal.ecrireString(msg);
            entier = Terminal.lireInt();
            correct=true;
         }catch(TerminalException e){
            Terminal.ecrireString("Vous␣avez␣tapé␣autre␣chose␣qu'un␣entier");
            Terminal.ecrireStringln("␣Veuillez␣recommencer");
            correct = false;
         }
      }while(!correct);
      return entier;
   }
   public static void main(String[] args){
      int[] tab ={};
      int taille=0, indice;
      boolean correct;
      do{
         try{
            taille = lireEntier("Taille␣du␣tableau?␣");
            tab = new int[taille];
            correct=true;
         }catch(NegativeArraySizeException e){
```

```
            Terminal.ecrireString("Une taille doit être positive.");
            Terminal.ecrireStringln(" Veuillez recommencer");
            correct = false;
         }
      }while(!correct);
      for (int i=0; i<taille; i=i+1){
         tab[i]=lireEntier("Entrez un nombre: ");
      }
      do{
         try{
            indice=lireEntier("Quelle case voulez-vous doubler? ");
            tab[indice]=tab[indice]*2;
            correct=true;
         }catch(ArrayIndexOutOfBoundsException e){
            Terminal.ecrireString("Cette case n'existe pas.");
            Terminal.ecrireStringln(" Veuillez recommencer");
            correct = false;
         }
      }while(!correct);
      for (int i=0; i<taille; i=i+1){
         Terminal.ecrireString(tab[i]+" ");
      }
      Terminal.sautDeLigne();
   }
}
```

Ce programme est devenu assez compliqué, mais grâce à cela, il n'est plus à la merci d'une frappe incohérente de l'utilisateur. Même si un petit chat rentre n'importe quoi en machant sur le clavier, le programme ne plantera pas !

8.3 Définir une exception

Le programmeur peut définir des exceptions correspondant à des erreurs qui peuvent se produire dans son programme s'il souhaite que la détection et le traitement de cette erreur soient dans des méthodes différentes du programme.

Prenons l'exemple de la fonction factorielle. Voici un code possible pour cette fonction en java.

```
public class Fact_1{
   public static int factorielle(int n){
      int res = 1;
      for (int i=n; i!=1; i=i-1){
         res = res *i;
      }
      return res;
   }
   public static void main(String[] args){
      int x;
```

```
      Terminal.ecrireString("Entrez un nombre: ");
      x = Terminal.lireInt();
      Terminal.ecrireStringln("factorielle de " + x + " = " +
                        factorielle(x));
   }
}
```

Le problème de ce codage, c'est que si l'on donne un paramètre négatif à factorielle, le résultat obtenu est erroné. La fonction factorielle n'est définie que pour les nombres positifs ou nuls.

On peut créer une exception spécifique pour gérer cette erreur et empêcher le programme de donner un résultat faux. Appelons-la FactorielleNonDefinie. Pour créer cette exception il faut écrire le code suivant :

```
class FactorielleNonDefinie extends RuntimeException {}
```

Ce code peut être mis au début du fichier contenant le programme, avant la déclaration de la classe qui contient la méthode main.

8.4 Lancer une exception

Pour lancer l'exception, il faut utiliser le code suivant :

```
throw new FactorielleNonDefinie();
```

Il y a là deux morceaux : `throw` est une instruction qui lance l'exception et `new` est qui crée l'exception lancée. Ce `new` est le même que celui utilisé pour créer un tableau. Il y a une création de quelque chose de nouveau stocké dans le tas.

8.4.1 L'exemple de la factorielle continué

Voici comment utiliser concrètement l'exception dans le cas de notre exemple.

```
class FactorielleNonDefinie extends RuntimeException {}
public class Fact_2{

   public static int factorielle(int n){
      if (n<0){
         throw new FactorielleNonDefinie();
      }
      int res = 1;
      for (int i=n; i!=1; i=i-1){
         res = res *i;
      }
      return res;
   }
   public static void main(String[] args){
      int x;
      Terminal.ecrireString("Entrez un nombre: ");
      x = Terminal.lireInt();
```

```
      Terminal.ecrireStringln("factorielle␣de␣" + x + "␣=␣" +
                        factorielle(x));
   }
}
```

Exemple d'exécutions :

```
> java Fact_2
Entrez un nombre: 5
factorielle de 5 = 120
> java Fact_2
Entrez un nombre: -2
Exception in thread "main" FactorielleNonDefinie
        at Fact_2.factorielle(Fact_2.java:6)
        at Fact_2.main(Fact_2.java:19)
```

8.4.2 Attraper une exception que l'on a lancée

On peut attrapper les exceptions que l'on lance avec un try...catch de la même façon que pour les exceptions lancées par les méthodes prédéfinies.

```
class FactorielleNonDefinie extends RuntimeException {}
public class Fact_3{

   public static int factorielle(int n){
      if (n<0){
         throw new FactorielleNonDefinie();
      }
      int res = 1;
      for (int i=n; i!=1; i=i-1){
         res = res *i;
      }
      return res;
   }
   public static void main(String[] args){
      int x;
      boolean correct;
      do{
         try{
            Terminal.ecrireString("Entrez␣un␣nombre:␣");
            x = Terminal.lireInt();
            Terminal.ecrireStringln("factorielle␣de␣" + x + "␣=␣" +
                              factorielle(x));
            correct = true;
         }catch(FactorielleNonDefinie e){
            Terminal.ecrireString("La␣factorielle␣n'est␣pas␣définie␣");
            Terminal.ecrireString("pour␣les␣nombres␣négatifs.␣");
            correct = false;
```

```
         }
      }while(!correct);
   }
}
```

Exemple d'exécution :

```
> java Fact_3
Entrez un nombre: -2
La factorielle n'est pas définie pour les nombres négatifs. Entrez un nombre: 2
factorielle de 2 = 2
```

8.4.3 Jamais le throw directement dans le try

L'utilisation d'une exception ne se justifie que si on lance te on attrape l'exception dans deux méthodes différentes.

En effet, si l'on fait les deux dans la même méthode, on peut utiliser plus simplement un if et c'est préférable.

Voyons un exemple : la saisie d'un nombre compris dans un intervalle.

```
class HorsIntervalle extends RuntimeException{}
public class Excep_2{
   public static int lireEntierIntervalle(int min, int max){
      int res=0;
      boolean correct;
      do{
         try{
            Terminal.ecrireString("Entrez un nombre: ");
            res = Terminal.lireInt();
            if (res<min  res>max){
               throw new HorsIntervalle();
            }
            correct = true;
         }catch(HorsIntervalle e){
            Terminal.ecrireStringln("Nombre hors intervalle");
            correct = false;
         }
      }while(!correct);
      return res;
   }
}
```

Cette méthode peut se réécrire plus simplement avec le même comportement de la façon suivante.

```
public class Excep_2{
   public static int lireEntierIntervalle(int min, int max){
      int res=0;
      boolean correct;
      do{
```

```
         Terminal.ecrireString("Entrez un nombre: ");
         res = Terminal.lireInt();
         if (res<min  res>max){
            Terminal.ecrireStringln("Nombre hors intervalle");
            correct = false;
         }else{
            correct = true;
         }
      }while(!correct);
      return res;
   }
}
```

Retenez bien qu'un throw ne doit jamais être écrit directement dans le try...catch qui attrape l'exception : il doit être écrit dans une méthode appelée directement ou indirectement dans le try...catch.

8.5 Appel indirect de la méthode qui lance l'exception

Il peut y avoir une chaîne de plusieurs méthodes entre la méthode appelée dans la try...catch et la méthode qui lance l'exception avec `throw`. Il peut y avoir dans le try...catch une méthode 1 et dans le code de cette méthode une méthode 2 et dans le code de cette méthode 2 un appel à méthode 3 et dans celle-là un `throw`. Il peut ainsi y avoir une chaîne d'appels.

Le principe est que si une méthode appelle une autre méthode qui lance une exception et qu'elle n'attrape pas cette exception, c'est comme si elle la lançait elle-même et elle peut être mise dans un try.

Voyons un exemple simplifié qui retranscrit ce qui se passe lors de la levée d'une exception.

```
class LException extends RuntimeException{}
public class Excep_3{
   public static void methode1(int x){
      System.out.println("methode 1 commence");
      methode2(x);
      System.out.println("methode 1 se termine");
   }
   public static void methode2(int x){
      System.out.println("methode 2 commence");
      methode3(x);
      System.out.println("methode 2 se termine");
   }
   public static void methode3(int x){
      System.out.println("methode 3 commence");
      if (x==0){
         throw new LException();
      }
      System.out.println("methode 3 se termine");
   }
   public static void main(String[] args){
```

```
      int x;
      System.out.println("Debut du main");
      System.out.print("Entier: ");
      x = Terminal.lireInt();
      try{
         methode1(x);
         System.out.println("Fin du try");
      }catch(LException e){
         System.out.println("Exception capturee");
      }
      System.out.println("fin du main");
   }
}
```

Exécution :

```
> java Excep_3
Debut du main
Entier: 2
methode 1 commence
methode 2 commence
methode 3 commence
methode 3 se termine
methode 2 se termine
methode 1 se termine
Fin du try
fin du main
> java Excep_3
Debut du main
Entier: 0
methode 1 commence
methode 2 commence
methode 3 commence
Exception capturee
fin du main
```

Dans ce programme, on a choisit de mettre le try dans la méthode main. On aurait pu le mettre aussi dans methode1 ou methode2.