BLOC 2 – Ch. 11 – Les exceptions

La gestion des erreurs est un élément indispensable en programmation. Nous abordons ici les erreurs qui surviennent à l'exécution d'un programme. La gestion des erreurs d'exécution est facilitée grâce à un mécanisme élaboré. On appelle **exception** une erreur générée lors de l'exécution d'un programme ou d'une fonction.

Cette exception peut être le résultat d'une erreur ou bien être provoquée par le développeur afin d'alerter le programme appelant d'une situation anormale. Les langages objet disposent de classes d'exception correspondant aux causes de l'exception.

1 Exemples d'exceptions

1.1 Activité

Considérons les deux programmes suivants :

```
int num = 10;
int den = 0;
int fraction = num / den;
```

```
int[] t = { 1, 2, 3, 4, 5 };
int n = t[6];
```

Travail à faire : indiquer quelles sont les erreurs déclenchées à l'exécution des programmes et entourer les instructions qui les provoquent.

1.2 Exemples d'exceptions

Considérons de nouveau le code suivant :

```
8   public class Program
9   {
10      public static void main(String[] args)
11      {
12          int num = 10;
13          int den = 0;
14          int fraction = num / den;
15      }
```

Il déclenche une erreur qui interrompt le programme à la ligne 14 et affiche le message :

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at cours.Program.main(Program.java:14)
```

De même, le code suivant :

```
8   public class Program
9   {
10      public static void main(String[] args)
11      {
12          int[] t = { 1, 2, 3, 4, 5 };
13          int n = t[6];
14      }
```

Déclenche l'erreur :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
      at cours.Program.main(Program.java:13)
```

2 Définition d'une exception

Une **exception** est une erreur survenue au cours de l'**exécution** d'un programme. L'origine d'une exception peut-être très diverse :

- mauvaise action de l'utilisateur (entrée d'une lettre dans un champ destiné à recevoir des nombres) ;
- problème de connexion entre le programme et son environnement ;
- exception créée par le programmeur, etc.

Une exception est un objet.

3 Gestion des exceptions grâce à try/catch

Pour intercepter l'erreur, afin de ne pas interrompre brutalement le programme, il faut mettre le programme dans un contexte de gestion d'erreur. Ceci se fait grâce à un bloc **try**.

3.1 Bloc try

```
public static void main(String[] args)
{
    try
    {
            int[] t = { 1, 2, 3, 4, 5 };
            int n = t[6];
            System.out.println("Cette ligne ne sera jamais atteinte");
    }
    // Ce code doit être suivi d'instructions qui indiquent au programme
    // ce qu'il doit faire en cas d'erreur :
    catch (ArrayIndexOutOfBoundsException exc)
    {
            System.out.println("L'exception a été attrapée");
    }
    System.out.println("Suite du programme...");
}
```

Travail à faire - Qu'est-ce qui s'affiche ?

. . . . . .
. . . . . .

3.2 Traitement de l'exception

Exemple :

```
    public static void main(String[] args)
    {
      int fraction;
      try
      {
          int num = 10;
          int den = 0;
          fraction = num / den;

          System.out.println("Cette ligne ne sera jamais atteinte");
      }
```

// on surveille l'exception ArithmeticException

```
        catch (ArithmeticException exc)
        {
            System.out.println("Une erreur est survenue :");

            // on affiche le message encapsulé par l'exception
            System.out.println( exc.getMessage() );

            // résolution du problème
            fraction = 0;
        }
        // le programme peut continuer…
    }
```

Affichage :

```
Une erreur est survenue :
/ by zero
```

Autre possibilité :

On peut écrire dans le **catch** :

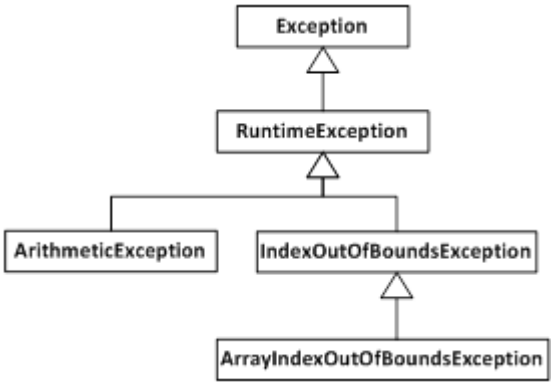
```
        catch (ArithmeticException exc)
        {
            ...
            // au lieu de :        System.out.println( exc.getMessage() );
            exc.printStackTrace();

            ...
        }
```

Cela permet d'afficher où s'est produite l'exception :

```
Une erreur est survenue :
java.lang.ArithmeticException: / by zero
      at cours.Program.main(Program.java:21)
```

4 Surveiller et traiter plusieurs exceptions

4.1 Surveiller et traiter plusieurs exceptions



Commentaires :

. . .

. . .

```
public static void main(String[] args)
{
    // on suppose que num, den, t ont été définis et initialisés avant
    try
    {
       fraction = num / den;
       int n = t[6];
       // autre lignes de code
       // ...
    }

    catch (ArrayIndexOutOfBoundsException exc)
    {
       // on traite les exceptions de type ArrayIndexOutOfBoundsException
    }

    catch (ArithmeticException exc)
    {
       // on traite les exceptions de type ArithmeticException
    }

    catch (Exception exc)
    {
       // on traite n'importe quelle exception de manière générale
    }

    System.out.println("Suite du programme...");
}
```

A un bloc **try** peuvent correspondre plusieurs blocs **catch**

4.2 Exceptions qui doivent être surveillées

Certaines méthodes susceptibles de déclencher des exceptions doivent être surveillées explicitement. Par exemple le code suivant ne peut pas être compilé :

```
       // ouverture d'une connexion à une base de données
       Connection conn = DriverManager.getConnection(URL, LOGIN, PASSWORD);
       conn.close();
```

Le compilateur indique l'erreur : "unreported exception SQLException; must be caught or declared to be thrown".
Il faut mettre ce code dans un bloc try :

```
try
{
       Connection conn = DriverManager.getConnection(URL, LOGIN, PASSWORD);
       conn.close();
}
catch (SQLException exc)
{      exc.printStackTrace();       }
```

Seules les exceptions de type **RuntimeException** n'ont pas à être surveillées explicitement.

5 Déclenchement d'une exception

Le développeur peut décider de déclencher une exception en utilisant l'instruction **throw**.
Prenons cet exemple très simple :

```
14      public static void main(String[] args)
15      {
16          System.out.println("début du programme");
17          // instanciation d'une exception
18          RuntimeException exc = new RuntimeException("Erreur déclenchée par moi-même !");
19          // lancement de l'exception
20          throw exc;
21
22          // Cette ligne ne sera jamais atteinte
23      }
```

Cela produit l'affichage suivant :

```
début du programme
Exception in thread "main" java.lang.RuntimeException: Erreur déclenchée par moi-même !
        at cours.LanceExc.main(LanceExc.java:18)
```

Ce mécanisme peut servir pour créer des méthodes qui vont créer des exceptions en cas de problème. Exemple :

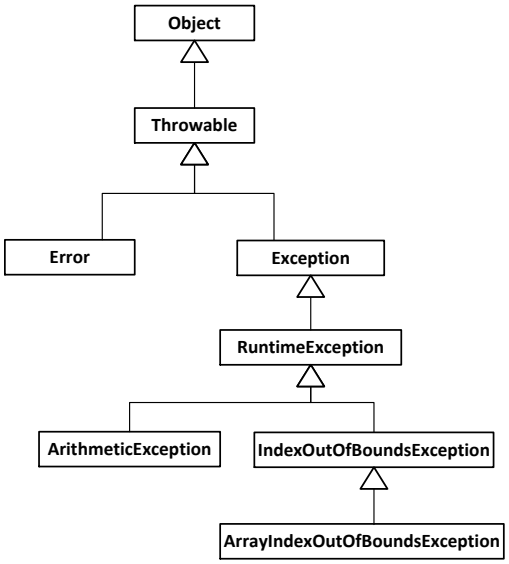
```
// Programme principal
public static void main(String[] args)
{
    try
    {
      System.out.println("appel de valideNote");

      valideNote(25);

      System.out.println("fin du programme");
    }
    catch (Exception exc)
    {
      System.out.println(exc.getMessage());
    }
}

    /**
     * Valide une note en vérifiant qu'elle est comprise entre 0 et 20 (inclus).
     * @param note la note à valider
     * @throws Exception l'exception levée si la note est invalide
     */
    public static void valideNote(int note) throws Exception
    {
        if (note < 0)
                throw new Exception("Note invalide (négative) : " + note);
        if (note > 20)
                throw new Exception("Note invalide (trop grande) : " + note);
    }
```

Cela affiche :

```
      appel de valideNote
      Note invalide (trop grande) : 25
```

Synthèse
- Le mécanisme des exceptions permet la gestion des erreurs ;
- Une exception est un objet, elle est une instance d'une classe ;
- On peut traiter des exceptions avec les blocs **try**{ } et **catch**{ } ;
- Une méthode peut propager une exception avec **throws** ;
- Il est possible de créer et de lancer une exception : **throw new Exception().**

6 Hiérarchie des classes d'exceptions



Commentaires :

. . .

. . .

```
// on suppose que num, den, t ont été définis et initialisés avant
try
{
   fraction = num / den;
   int n = t[6];
   // autre lignes de code
   // ...
}
catch (ArrayIndexOutOfBoundsException exc)
{
   // on traite les exceptions de type ArrayIndexOutOfBoundsException
}
catch (ArithmeticException exc)
{
   // on traite les exceptions de type ArithmeticException
}
catch (Exception exc)
{
   // on traite n'importe quelle exception de manière générale
}
System.out.println("Suite du programme...");
```

A un bloc **try** peuvent correspondre plusieurs blocs **catch**

Lorsque l'on surveille dans des blocs **catch** des exceptions qui appartiennent à la même hiérarchie, il faut surveiller d'abord l'exception la plus basse dans la hiérarchie, puis ensuite les plus hautes.

7 Classes d'exceptions personnalisées

Il est possible de créer ses propres classes d'Exception qui héritent d'une classe d'Exception existante. Par exemple, la classe **MonException** hérite de **Exception,** elle a un constructeur qui fait appel au constructeur de la classe mère :

```
public class MonException extends Exception     {
    public MonException(String message)     {
       super(message);
    }
}

public class Program {
    public static void main(String[] args)     {
       try
       {
           MonException exc = new MonException("Ma propre exception !");
           throw exc;
       }
       catch (MonException exc)
       {
           System.out.println(exc.getMessage());       // affiche : « Ma propre exception ! »
       }
    }
}
```