GESTION DES EXCEPTIONS

Lors de l'écriture d'un programme, la prise en compte d'erreurs prend une place très importante si l'on souhaite écrire un programme robuste. Par exemple, la simple ouverture d'un fichier peut provoquer beaucoup d'erreurs telles que l'inexistence du fichier, un mauvais format, une interdiction d'accès, une erreur de connexion au périphérique, … Pour que notre programme soit robuste, il faut que toutes les erreurs possibles soient détectées et traitées.

Certains langages de programmation, dont le langage Java, proposent un mécanisme de prise en compte des erreurs, fondé sur la notion d'*exception*. Une exception est un objet qui peut être émis par une méthode si un événement d'ordre "exceptionnel" (les erreurs rentrent dans cette catégorie) se produit. La méthode en question ne renvoie alors pas de valeur de retour, mais émet une exception expliquant la cause de cette émission. La propagation d'une émission se déroule selon les étapes suivantes :

1. Une exception est générée à l'intérieur d'une méthode ;
2. Si la méthode prévoit un traitement de cette exception, on va au point 4, sinon au point 3 ;
3. L'exception est renvoyée à la méthode ayant appelé la méthode courante, on retourne au point 2 ;
4. L'exception est traitée et le programme reprend son cours après le traitement de l'exception.

La gestion d'erreurs par propagation d'exception présente deux atouts majeurs :

- **Une facilité de programmation et de lisibilité** : il est possible de regrouper la gestion d'erreurs à un même niveau. Cela évite des

redondances dans l'écriture de traitements d'erreurs et encombre peu le reste du code avec ces traitements.

- **Une gestion des erreurs propre et explicite** : certains langages de programmation utilisent la valeur de retour des méthodes pour signaler une erreur à la méthode appelante. Etant donné que ce n'est pas le rôle de la valeur de retour de décrire une erreur, il est souvent impossible de connaître les causes réelles de l'erreur. La dissociation de la valeur de retour et de l'exception permet à cette dernière de décrire précisément la ligne de code ayant provoqué l'erreur et la nature de cette erreur.

Déclaration

Il est nécessaire de déclarer, pour chaque méthode, les classes d'exception qu'elle est susceptible d'émettre. Cette déclaration se fait à la fin de la signature d'une méthode par le mot-clé ***throws*** à la suite duquel les classes d'exceptions (séparées par une virgule s'il en existe plusieurs) qui peuvent être générées sont précisées. La méthode parseInt de la classe Integer est un bon exemple :

```
public static int parseInt(String s) throws NumberFormatException {
    ...
}
```

Cette méthode convertit une chaîne de caractères, qui doit contenir uniquement des chiffres, en un entier. Une erreur peut se produire si cette chaîne de caractères ne contient pas que des chiffres. Dans ce cas une exception de la classe NumberFormatException est émise. Une exception peut être émise dans une méthode de deux manières : (i) par une autre mé- thode appelée dans le corps de la première méthode ; (ii) par la création d'un objet instanciant la classe Exception

(ou la classe Throwable) et la levée explicite de l'exception en utilisant le mot-clé throw.

L'exemple ci-dessous illustre ce second cas :

```
public class ExempleException {
   /*
    * Cette méthode renvoie le nom du mois
    * correspondant au chiffre donné en paramètre.
    * Si celui-ci n'est pas valide une exception de classe
    * IndexOutOfBoundsException est levée.
    */

   public static String month(int mois)
           throws IndexOutOfBoundsException {
      if ((mois < 1) || (mois > 12)) {
         throw new IndexOutOfBoundsException(
                 "le numero du mois qui est "
                    + mois
                    + " doit être compris entre 1 et 12");
      }
      if (mois == 1)
         return "Janvier";
      else if (mois == 2)
         return "Février";
      ...
      else if (mois == 11)
         return "Novembre";
      else
         return "Décembre";
   }
}
```

La signification des exceptions de la classe IndexOutOfBoundsException est qu'un index donné dépasse les bornes minimum et maximum qu'il devrait respecter. Si une méthode demande la chaîne de caractères correspondant à des mois inexistants (inférieur à 1 ou supérieur à 12) une exception signalera cette erreur. On peut remarquer dans cet exemple que la détection et la formulation de l'erreur sont codées dans la méthode mais pas son traitement

Interception et traitement

Avant de coder le traitement de certaines exceptions, il faut préciser l'endroit où elles vont être interceptées. Si une méthode A appelle une méthode B qui appelle une méthode C qui appelle une méthode D, et que cette méthode D lève une exception, celle-ci est d'abord transmise à C qui peut l'intercepter ou la transmettre à B, qui peut aussi l'intercepter ou la transmettre à A. L'interception d'exceptions se fait par une sorte de "mise sur écoute" d'une portion de code. Pour cela on utilise le mot-clé try suivi du bloc à surveiller. Si aucune exception ne se produit dans le bloc correspondant, le programme se déroule normalement comme si l'instruction try était absente. Par contre, si une exception est levée, le traitement de l'exception est exécuté puis l'exécution du programme reprend son cours après le bloc testé. Il est également nécessaire de préciser quelles classes d'exception doivent être interceptées dans le bloc testé. L'interception d'une classe d'exception s'écrit grâce au mot-clé catch suivi de la classe concerné, d'un nom de variable correspondant à l'objet exception, puis du traitement. Si une exception est levée sans qu'aucune interception ne soit prévue pour sa classe, celle-ci est propagée à la méthode précédente.

Dans l'exemple ci-dessous, le programme demande à l'utilisateur de saisir le numéro d'un mois et affiche à l'écran le nom de ce mois. Les exceptions qui peuvent être levées par ce programme sont traitées.

```java
public class ExempleTraitementException {

   public static void main(String[] args) {
      System.out.print("Entrez le numero d'un mois : ");
      try {
         BufferedReader input = new BufferedReader(
                 new InputStreamReader(System.in));
         String choix = input.readLine();
         int numero = Integer.parseInt(choix);
         System.out.println(ExempleException.month(numero));
      } catch (IndexOutOfBoundsException e) {
         System.err.println("Numero incorrect : "
            + e.getMessage());
      } catch (NumberFormatException e) {
         System.err.println("Entrée incorrecte : "
            + e.getMessage());
      } catch (IOException e) {
         System.err.println("Erreur d'accès : "
            + e.getMessage());
      }
   }
}
```

Trois classes d'exception sont ici traitées :

- IndexOutOfBoundsException (levé par la méthode month) se produit si le numéro entré par l'utilisateur est inférieur à 1 ou supérieur à 12 ;
- NumberFormatException (levé par la méthode parseInt) qui se produit si le texte entré par l'utilisateur n'est pas convertible en entier ;
- IOException (levé par la méthode readLine) qui se produit si il y a eu une erreur d'accès au périphérique d'entrée.

Dans chacun de ces cas, le traitement consiste à afficher le message d'erreur associé à l'exception.

Classes d'exception

Une classe est considérée comme une classe d'exception dès lors qu'elle hérite de la classe Throwable. Un grand nombre de classes d'exception sont proposées dans l'API pour couvrir les catégories d'erreurs les plus fréquentes. Les relations d'héritage entre ces classes permettent de lever ou d'intercepter des exceptions décrivant une erreur à différents niveaux de précision. Les classes d'exception les plus fréquemment utilisées sont récapitulées dans le tableau 5.1

Classe	Description
`AWTException`	Les exceptions de cette classe peuvent se produire lors d'opérations de type graphique.
`ClassCastException`	Signale une erreur lors de la conversion d'un objet en une classe incompatible avec sa vraie classe.
`FileNotFoundException`	Signale une tentative d'ouverture d'un fichier inexistant.
`IndexOutOfBoundsException`	Se produit lorsque l'on essaie d'accéder à un élément inexistant dans un ensemble.
`IOException`	Les exceptions de cette classe peuvent se produire lors d'opérations d'entrées/ sorties.
`NullPointerException`	Se produit lorsqu'un pointeur `null` est reçu par une méthode n'acceptant pas cette valeur, ou lorsque l'on appelle une méthode ou une variable à partir d'un pointeur `null`.

Tableau 4.1 – Classes d'exception fréquentes

Si aucune des classes d'exception ne correspond à un type d'erreur que vous souhaitez exprimer, vous pouvez également écrire vos propres classes

d’exception. Pour cela, il suffit de faire hériter votre classe de la classe java.lang.Exception.

Classification des erreurs en Java

On peut finalement distinguer quatre types de situations d’erreurs en Java :

Erreurs de compilation. Avant même de pouvoir exécuter le programme, notre code source génère des erreurs par le compilateur. Il faut alors réviser et corriger le code pour ne plus avoir d’erreurs.

Erreurs d’exécution. Alors que notre programme est en cours d’exécution, la JVM étant mal configurée ou corrompue, le programme s’arrête ou se gèle. A priori, c’est une erreur non pas due à notre programme, mais à la configuration ou l’état de l’environnement d’exécution de notre programme.

Exception non vérifiée. Alors que notre programme est en cours d’exécution, une trace de la pile des exceptions est affichée, pointant vers une partie de notre code sans gestion d’exception. Visiblement, nous avons utilisé du code qui est capable de lever une exception non vérifiée (comme NullPointerException). Il faut modifier le programme pour que cette situation ne survienne pas.

Exception vérifiée. Alors que notre programme est en cours d’exécution, une trace de la pile des exceptions est affichée, pointant vers une partie de notre code avec gestion d’exception. Visiblement, nous avons produit du code qui est capable de lever une exception vérifiée (comme FileNotFoundException) mais les

données passées à notre programme ne valide pas ces exceptions (par exemple, lorsque l'on essaie d'ouvrir un fichier qui n'existe pas). Il faut alors revoir les données passées en paramètre du programme. Notre code a bien détecté les problèmes qu'il fallait détecter. Le chapitre suivant sur les entrées/sorties présentent de nombreux exemples relatifs à ce cas.