CHAPITRE 5 : EXCEPTIONS

Java, un Langage de programmation orientée objets

SOMMAIRE

Chapitre 5 : Les exceptions

1- Introduction

2- Un premier exemple

3- Gestion des exceptions

- Les blocs ***try*** et ***catch***
- Le mot clé ***throws***
- Le bloc ***finally***

4- Définir sa propre exception

CHAPITRE 5 : EXCEPTIONS

1- Introduction

Exceptions : Introduction

Problème :

- **Un programme ne se déroule pas toujours comme prévu,**
  la cause **:** *des erreurs peuvent perturber son bon déroulement.*

- **Des erreurs telles que :**
  - une division entière par zéro,
  - un dépassement des bornes d'un tableau,
  - l'inexistence d'un fichier,
  - etc.

- Dans tout programme, le traitement des erreurs représente une tâche importante souvent négligée par les programmeurs

Exceptions : Introduction

Solution :

- *Mécanisme d'exception* pour **obliger les programmeurs (Java)** à prendre en compte les erreurs.

- Les exceptions admettent **deux principes fondamentaux** :
  - No -01 : la plus grande partie des erreurs qui pourraient survenir **doit être détectée par le compilateur** de façon à limiter autant que possible les occasions de les voir se produire pendant l'utilisation des programmes.

  - No- 02 : le **traitement des erreurs doit être séparé** du reste du code de façon que celui-ci reste lisible.

Exceptions : définition

- **Exceptions (définition) :**
  Une exception est une interruption de l'exécution d'un programme suite à une erreur.

- **Exemple** : une division par zéro provoque une exception de type ***ArithmeticException***.

Exceptions : diagramme d'héritage

- Les **exceptions** en java **sont des instances de sous-classes** de la super classe ***java.lang.Throwable*** :

  - ***java.lang.Error*** **:** pour des erreurs graves, qui devront généralement conduire à l'arrêt du programme;

  - ou ***java.lang.Exception*** : pour des événements inattendus, qui seront traités de sorte qu'ils ne provoquent pas l'arrêt du programme.

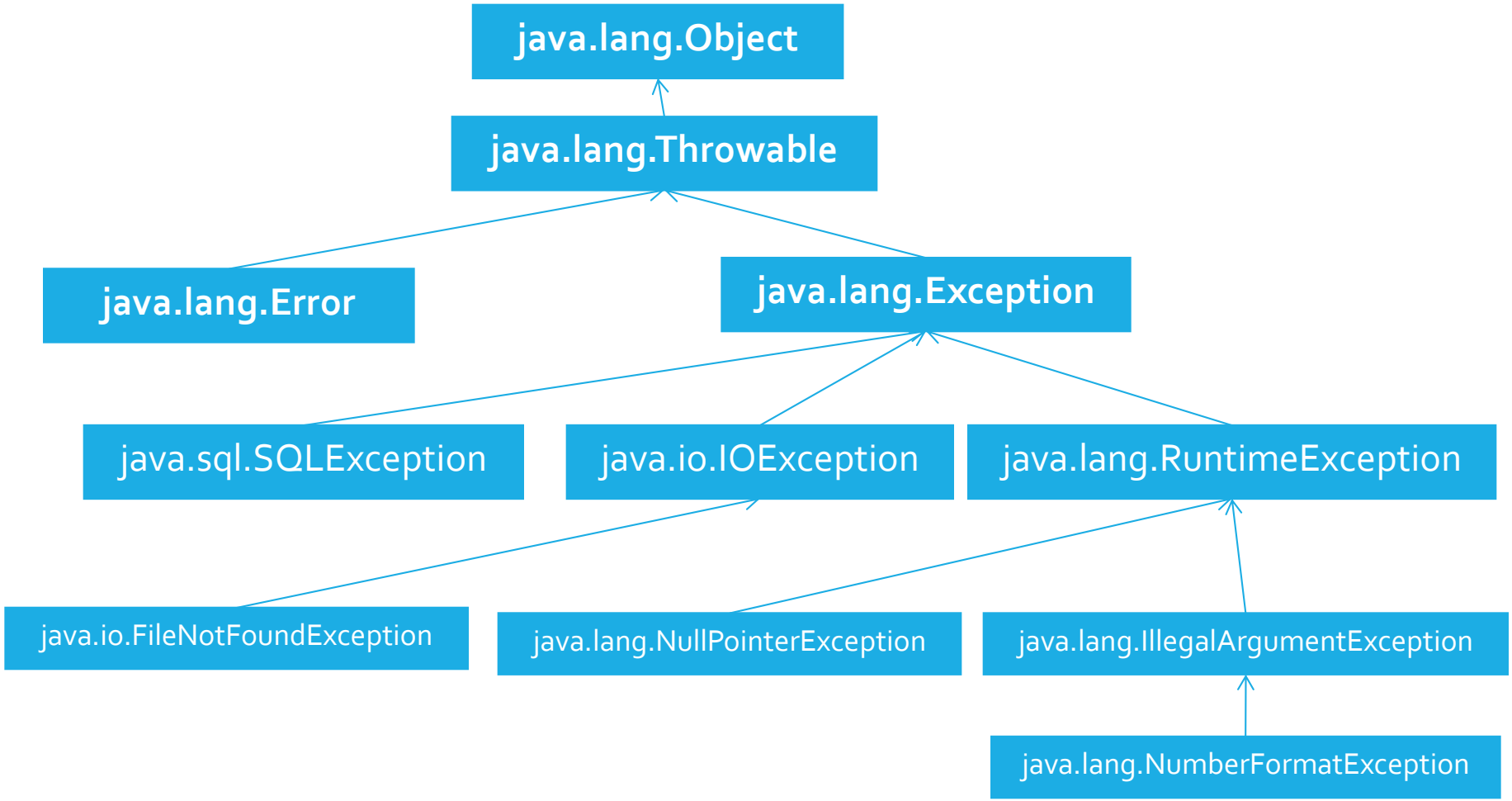
Exceptions : intérêts

- Le **traitement des exceptions** permet à une application d'avoir un comportement adéquat face à une situation inattendue.
- Beaucoup de méthodes sont susceptibles de déclencher une exception.
- Une exception peut être détectée et traitée par une clause ***try/catch*** .

Exceptions : intérêts

- L'utilisation d'une clause ***try/catch*** *est obligatoire pour les* **exceptions sous contrôle.**
- **Une exception sous contrôle est une instance d'une sous-classe de *Exception* mais pas de *RuntimeException* ➔ *voir hiérarchie des classes*.**
- Le **concepteur d'un programme java a un devoir** de capture des exceptions pour ne pas gêner les utilisateurs.
- Les exceptions permettent de renforcer la sécurité du code Java.

Exceptions : Hiérarchie des classes



Exceptions : classe *Throwable*

- ***Throwable*** est la classe de base pour le traitement des erreurs.

- ***Throwable*** possède 2 constructeurs :
  - ***Throwable***()
  - ***Throwable(String)*** : la chaîne en paramètre permet de définir un message qui décrit l'exception et qui pourra être consultée dans un bloc *catch*.

Exceptions : classe *Throwable*

- **Les principales méthodes de la classe *Throwable* sont :**
  - ***String getMessage( )*** : lecture du message
  - ***void printStackTrace( )*** : affiche l'exception et l'état de la pile d'exécution au moment de son appel
  - ***void printStackTrace(PrintStream s)*** : idem mais envoie le résultat dans un flux

Exceptions : classes *Error*, *Exception* et *RuntimeException*

- La classe ***Error*** représente une erreur grave intervenue dans la JVM.
- L'application Java s'arrête instantanément dès l'apparition d'une exception de la classe ***Error***.
- La classe ***Exception*** représente des erreurs moins graves.
- Les **exceptions** héritant de la classe ***RuntimeException*** n'ont pas besoin d'être détectées impérativement par des blocs ***try/catch***.

CHAPITRE 5 : EXCEPTIONS

2- Un premier exemple

Exceptions : Exemple d'une division par zéro

```
class DivParZero
{
  public static void main(String[] argv)
  {
        int zero=0;
        zero=2000/zero;
  }
}
```

Résultat de l'exécution :

```
Exception in thread "main"
java.lang.ArithmeticException: / by
zero at
DivParZero.main(DivParZero.java:6)
```

Commentons l'exemple :

- ***Java.lang.ArithmeticException*** est le nom complet de l'exception qui a été levée
- Un message précise la cause de cette erreur: */ by zero*
- Sont indiqués la classe, la méthode et le n° de ligne où s'est produite cette exception:

  at ***DivParZero.main(DivParZero.java:6)***

Division par zéro : détection/traitement de l'exception

- Pour éviter l'interruption du programme, on peut gérer l'exception en utilisant les blocs ***try*** et ***catch()*** de la façon suivante :

```
class DivParZero {
    public static void main(String[] argv) {
        int zero=0;

        //DETECTION
        try { zero=2000/zero; }

        //TRAITEMENT
        catch(ArithmeticException e) {
            System.out.println("Une exception arithmétiq a été levée");
            System.out.println("Message:"+e.getMessage()); }
    }
}
```

Division par zéro : détection/traitement de l'exception

- A l'exécution on obtient :

  Une exception arithmétique a été levée

  Message: / by zero

- Lors de la détection d'une erreur, un objet qui hérite de la classe ***Exception*** est créé (on dit qu'une exception est levée) et propagé à travers la pile d'exécution jusqu'à ce qu'il soit traité.

CHAPITRE 5 : EXCEPTIONS

3- Gestion des exceptions

Exceptions : Attraper une exception

- **Attraper** une exception : c'est ne pas la laisser arrêter l'exécution du programme.
- Pour attraper une exception, il faut mettre dans un bloc ***try*** le code susceptible de la lancer (générer).
- L'exception lancée dans le bloc ***try*** peut être attrapée dans le bloc ***catch()***.
- ***catch()*** : est une sorte de méthode d'un type particulier prenant pour paramètre un objet du type de l'exception lancée ou d'une classe parente.

Exceptions : Attraper une exception (*try* et *catch*)

- Si on décide de traiter une exception, on dit qu'on attrape l'exception : on utilise alors les mots clé ***try*** et ***catch*()**.
- ***try*** permet de spécifier un bloc de code au niveau duquel une exception peut être levée.
- Le bloc ***try*** doit être suivi d'un bloc ***catch()*** pour spécifier le code à exécuter pour une exception donnée.
- ***catch*()** est une sorte de méthode qui prend en argument un objet de la classe de l'exception levée.

Exceptions : Attraper une exception (*try* et *catch*)

- Les blocs ***try*** et ***catch*()** sont utilisés comme suit :

```
try {
 /*zone contenant des instructions pouvant générer
  des erreurs (lever des exceptions)*/
 }
catch(NomException e) {
 /*Traitement à faire dans le cas de l'exception e
  (erreur e a été rencontrée dans le bloc try)*/
}
```

Exceptions : Attraper une exception (*try* et *catch*)

- Un bloc ***try*** peut être suivi de plusieurs blocs ***catch()*** chacun spécifiant un type d'exception différent :

```
try {
  instruction_dangereuse1 ;
  instruction_dangereuse2 ;
}
catch(Exception1 e1) {
}
catch(Exception2 e2) {
}
```

Exceptions : Attraper une exception (*try* et *catch*)

- Si une exception est générée par une des deux instructions dangereuses alors :
  - on saute immédiatement à la fin du bloc, sans exécuter l'instruction dangereuse suivante, de la même manière que le ferait un ***break***.
  - ensuite on teste successivement les différents paramètres de ***catch()***.
  - si l'exception générée est une instance de la classe déclarée dans ***catch*()** (où d'une classe dérivée), alors on exécute le bloc qui lui est associé.

Exceptions : Mécanisme de lancement

- On suppose qu'une instruction ***i*** d'une méthode de nom ***uneMethode*** génère une erreur (ou lance une exception), alors on a deux cas :

(1) Si ***i*** se trouve dans un bloc ***try*** de ***uneMethode*** et suivi d'un bloc ***catch*()** assorti d'un argument de la classe ou d'une superclasse de l'exception levée, alors :

- les instructions qui suivent le lancement de l'exception et intérieures au bloc ***try*** sont ignorées ;
- les instructions du bloc ***catch*()** sont executées
- le **programme reprend normalement** avec l'instruction qui suit le bloc ***catch()***.

Exceptions : Mécanisme de lancement

```
uneMethode {
        try {
                i //instruction qui génère une erreur
                //instructions ignorées
        }
        catch(Exception_i e)  {
                // traitement de l'erreur de type Exception_i
        }
        // suite normale du programme
}
```

Exceptions : Mécanisme de lancement

(2) Si **l'instruction *i*** n'est pas située dans un bloc ***try***, alors :

- si ***uneMethode*** est la méthode *main*, alors le programme se termine et l'exception n'a pas été attrapée.

- sinon, on se retrouve dans la méthode qui a appelé ***uneMethode***, au niveau de l'instruction *j* qui a fait appel à ***uneMethode***. L'instruction *j* lance à son tour l'exception.

- Si une exception est levée et pas attrapée, et donc qu'elle provoque l'arrêt du programme, la pile des méthodes traversées par l'exception est indiquée à l'utilisateur : on dit qu'il y a propagation de l'exception.

Exceptions : Attraper une exception --> exemple

Exemple : calcul de moyenne d'entiers

On veut calculer une moyenne (entière) de notes entières envoyées en arguments par la ligne de commande.

Les arguments non entiers doivent être ignorés et signalés à l'utilisateur.

```
class ExceptionCatch {
    static int moyenne(String[ ] liste)  {
        int somme=0, entier, nbNotes=0;
        for (int i=0; i<liste.length; i++)  {
            try {
                entier=Integer.parseInt(liste[i]);
                somme+=entier;
                nbNotes++;
            }
            catch(NumberFormatException e) {
                System.out.println((i+1)+"ème note pas entière");
            }
        }
        return (somme/nbNotes);
    } //fin méthode moyenne                    …/…
```

```
…/…
        public static void main(String[ ] argv)        {
                System.out.println("la moyenne
 est:"+moyenne(argv));
        }
 }
```

- A l'exécution : ***java ExceptionCatch 12 a 15***
  2ème note pas entière
  la moyenne est : 13

- A l'exécution: ***java ExceptionCatch 12 a 15***

  2ème note pas entière

  la moyenne est: 13

- Si on donne en exécution: ***java ExceptionCatch 12.5 d***

  1ème note pas entière

  2ème note pas entière

Exception in thread "main" java.lang.ArithmeticException: / by zero)

At exceptionCatch.moyenne(compiled code) at ExceptionCatch.main(ExceptionCatch.java:23)

La division par zéro n'est pas attrapée.

Exception : le mot clé *throws*

- Une **méthode susceptible de lancer une exception sans l'attraper** (c-à-d contenant une instruction susceptible de générer une erreur sans que celle-ci ne soit traitée à l'intérieur de la méthode) **doit l'indiquer** dans son entête par le mot clé ***throws***.

- Ce mot clé permet d'avertir le système qu'une certaine catégorie d'exceptions ne seront pas traitées.

- Dans la déclaration d'une méthode, ***throws*** permet de déclarer la liste des classes d'exceptions que la méthode est susceptible de déclencher.

Exception : le mot clé *throws*

- Les méthodes pouvant lever des exceptions doivent inclure une clause ***throws Classe_Exception*** dans leur entête.

- L'objectif est de préciser au compilateur que la méthode pourra lever telle exception et que toute méthode qui l'appellera devra prendre en compte l'exception en question.

- Si la méthode appelante ne traite pas l'erreur ou ne la propage pas, le compilateur génère une erreur.

Exception : le mot clé *throws*

- **Java n'oblige la déclaration des exceptions** dans l'entête de la méthode que pour les exception **sous contrôle** (exceptions et erreurs n'héritant pas de ***Error*** **ni de** ***RuntimeException***).

- **Syntaxe** :

  entête_méthode throws ClasseException

  exemple : ***void methode() throws IOException***

  La méthode ***methode*** est susceptible de lancer des exceptions de type entrée/sortie.

Exception : le bloc *finally*

Le bloc ***finally*** :

- est un bloc d'instructions précédé du mot clé ***finally***.
- sera toujours exécuté après le traitement d'exception.
- est en général utilisé pour fermer des fichiers, libérer des ressources, …
- suit un bloc ***try*** ou un bloc ***catch***

Exception : exemple d'utilisation du bloc *finally*

```
class UtiliseFinally  {
        static int moyenne(String[ ] liste)  {
                int somme=0, entier,nbNotes=0,i=0;
                for (int i=0; i<liste.length; i++) {
                        try  {
                                entier=Integer.parseInt(liste[i]);
                                somme+=entier;
                                nbNotes++;
                        }
                        finally {
                                System.out.println("donnée
traitée:"+liste[i]); }
                }
                return somme/nbNotes;
        }       …/…
```

Exception : exemple d'utilisation du bloc *finally*

```
…/…
        public static void main(String[] argv)   {
                try {
                     System.out.println("Moyenne:"+moyenne(argv));
                }
                catch(NumberFormatException e) {
                    System.out.println("Erreur sur vos entiers");
                }
        }
}
```

A l'exécution: java UtiliseFinally 20 xy

On aura: *donnée traitée: 20*

donnée traitée : xy

Erreur sur vos entiers

CHAPITRE 5 : EXCEPTIONS

4- Définir sa propre exception

Exceptions : Définir sa propre exception

- Si on veut pouvoir signaler un événement exceptionnel d'un type non prévu par l'API, il faut étendre la classe ***java.lang.Exception***.

- En général, la classe étendue ne contient qu'un ou plusieurs constructeurs et éventuellement une redéfinition de la méthode ***toString()***.

Exceptions : Définir sa propre exception

- Lors du lancement de l'exception (à l'aide du mot réservé ***throw***), on crée une instance de la classe définie.

- Si l'on veut générer une exception dans une méthode avec ***throw***, il faut l'indiquer dans la déclaration de la méthode en utilisant le mot clé ***throws***.

- En **créant ses propres exceptions** :
  - **éviter** de dériver la classe ***Error***.
  - respecter la **convention de nommage** de la classe d'exception en **incluant le mot «Exception»** dans le nom de la nouvelle classe d'exception créée.

Exceptions : Définir sa propre exception

Exemple 1 :

```
class ExceptionRien extends Exception      {
/*la méthode toString() retourne une chaîne de caractère pour décrire l'objet*/
public String toString()  {
            return "Aucune note n'est valide";
      }
}

…/…
```

```
class ExceptionThrow   {
    static int moyenne(String[] liste) throws
ExceptionRien  {
            int somme=0, entier,nbNotes=0;
            for (int i=0; i<liste.length; i++) {
                    try  {
                            entier=Integer.parseInt(liste[i]);
                            somme+=entier;
                            nbNotes++;
                    }
                    catch(NumberFormatException e) {
                        System.out.println((i+1)+"ème note pas
                                                    entière");
                    }
            }               …/…
```

.../...

```
if(nbNotes==0) throw new ExceptionRien();
return (somme/nbNotes);
}
public static void main(String[] argv)  {
      try    {
            System.out.println("Moyenne
est:"+moyenne(argv));
      }
      catch(ExceptionRien e) {
            System.out.println(e); /*est équivalent à
System.out.println(e.toString());*/
      }
}
}
```

Exceptions : Définir sa propre exception

Exemple 2 (1/2) :

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

public class TestSaisieErroneeException  {
  public static void controler(String chaine) throws
  SaisieErroneeException   {
      if (chaine.equals("") == true) throw new
  SaisieErroneeException("Saisie erronée : chaine vide");  }
```

Exceptions : Définir sa propre exception

Exemple 2 (2/2) :

```
 public static void main(String[] args) {
        String chaine1 = "bonjour";
        String chaine2 = "";
        try {  controler(chaine1);  }
        catch (SaisieErroneeException e)
        {System.out.println("Chaine1 saisie erronee");}
        try {  controler(chaine2);  }
        catch (SaisieErroneeException e)
        {System.out.println("Chaine2 saisie erronee");}
 }
} //Fin de classe TestSaisieErroneeException
```

Exceptions : Définir sa propre exception

Exemple 3 (1/2) :

```
public class Pile {
    private int table [] ;     private int hauteur = 0 ;
    public Pile () { table = new int [3] ; }
    public Pile (int h) { table = new int [h] ; }
  public void insererValeur (int valeur) throws PileException {
        if (hauteur == table.length) throw new PileException
  ("Pile pleine") ;
        else table [hauteur++] = valeur ;
  }
  public int supprimerValeur() throws PileException {
       if (hauteur == 0) throw new PileException ("Pile vide") ;
       else return table [--hauteur] ;
  }     }
```

Exceptions : Définir sa propre exception

- **Exemple 3** (2/2) :

```
class PileException extends Exception {
 public PileException(String m) {  super (m) ; }
}
```

- **Utilisation**:

```
Pile pile = new Pile() ;
try {
System.out.println (pile.supprimerValeur()) ;
} catch (PileException e) {
    System.out.println (e.getMessage()) ;
}
```

SOMMAIRE

- Chapitre 1 : Concepts fondamentaux de la POO
- Chapitre 2 : Introduction à la POO avec JAVA
- Chapitre 3 : Techniques de base de la programmation en JAVA
- Chapitre 4 : Classes et Héritage en Java
- Chapitre 5 : Exceptions en Java