Bases de la POO / Java

Gestion des exceptions

Introduction

- **Forte analogie C / C++**
  - Concepts et sémantique identiques
  - En Java, les exceptions sont des **objets**
  - Classes d'exceptions standards prédéfinies
    - Classes **Exception, Throwable,…**
  - Instruction ***throw***
  - Blocs ***try*** et ***catch***

- Spécificateur ***throws***

Hiérarchie des classes Java

- Throwable Niveau 1
- Exception Niveau 2
- RuntimeException Niveau 3
  - NullPointerException
  - ArithmeticException
  - ClassCastException
  - IndexoutOfBoundsException
- et autres classes créées par le programmeur (classes dérivées de Exception)

Mécanisme

- Lancer → throw (throws)
- Surveiller → try
- Traiter → catch

Exemple


```
public class Test_1
{
  public static void main (String[] args)
  {
          int X=23, Y=0, Z;

    System.out.println ("Avant l'orage ...");
          Z=X/Y;
    System.out.println ("Apres l'orage ???");
  }
}
```

➔ Trace dans la console d'exécution

Avant l'orage ...

Exception in thread "main"
java.lang.ArithmeticException :
/ by zero at Test_1.main (Test_1.java : 6)

L'exécution du programme est interrompue !

Lancer une exception

➔ Instruction *throw*

Le lancement s'effectue par l'instruction : ***throw* objet;**

➔Contrainte et déclaration *throws*

La méthode qui lève une exception doit déclarer cette action "potentielle"

La déclaration s'effectue à la suite de la signature et avant l'accolade ouvrante de définition par :

<declaration methode> (---) ***throws*** nomClasse **{**

Exemple


```
public Segment (Point a, Point b) throws Throwable
{
        if (a.confondus(b)) throw new Throwable("-1");
        origine= a;
        extremite= b;
}
```

Surveiller les exceptions

➔ Objectifs

L'objectif de la surveillance est de capturer les exceptions éventuelles lancées par une séquence cible d'instructions.

Il n'est pas possible de surveiller sans capturer (traiter).

➔Syntaxe de la clause *try*

```
try {
    - - -                    // séquence surveillée
    - - -
}
```

La séquence surveillée peut comporter des appels de méthodes.

Capturer les exceptions

➔ Objectifs

L'objectif de la capture est d'éviter le débranchement par défaut et d'y substituer une séquence ad hoc définie par le programmeur.

La séquence exécute un traitement local, souvent partiel.

En fin d'exécution de la séquence, le programmeur peut requérir une propagation de l'exception le long de la chaîne d'appels (traitements de niveaux supérieurs).

➔Syntaxe d'une clause *catch*

```
catch ( ... ) {
        -- -                          // Séquence de traitement
}
```

La séquence de traitement peut comporter des appels de méthodes.

La séquence de traitement peut lever des exceptions.

Exemple


```
public class Test
{
        public static void main (String[] args)
        {
                Point p0= new Point();
                Point p1= new Point(0.5, 3.45);
                Point p2= new Point(0.5, 3.45);

                Segment s1 = null;
                Segment s2 = null;

                try {
                        s1 = new Segment (p0, p1);
                        s2 = new Segment (p1, p2);
                    }
                catch (Throwable t) {
                   System.out.println (t.getMessage());
                }

        }
}
```

```
-1
Process completed.
```

Propager les exceptions

➔ Propagation directe

```
catch (S1 e ) {
    -- -                          // Séquence de traitement
    throw e;
}
```

➔Propagation indirecte

Une exception est propagée indirectement par l'appel de toute méthode qui propage elle-même la classe d'exception correspondante.

Reprise après exception

➔ **Choix du langage Java**

Le mécanisme de gestion des exceptions de Java est dit **sans reprise** : une fois l'exception traitée, l'exécution du programme ne reprend pas là où l'exception a été levée…

… mais le programme poursuit son exécution après avoir traité l'exception (catch).

Exemple


```
public class Test
{
        public static void main (String[] args)
        {
                Point p0= new Point();
                Point p1= new Point(0.5, 3.45);
                Point p2= new Point(0.5, 3.45);
                Segment s2 = null;

                System.out.println ("avant !");
                try {
                        s2 = new Segment (p1, p2);
                        System.out.println (s2.toString());
                }
                catch (Throwable t) {
                   System.out.println (t.getMessage());
                }
                System.out.println ("après !");
        }
}
```

```
avant !
-1
après !

Process completed.
```