P.O.O. (Programmation Orientée Objet)

CHOUITI Sidi Mohammed

Cours pour L2 en Informatique

Département d'Informatique

Université de Tlemcen

2019-2020

Plan du cours

1. Introduction à la Programmation Orientée Objet
2. Classes et Objets
3. Héritage, polymorphisme
4. Abstraction et déclaration finale
5. Interface, implémentation et Paquetage
6. Classes Courantes en Java
7. **Gestion des Exceptions**
8. Interfaces graphiques

Introduction

Souvent, un programme doit traiter des **situations inattendues** (**exceptionnelles**) en dehors de sa tâche principale.

Une situation exceptionnelle peut être assimilée à une **erreur** qui génère une **interruption** gérée habituellement par le système d'exploitation.

Exemples d'erreurs/exceptions courantes :
Accès non autorisé à une zone mémoire (erreur de manipulation de pointeur), division par zéro, débordement d'indices dans une collection, etc.

Exemple 1

```
public class TestSansException {

    public static void main(java.lang.String[] args) {

        int i = 3;
        int j = 0;

        System.out.println("résultat = " + (i / j));

        System.out.println("ne sera jamais exécutée !");
    }
}
```

Exemple (suite)

Lors de l'exécution, la JVM va générer cette erreur (exception)

java.lang.ArithmeticException: / by zero

Et le programme **TestSansException** sera interrompu.

Définition

Une exception est un **événement**, se produisant lors de l'exécution d'un programme, qui interrompt l'enchaînement normal des instructions

L'**objectif** est de **gérer ces exceptions** par le **programme lui-même** en les **interceptant** (en les capturant).

Le **principe** consiste à **repérer les morceaux de code** (par exemple, une division) qui pourraient générer une exception, de **capturer l'exception** correspondante et enfin de **la traiter**, c'est-à-dire d'afficher un message personnalisé et de continuer l'exécution.

Les exceptions représentent un **mécanisme de gestion des erreurs**

Il se compose de (en java) :

- **Objets** représentant les erreurs
- Un ensemble de trois mots clés (une instruction) qui permettent de détecter et de traiter ces erreurs (**try, catch et finally**)

et

- Un moyen de les **lever** ou les **propager** (**throw** et **throws**).

Ce mécanisme permet de renforcer la **fiabilité** des programmes

Gestion des exceptions en java

En Java, on distingue **trois types d'erreurs** :

1. Les **erreurs graves** qui causent généralement l'arrêt du programme et qui sont représentées par la classe **java.lang.Error** .

Exemple : OutOfMemoryError

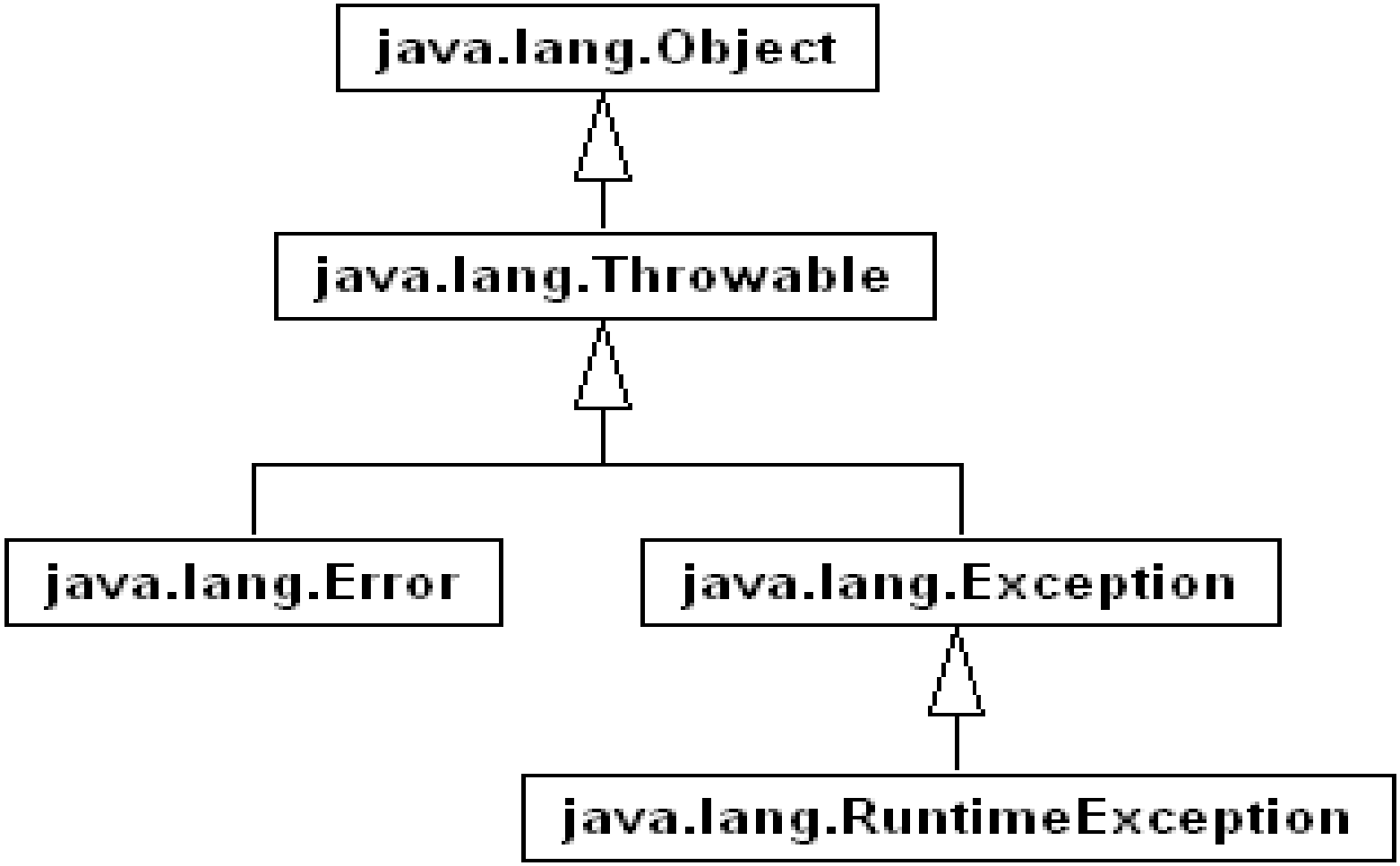
2. Les erreurs qui **doivent généralement être traitées** et qui sont représentées par la classe **java.lang.Exception**.

Exemple : FileNotFoundException

3. Les erreurs qui **peuvent ne pas être traitées** et qui sont des objets de la classe **java.lang.RuntimeException** qui hérite de **java.lang.Exception**.

Exemple : ArrayIndexOutOfBoundsExceptions

Gestion des exceptions en java



Instruction try/catch/finally

Le bloc try rassemble les blocs d'instructions susceptibles de produire des erreurs ou des exceptions.

```
try {
    bloc_susceptible_de_produire_des_erreurs
} catch (type_exception_1 arg_1) {
    bloc_1
} catch (type_exception_2 arg_2) {
    bloc_2
}...
} finally {
    bloc_optionnel_qui_s_exécute_toujours
}
```

Finally

- La clause finally définit un bloc qui sera toujours exécuté, qu'une exception soit levée ou non.
- Ce bloc est facultatif.

Exemple : Instruction try/catch/finally

```
public class TestException {
   public static void main(java.lang.String[] args) {
      int i = 3;    int j = 0;
      try {
         System.out.println("résultat = " + (i / j));
      } catch (ArithmeticException e) {
         System.out.println(" Attention ! Division par 0");
      }
    System.out.println("s'exécute sans problème");
  }
}
```

Instruction try/catch/finally

```
public class TestException {
   public static void main(java.lang.String[] args) {
      int i = 3;    int j = 0;
      try {
         System.out.println("résultat = " + (i / j));
      } catch (ArithmeticException e) {
         System.out.println("getmessage  :  "  +  e.getMessage());
         System.out.println("toString : "  + e.toString());
         System.out.println("printStackTrace : " );
         e.printStackTrace());
      }
    System.out.println("s'excute sans problème");
  }
}
```

Méthodes de l'objet Exception

getmessage() affiche le message de l'exception levée (e.g. / by zero)

toString() affiche en plus le nom complet de cette classe exception (e.g. java.lang.ArithmeticException: / by zero)

printStackTrace() : affiche l'état de la pile lors de la remontée de l'exception. Utile pour trouver les causes de celle-ci.

Gestion des exceptions en java

```
java.lang.ArithmeticException: / by zero
    at TestException.main(TestException.java:7)
    at __SHELL39.run(__SHELL39.java:6)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessor
Impl.java:62)
    at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethod
AccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:483)
    at bluej.runtime.ExecServer$3.run(ExecServer.java:730)
```

Exemple de propagation de l'exception

```
class Propagation {
        public static void m1() {     m2();   }
        public static void m2() {     m3();   }
        public static void m3() {throw new RuntimeException();}
        public static void main(String[] args) {
                m1();
        } // end of main()
}
```

Gestion des exceptions en java

```
java.lang.RuntimeException
    at Propagation.m3(Propagation.java:4)
    at Propagation.m2(Propagation.java:3)
    at Propagation.m1(Propagation.java:2)
    at Propagation.main(Propagation.java:6)
```

Quelques exceptions prédéfinies

Il existe en java des exceptions prédéfinies :

- **ArithmeticException** : levée lorsqu'une condition arithmétique exceptionnelle se produit. Par exemple, elle se manifeste lors d'une division par zéro.

- **NullPointerException** : elle se manifeste lorsqu'on cherche à utiliser un objet ou un tableau non initialisé (sur lequel on n'a pas fait de new.

- **ArrayIndexOutOfBoundsException** : il s'agit d'une erreur sur l'indice d'un tableau. On cherche à accéder à une case qui n'existe pas.

Exemple d'exceptions prédéfinies

```
class Exemple3{
   static int[] tableau = {17, 12, 15, 38, 29, 157, 89, -22, 0, 5};
   static int division(int indice, int diviseur){
         return tableau[indice]/diviseur;
   }

   public static void main(String[] args){
      int x, y;
      boolean ok = true;
      Integer i;
      java.util.Scanner scan = new java.util.Scanner(System.in);
```

Exemple (suite)

```
do{
            System.out.print("Entrez l'indice de l'entier a diviser: ");
            x = scan.nextInt();
            System.out.print("Entrez le diviseur: ");
            y = scan.nextInt();
            System.out.println("Le resultat de la division est: ");
            System.out.println(division(x,y));
            i=null;
            if (y==1) {i=y;ok=true;}
            System.out.print("Objet initialisé : "+i.toString());
}while(!ok);
   }
}
```

Aussi

- Il est possible de créer ses **propres exceptions** et de les lancer (throw) lors de l'exécution d'un programme.

- Pour créer sa propre classe d'exception, il suffit donc de créer une classe qui hérite de **java.lang.Exception**

Exemple

```
class Point {

    public static final int X_MAX = 1024, Y_MAX = 768;
    private int x, y;

    public Point (int a, int b) {
            x = a;
            y = b;
    }
}
```

Créer son propre type d'exception

en héritant de la classe Exception

```
class CoordonneesIllegalesException  extends Exception {

    public CoordonneesIllegalesException () {
        super("Coordonnées illégales."); }

    public CoordonneesIllegalesException (String msg) {
        super(msg); }
}
```

Lancer une exception

Une exception peut être lancée via la syntaxe suivante :

throw exception;

où exception est une expression dont la valeur doit être un objet de type Throwable

```
class Point {
  public static final int X_MAX = 1024, Y_MAX = 768;
  private int x, y;

  public Point2 (int a, int b) throws CoordonneesIllegalesException{
    if (a < 0 || a > X_MAX || b < 0 || b >= Y_MAX) {
      throw new CoordonneesIllegalesException("Coordonnées
                                                  illégales.");
    }
    x = a;
    y = b;
  }
}
```

```
public class TestException3 {

 public static void main(java.lang.String[] args) {

    try {
          Point2 p = new Point2(-1, 5);

    } catch(CoordonneesIllegalesException e) {
          System.out.println(e.getMessage());
     }

 }
```

Autre Exemple

Soit la classe suivante :

```
class Personne {

   private int age ; / / doit être >= 0

   public Personne ( int age ) {
     this.age = age ;
   }
   public void setAge ( int age ) {
     this.age = age ;
   }
}
```

Autre Exemple

```
class NegativeAgeException extends Exception {

    NegativeAgeException ( String message ) {
        super (message ) ;
    }
}
```

Autre Exemple

```
. . .
public void setAge ( int age ) throws NegativeAgeException {
        if ( age < 0 ) {
            throw (new NegativeAgeException ( " erreur :
                l'age doit être > 0 , valeur entrée : "+age ) ) ;
        }
        this.age = age ;
}
. . .
```

Autre Exemple

```
class TestPersonne {
    public static void main ( String [ ] args ) {
        Personne pers = new Personne ( 10 ) ;
        t ry {
            pers.setAge (-1);
        } catch ( NegativeAgeException e ) {
        e . pr intStackTrace ( ) ;
        }
    }
}
```

Autre Exemple

Résultat :

prog . NegativeAgeException : erreur : l'age doit etre > 0 ,
valeur entree : -1
at prog . Personne . setAge (Personne . java :13)
at prog . TestPersonne .main (TestPersonne . java : 6)

Encore un exemple

Ecrire un programme qui indique si la valeur saisie est valide ou non.

Pour cela, il demande un entier à l'utilisateur tant que la saisie est invalide.

Cycle de développement d'un programme java