www.dirasats.com

هذا الغلاف لا يعبر عن حقوق الملكية او فحوى الكتاب ,فهو مجرد واجهة للموقع المحمل منه

شكرا لك على ثقتك بنا وعلى اختيار موقعنا

www.dirasats.com

من اجل تواصل معنا المرجو زيارة الموقع ستجد جميع المعلومات

www.dirasats.com

Chapitre 4

La gestion des exceptions

Introduction

- Une exception est une erreur qui se produit lors de l'exécution d'un programme.
- Le langage Java offre un mécanisme qui permet de :
  1. Isoler la partie du code générant l'erreur
  2. Dissocier la détection et le traitement de cette erreur.
- Lorsqu'une erreur est **détectée**, un objet Exception est créé : on dit qu'une exception est **levée**
- Lorsqu'elle est **traitée** on dit qu'elle est **capturée**.

Rôle d'un objet exception

- Un objet exception fournit une description de l'erreur qui s'est produite
- Son rôle est de :
  - Signaler qu'une erreur s'est produite
  - Fournir un message décrivant l'erreur
  - Fournir le numéro de la ligne ou l'erreur s'est produite
  - Fournir l'enchainement des appels des fonctions ou l'erreur s'est produite (pile des appels ou Stack trace)

Exceptions prédéfinis

- Java dispose d'un ensemble d'exceptions prédéfinies
- Ces exceptions correspondent aux erreurs les plus communes
- La plupart des exceptions appartiennent au package java.lang et héritent de la classe RuntimeException
- Les exceptions plus spécialisées sont intégrées dans les packages de leurs technologies respectives

Exceptions prédéfinis

- Exemples :
  - ArithmeticException : erreurs arithmétiques comme dans la division par zéro
  - ArrayIndexOutOfBoundsException : index d'un tableau endehors de ses limites
  - NullPointerException : tentative d'utiliser un objet qui est null
  - ClassCastException : Cast invalide.

Lever une exception standard

- La détection des erreurs est faite grâce au code suivant : **throw new ClasseDeLexception()**
- Exemple :
  - Si nous détectons une division par zéro :

```
If (y==0)
    throw new ArithmeticException(" division par zéro ")
```

Lever une exception standard

- Une méthode qui lève une exception doit l'indiquer dans sa signature comme suit :
  - …méthode(…) **throws ClasseDeLexception**
- Ceci permet à l'appelant de traiter les exceptions levées
- Exemple :

```
public int diviser(int x, int y)   throws    ArithmeticException   {
        if (y==0)
                throw new ArithmeticException();
        return (x/y);
}
```


Lever une exception personnalisée

- Il existe des situations ou nous aimerions détecter un type personnalisé d'erreurs
- Exemple :
  - Une note en dehors de l'intervalle [0,20]
  - Un nom ne respectant pas une forme particulière
  - …etc
- Dans ce cas il faut définir une classe pour cette exception.
- Cette dernière dois hériter de la classe **Exception**.

Lever une exception personnalisée

- Exemple :

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

- Elle s'utilise de la même façon qu'une exception standard :

```
if (….) throw new MaPropreException("mon message");
```


Lever une exception personnalisée

- Exemple :

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

Lever une exception personnalisée

- Exemple d'Utilisation :

```
public void setNote (float note) throws NoteDebordanteException {
 if ((note>20)||(note<0))
 throw new NoteDebordanteException(" Erreur  note qui déborde" )
}
```


Comportement face à une exception

- Soit une méthode *methode1()* qui lève une exception
- Et soit une méthode *methode2()* qui appelle *methode1()*
- Methode2 a deux choix :
  1) **Ne pas traiter l'exception** : Dans ce cas elle doit déclarer l'exception dans sa signature avec « throws »
  2) **Capturer ou traiter l'exception**

Capture d'une exception

- La gestion d'une exception se fait selon le schéma suivant :

```
try{
   appel de la fonction susceptible de générer  l'exception
}
catch (Exception e){
   traiter l'exception e
}
instruction suivante
```


Capture d'une exception

- Exemple :

```
try {        c= diviser(a,b);        }
 catch (ArithmeticException e) {
     System.out.println("Erreur : Division par zéro ");
 }
 System.out.println("résultat = " + c);
```

- Remarque :
  - En cas d'exception les instructions qui suivent le lancement de l'exception sont ignorées.

Capture de plusieurs exceptions

- Cas d'une instruction levant plusieurs exceptions :

```
try {
fonction susceptible de générer l'exception }
catch (Exception1 e){ traitement 1 }
catch (Exception2 e){traitement 2 }
…
catch (ExceptionN e){traitement N }
instruction suivante
```


Capture de plusieurs exceptions

- Depuis la version 7 de Java, il est possible de réécrire le code précédent sous la forme :

```
try {
fonction susceptible de générer l'exception }
catch (Exception1 | ... | ExceptionN e){
   traitement ;
}
instruction suivante
```

Capture de plusieurs exceptions

- Cas de plusieurs instructions levant plusieurs exceptions :

```
try {   }
catch (){  }
try {   }
catch (){  }
try {   }
catch (){  }
…
```


Traitement de plusieurs exceptions

- Remarque :
  - Dans les clauses catch il faut traiter exceptions les plus précises en premier les avant les exceptions plus générales.
  - Un message d'erreur est émis par le compilateur dans le cas contraire.

Traitement de plusieurs exceptions

- Exemple:

```
public class TestException {
    public static void main(String[] args) {
      int i = 3;
      int j = 0;
      try { System.out.println("résultat = " + (i / j)); }
      catch (ArithmeticException | Exception e) { }
    }
}
```


Le bloc finally

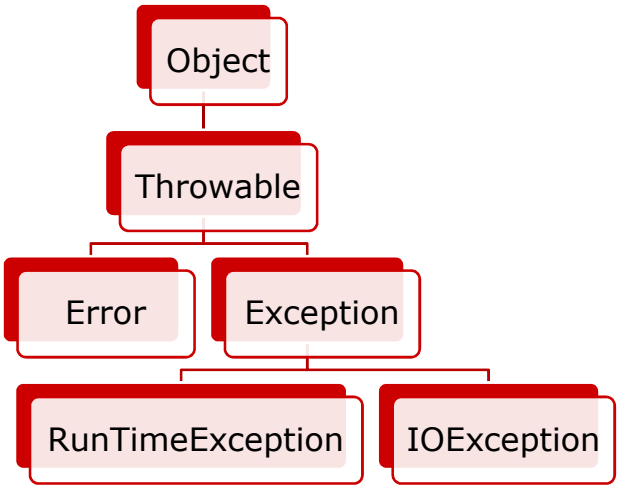
- Certaines ressources peuvent rester non libérée après qu'une exception sois levée.
- Pour forcer le compilateur à exécuter certaines instructions, qu'il y ait exception ou non, ils doivent être placées dans un bloc **finally**.
- Remarque :
  - Ce bloc est facultatif !

Le bloc finally

- Structure

```
try {
 }
catch (){
}
finally {
}
instruction suivante
```


Hiérarchie des exceptions



Hiérarchie des exceptions

- La classe Throwable
  - Classe de base pour le traitements des erreurs.
  - Possède quatre constructeurs :
    - **Throwable()** : constructeur par défaut
    - **Throwable(String)** : constructeur avec un message qui décrit l'exception.
    - **Throwable(Throwable cause)** : dans le cas ou l'exception est causée par une autre exception.
    - **Throwable(String message, Throwable cause)** : constructeur avec message et cause.

Hiérarchie des exceptions

- Principales méthodes :
  - String **getMessage( )** : lecture du message
  - void **printStackTrace( )** : affiche l'exception et l'état de la pile d'exécution au moment de son appel

Hiérarchie des exceptions

- **Exemple**:

```
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());
        e.printStackTrace();
     }
    }
}
```

Résultat :

```
getmessage  = / by zero
toString  = java.lang.ArithmeticException: / by zero
printStackTrace = java.lang.ArithmeticException: / by zero at
tests.TestException.main(TestException.java:24)
```


Hiérarchie des exceptions

- Classes **Exception**, **RunTimeException** et **Error**
  - La classe **Error** représente une erreur grave intervenue dans la machine virtuelle Java ou dans un sous système Java.
  - 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 classe **RuntimeException** n'ont pas besoin d'être détectées impérativement par des blocs try/catch.

Contrôle des exceptions

- Lorsque Java oblige la déclaration des exceptions dans l'en tête de la méthode, ces exceptions sont dites contrôlées (checked).
- Message de Java : « nom-exception must be caught or it must be declared in the throws clause of this method ».

Contrôle des exceptions

- Les exceptions et erreurs qui héritent de RunTimeException et de Error sont non contrôlées.
- Toutes les autres exceptions sont contrôlées.
- Les exceptions non contrôlées (unchecked) peuvent être capturées mais n'ont pas a être déclarées.

Chaînage des exceptions

- Dans le traitement d'une exception on ne se contente pas que d'afficher le message d'erreur.
- Souvent on a recours à un traitement durant lequel nous faisons appel à une instruction qui peut lever une autre exception.
- Cette nouvelle exception doit être liée avec l'exception d'origine pour conserver l'empilement des exceptions levées durant les traitements.

Chaînage des exceptions

- Il y a deux façons de chaîner deux exceptions :

1) Utiliser la surcharge du constructeur de Throwable qui attend un objet Throwable en paramètre représentant la cause.
Exemple :

```
catch(Exception1 e) {
    ....
    throw new Exception2(e);
    // ou throw new Exception2 ("un message" , e);
 }
```

Chaînage des exceptions

2) Utiliser la méthode initCause() d'une instance de Throwable

Exemple :

```
catch(Exception1 e) {
    throw (Exception2) new Exception2().initCause(e);
}
```


Chaînage des exceptions

- **Exemple** :

```
package testchainageexception;

public class MonException extends Exception{
   // constructeur 1
   MonException(String message){
         super(message);
   }
  // constructeur 2
   MonException(String message , Throwable cause){
         super(message, cause);
   }
}
```

Chaînage des exceptions

```
package testchainageexception;
public class TestChainageException {
  public static int DivisionEntiere (int x,int y) throws MonException{
    int resultat=0;
    try { resultat=x/y; }
    catch(ArithmeticException e){
      throw new MonException("Attention Il y a une exception !!",e);
    }
    return resultat;
    }
```


Chaînage des exceptions

```
public static void main(String[] args) {
      int a=3,b=0,z;
      try { z=DivisionEntiere(a, b); }
      catch(MonException e){
         e.printStackTrace();
      }
  }
}
```

Chaînage des exceptions

- Résultat :

testchainageexception.MonException: Attention Il y a une exception !! **at** testchainageexception.TestChainageException.DivisionEntiere(TestChainageException.java:27) **at** testchainageexception.TestChainageException.main(TestChainageException.java:15)

Caused by: java.lang.ArithmeticException: / by zero **at** testchainageexception.TestChainageException.DivisionEntiere(TestChainageException.java:25)

... 1 more

Chaînage des exceptions

- La méthode getCause() héritée de Throwable permet d'obtenir l'exception qui est à l'origine de l'exception.
- Exemple :

```
public static void main(String[] args) {
      int a=3,b=0,z;
      try { z=DivisionEntiere(a, b); }
      catch(MonException e){
        System.out.println( e.getCause().getMessage());
      }
  }
```

- Résultat : **/ by zero**