



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 6

La gestion des exceptions

Introduction

- Soit le programme suivant :

```
public class Division {  
    public static int diviser(int x, int y) {  
        return (x/y);  
    }  
    static void main (String [] args) {  
        int a=1,b=0;  
        int c= diviser(a,b);  
        System.out.println("res: " + c);  
    }  
}
```

Notion d'exception

- ❑ Le système détecte une erreur à l'exécution et affiche le message suivant:
 - **Exception in thread "main" java.lang.ArithmeticException: / by zero at Division.diviser(Division.java:3) at Division.main(Div.java:7)**
- ❑ Ces erreurs sont appelés Exceptions en Java.
- ❑ Une exception est une erreur qui se produit lors de l'exécution d'un programme.

Gestion classique des erreurs

- ❑ Dans l'exemple précédent, nous aurions pu écrire :

```
if (y==0) { // traitement de l'erreur.  
           // par exemple : return (-1)  
}  
else {    return(x/y);  
}
```
- ❑ Problèmes :
 - Ce traitement devient fastidieux à la longue!
 - Code métier & traitement des erreurs deviennent mélangés

La gestion des exceptions en Java

- 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.
- Lors de la détection d'une erreur, un objet Exception est créé : on dit qu'une exception est levée
- Lorsqu'elle est traitée on dit qu'elle est capturée.

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 d'une exception

□ Exemple 2 :

```
int x;
String ch;
...
try { x= Integer.parseInt(ch);          }
catch (NumberFormatException e) {
    System.out.println("Erreur : Enter un entier");
}
System.out.println("X= " + x);
```

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

- Cas de plusieurs instructions levant plusieurs exceptions :

```
try { }  
catch (){ }  
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 e)  
        { }  
        catch (Exception e)  
        { }  
    }  
}
```

Le bloc finally

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

Le bloc finally

- ❑ Structure

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


Le bloc finally

❑ Exemple :

```
public class Division3 {  
    public static int diviser (int x, int y) { return (x/y); }  
    static void main (String [] args) {  
        int c=0,a=1,b=0;  
        try { c= diviser(a,b); }  
        catch (ArithmeticException e) {  
            System.out.println("Erreur a été capturée"); }  
        finally { System.out.println("bloc finally"); }  
        System.out.println("res: " + c);  
    }  
}
```

Java - Dr A. Belangour

355

Lever une exception standard

- ❑ Java dispose d'une multitude de classes exceptions héritant de la classe Exception.
- ❑ Lorsqu'une erreur est détectée un objet représentant une de ces exception est levée grâce à l'instruction throw.

❑ Exemple :

- if (y==0) throw new ArithmeticException();

Java - Dr A. Belangour

356

Lever une exception standard

- Une fonction qui lève une exception sans la traiter doit la déclarer dans sa signature grâce à l'instruction `throws`.

- Exemple :

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

Lever une exception standard

- Exemple complet :

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

Lever une exception standard

```
public static void main (String [] args) {  
    int c=0,a=1,b=0;  
    try {  
        c= diviser(a,b);  
    }  
    catch (ArithmeticException e) {  
        System.out.println("Erreur a été capturée");  
    }  
    System.out.println("res: " + c);  
    System.exit(0);  
}
```

Java - Dr A. Belangour

359

Lever une exception personnalisée

- ❑ Pour lever une exception personnalisée, il faut commencer par définir la classe de cette exception.
- ❑ Cette dernière doit hériter de la classe Exception.
- ❑ Exemple :

```
class MonArithmeticException extends Exception {  
    // quelque chose ...  
}
```

Java - Dr A. Belangour

360

Lever une exception personnalisée

- L'utilisation de cette classe est comme suit :

```
public class Division4 {  
    public static int diviser(int x, int y) throws  
        MonArithmeticException {  
        if (y==0) throw new MonArithmeticException();  
        return (x/y);  
    }  
}
```

Lever une exception personnalisée

```
public static void main (String [] args) {  
    int c=0,a=1,b=0;  
    try {  
        c= diviser(a,b);  
    }  
    catch (MonArithmeticException e) {  
        System.out.println("Erreur a été capturée");  
    }  
    System.out.println("res: " + c);  
    System.exit(0);  
}
```

Lever une exception personnalisée

- Pour personnaliser le message d'erreur, on peut passer par le constructeur de la classe Exception.

- Exemple:

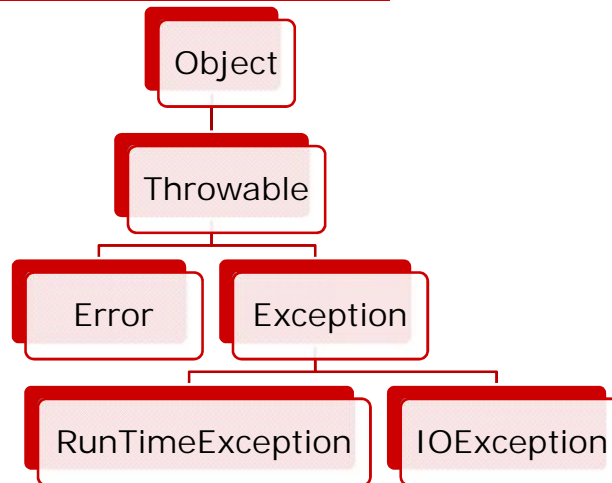
```
public class MonArithmeticException extends Exception {  
    public MonArithmeticException(String message){  
        super(message);  
    }  
}
```

Lever une exception personnalisée

- Exemple d'utilisation :

```
public class Divison7 {  
    public static int divint (int x, int y) throws  
        MonArithmeticException {  
        if (y==0) throw new MonArithmeticException( "Erreur a été  
        capturée dans MonArithmeticException");  
        return (x/y); }  
    public static void main (String [] args) {  
        int c=0,a=1,b=0;  
        try { c= divint(a,b); }  
        catch (MonArithmeticException e) {  
            System.out.println(e.getMessage()); }  
        System.out.println("res: " + c);  
        System.exit(0);  
    }  
}
```

Hiérarchie des exceptions



Java - Dr A. Belangour

365

Hiérarchie des exceptions

□ La classe Throwable

- Classe de base pour le traitements des erreurs.
- Possède quatre 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.
 - Throwable(Throwable cause) : dans le cas ou l'exception est causée par une autre exception.
 - Throwable(String message, Throwable cause)
- 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

Java - Dr A. Belangour

366

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 :

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

Java - Dr A. Belangour

367

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.

Java - Dr A. Belangour

368

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 à 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 !! 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 !! y a une exception !! at
testchainageexception.TestChainageException.DivisionEntiere(TestChainage
Exception.java:27) at
testchainageexception.TestChainageException.main(TestChainageException
.java:15)
Caused by: java.lang.ArithmeticException: / by zero at
testchainageexception.TestChainageException.DivisionEntiere(TestChainage
Exception.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