

# Chapitre 5

## Les Exceptions

Programmation Java 2003/2004 - D.Pastre

Chap.5.1

## Les Exceptions

Elles servent à gérer et contrôler les erreurs, ou simplement ce qui n'est pas souhaitable.

- Indication du type d'erreur
- de l'endroit où elle a été détectée
- possibilité de continuer le programme
- possibilité d'effectuer un traitement particulier

Les Exceptions sont des classes

- sous-classes de la classe **Throwable**
- prédéfinies dans le paquetage **java.lang**
- ou
- définies par l'utilisateur

Programmation Java 2003/2004 - D.Pastre

Chap.5.2

Object

- Throwable
  - Exception
    - RuntimeException
      - ArithmeticException
      - IllegalArgumentException
        - NumberFormatException
      - IndexOutOfBoundsException
      - NullPointerException
    - IOException
      - FileNotFoundException
      - EOFException
  - Error
    - VirtualMachineError
    - OutOfMemoryError
    - StackOverflowError

## Hierarchie des exceptions prédéfinies

Programmation Java 2003/2004 - D.Pastre

Chap.5.3

**En cas d'erreur**, on a, par défaut

- le nom de l'exception qui a été *levée* et la cause de l'erreur
- la méthode qui a *levé* l'exception et le numéro de ligne
- les méthodes appelantes

Exemple dans la classe Rationnel, une erreur a été *levée* par l'instruction  $a\%b$  de la méthode *pgcd* avec  $b=0$

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at rationnel.Rationnel.pgcd(Rationnel.java:27)
at rationnel.Rationnel.simplifier(Rationnel.java:33)
at rationnel.Rationnel.<init>(Rationnel.java:11)
at rationnel.TestRationnelException.main(
                                TestRationnelException.java:19)
et le programme s'arrête
```

Programmation Java 2003/2004 - D.Pastre

Chap.5.4

On peut *attraper* et *traiter* cette erreur dans le programme appelant par le bloc *try ... catch ...* et le programme continuera à s'exécuter

```
try {  
  r=new Rationnel(15,0);  
} catch(Exception e){  
  System.out.println(e+  
    "\n il y a une erreur à la construction du rationnel ");}
```

```
java.lang.ArithmeticException: / by zero  
il y a une erreur à la construction du rationnel  
  
et le programme continue
```

On peut aussi, au lieu d'attendre que la machine *lève* elle-même l'exception quand elle la détecte dans le *pgcd*, la *lever* dans le constructeur par l'instruction *throw*

```
public Rationnel(int n,int d){  
  if (d==0) throw new ArithmeticException(  
    "le dénominateur ne peut être nul");  
  this.num=n;this.den=d;this.simplifier();  
}
```

```
java.lang.ArithmeticException: le dénominateur ne peut être nul  
il y a une erreur à la construction du rationnel  
  
et le programme continue
```

Enfin, si on *lève* l'exception explicitement par *throw* mais qu'on ne l'*attrape* pas

```
Exception in thread "main" java.lang.ArithmeticException:
    le dénominateur ne peut être nul
at rationnel.Rationnel.<init>(Rationnel.java:10)
at rationnel.TestRationnelException.main(
    TestRationnelException.java:19)
```

*et le programme s'arrête*

Pour les exceptions autres que *RuntimeException* et *Error* (et leurs sous-classes), par exemple *IOException* et les exceptions que l'on définira, il faudrait alors *déclarer* que l'exception n'est pas attrapée, par le mot-clef *throws* (sinon erreur à la compilation)

On peut aussi donner beaucoup plus d'informations dans le *catch*, en utilisant des méthodes de la classe *Throwable*, par exemple :

- le message associé à l'exception par la méthode *getMessage* (dans l'exemple *" / by zero "*)
- la pile des appels jusqu'à l'endroit où l'exception est *levée* par la méthode *printStackTrace* (c'est ce qui est fait par défaut).

On peut aussi définir ses propres exceptions comme sous-classes des exceptions prédéfinies

## La classe Throwable

```
public class java.lang.Throwable extends java.lang.Object
```

### *Constructeurs*

#### **Throwable()**

Constructs a new *Throwable* with *null* as its error message string.

#### **Throwable(String message);**

Constructs a new *Throwable* with the specified error message.

### *Quelques méthodes*

#### **String getMessage();**

Returns the error message *String* of this *Throwable* object.

#### **void printStackTrace();**

Prints this *Throwable* and its backtrace to the standard error stream.

#### **String toString();**

Returns a short description of this *Throwable Object*.

## La classe Exception

```
public class java.lang.Exception extends java.lang.Throwable
```

### *Constructeurs*

#### **Exception();**

Constructs an *Exception* with no specified detail message.

#### **Exception(String s);**

Constructs an *Exception* with the specified detail message.

On peut définir ses propres classes d'exception comme sous-classes de la classe *Exception* en redéfinissant les constructeurs et éventuellement une méthode *toString()*

## Définition d'une classe d'exception

### Exemple pour la classe Compte

```
public class DecouvertException extends Exception {  
    public DecouvertException(double montant, Compte c){  
        super("Depassement du découvert autorisé : operation impossible\n"+  
            "montant= "+montant+" solde="+c.getSolde()+  
            " découvert autorisé="+c.getDecouvertAutorise());  
    }  
}
```

## Lever l'exception

```
public void addOperation(double montant, int moyenPaiement)  
    throws DecouvertException {  
    if(montant>=0 || -montant<=this.solde+this.decouvertAutorise) {  
        this.historique.addElement(this.getString(montant, moyenPaiement));  
        this.solde += montant;  
    }  
    else throw new DecouvertException(montant,this);  
}
```

## Attraper l'exception

```
public static void main(String[] tArg) {  
    ...  
    try {  
        ...  
        c.addOperation(..., ...);  
        ...  
    } catch (DecouvertException e) {  
        System.out.println( "Operation ignoree : "+  
                             "compte non suffisamment approvisionné"+e);  
    }  
    ...  
}
```

## Exécution

*début du programme*

*puis, si l'exception est levée :*

Operation ignoree : compte non suffisamment approvisionné  
Comptes.DecouvertException: Dépassement du découvert autorisé:  
opération impossible  
Montant= ... solde= ... découvert autorisé= ...

*suite du programme*

## Variante

```
public class DecouvertException extends Exception {
    private double montant;
    private compte c;
    public DecouvertException(double montant,Compte c){
        super();
        this.montant=montant;
        this.compte=compte;
    }
    public String toString() {
        return("Depassement du découvert autorise : operation impossible\n"
            + " montant= "+this.montant+" solde="+this.c.getSolde()
            + " decouvert autorise="+this.c.getDecouvertAutorise());
    }
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.5.15

## Définition de deux classes d'exception

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

    class DecouvertException extends CompteException {
        public DecouvertException(double montant,Compte c){
            super("Depassement du découvert autorise : operation impossible\n"+
                "montant= "+montant+" solde="+c.getSolde()+
                " decouvert autorise="+c.getDecouvert());} }

    class MontantNulException extends CompteException {
        public MontantNulException() {super("Montant nul");}
    }
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.5.16



## Lever deux exceptions

```
public void addOperation(double montant, int moyenPaiement)
    throws DecouvertException{
    if (montant != 0)
        if(montant>=0 || -montant<=this.solde+this.decouvertAutorise) {
            this.historique.addElement(this.getString(montant, moyenPaiement));
            this.solde += montant;}
        else throw new DecouvertException(montant,this);
    else throw new MontantNulException();
}
```

## Attraper deux exceptions

```
public static void main(String[] tArg) {
    ...
    try { ... ; c.addOperation(...,...); ... }
    catch (DecouvertException e) {
        System.out.println( "Operation ignoree : "+
            "compte non suffisamment approvisionné"+e);
    }
    catch (MontantNulException e) {
        System.out.println( "Operation ignoree : "+
            "rentrez des dnombres non nuls");
    }
    catch (CompteException e) {}
    ...
}
```

## Exercices

On peut reprendre tous les exercices des chapitres précédents en améliorant la programmation par des exceptions pour gérer les erreurs possibles et les cas exceptionnels.