

Part 6

LES EXCEPTIONS

Qu'est-ce qu'une exception?

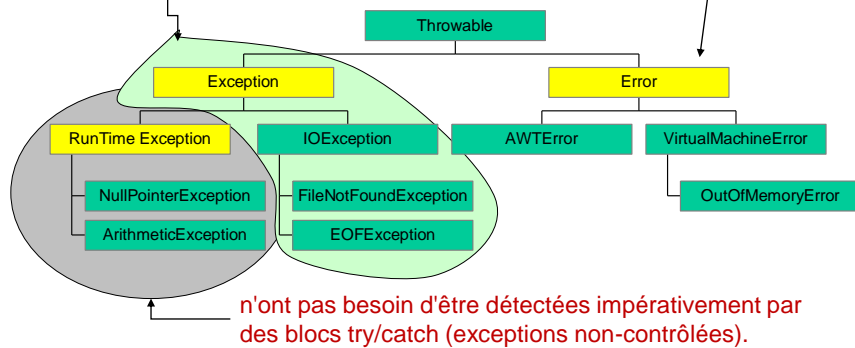
- Les exceptions représentent le **mécanisme de gestion des erreurs** intégré au langage Java.
- Une **exception** est un événement exceptionnel risquant de compromettre le bon déroulement du programme.
- **Lancer** (**throw**) une exception consiste à signaler cet événement.
- **Interceptor/Attraper** (**catch**) une exception permet d'exécuter les actions nécessaires pour traiter cette situation.

Hiérarchie des exceptions

- En JAVA les **exceptions sont des objets**: toute exception doit être une instance d'une sous-classe de la classe **java.lang.Throwable**

On doit traiter ces exceptions-ci
(exceptions contrôlées)

On ne peut pas traiter les
« Error »



Les exceptions de RuntimeException

- Les sous-classes de **RuntimeException** sont définies dans les packages **java.lang** et **java.util**.

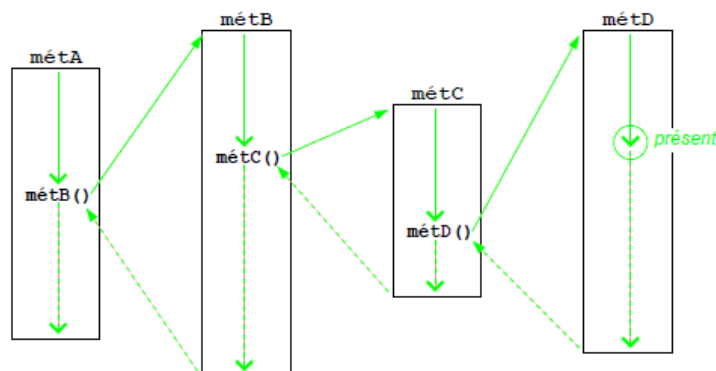
java.lang	
ArithmeticException	- Division par zéro.
ArrayStoreException	- Incompatibilité de type lorsqu'on tente de stocker un objet comme composante dans un tableau.
ClassCastException	- Conversion de type d'objet invalide.
IllegalArgumentException	- Passage d'un argument à une méthode dont le type est invalide.
IllegalMonitorStateException	- Voir le concept de fils d'exécution (« threads »).
IndexOutOfBoundsException	- Usage d'un indice hors de portée d'un tableau, d'un objet de la classe String ou de la classe Vector .
NegativeArraySizeException	- Définition d'un tableau de dimension négative.
NullPointerException	- Accès à un membre d'un objet via une variable de valeur null.
SecurityException	- Violation aux règles de sécurité.
java.util	
EmptyStackException	- Opération illégale sur une pile vide dans la classe Stack .
NoSuchElementException	- Usage de la méthode nextElement() (voir l'interface Enumeration) lorsque l'élément suivant n'existe pas.

Mécanisme des exceptions

- Détection de l'anomalie et construction d'une Exception qui la décrit.
- La méthode qui a construit l'exception la lance (**throw**) vers celle qui l'a appelée et se termine immédiatement.
- L'exception « remonte » de méthode appelée en méthode appelante, chaque méthode ainsi traversée se terminant immédiatement,
 - jusqu'à une méthode où on a prévu d'attraper (**catch**) l'exception (et, en principe, de résoudre le problème qu'elle pose),
 - ou, sinon, jusqu'à terminer le programme.

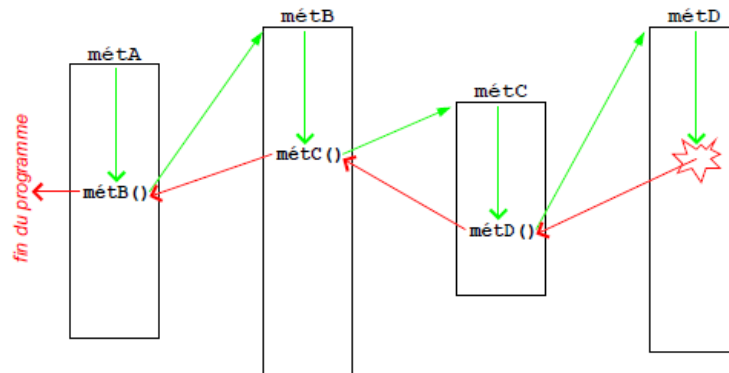
Mécanisme

- Déroulement normal d'un appel de la méthode metA



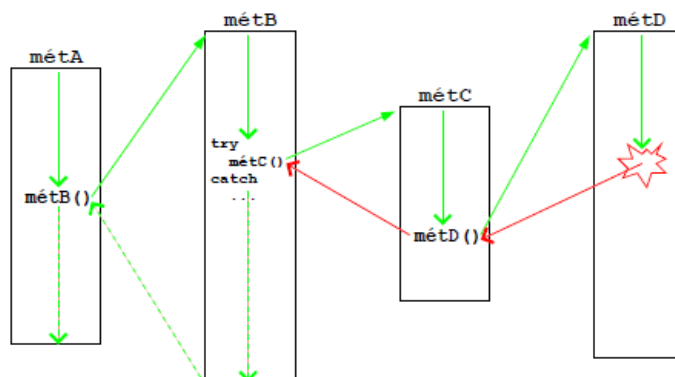
Mécanisme

- Lancement d'une exception (non attrapée) dans metD



Mécanisme

- Lancement d'une exception dans metD attrapée dans metB



Exemples:

- Exemple d'exception non contrôlée dérivant de *Error* :
 - [RecursiviteInfinie.java](#)
- Exemple d'exception dérivant de *RuntimeException* :
 - [ExceptionNonControleeDuRuntime.java](#)

Traitement des exceptions

- Lorsque que l'on utilise une méthode susceptible de soulever une exception, il faut spécifier comment on la gère.
- Une méthode dont le corps est susceptible de lever une exception doit :
 - soit intercepter l'exception et la traiter (par un **try ... catch**)
 - soit faire "remonter" (propager) l'exception au niveau supérieur : **Throws**


Debugging : les traces

- Une exception dérive de la classe **Object**. Utilisez la méthode **toString()** pour garder des traces d'exécution.
- La méthode **PrintStackTrace()** permet d'afficher la pile des appels depuis la méthode qui a déclenché l'exception.
- La classe **Throwable** prend un String à sa construction. Cela permet d'enrichir les traces avec des messages spécifiques, récupérés avec **getMessage()**.

Programme sans gestion de l'exception

```
class Action1{
    private int x;
    public void meth(){
        System.out.println("... avant incident");
        x = 1/0;
        System.out.println("... après incident");
    }
}

public class UseAction1 {
    public static void main(String args[]){
        Action1 obj = new Action1();
        System.out.println("Début du programme.");
        obj.meth();
        System.out.println("Fin du programme.");
    }
}
```

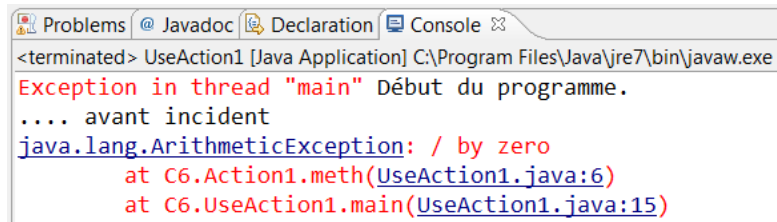


Sortir du bloc

Sortir du bloc

Programme sans gestion de l'exception

- Lors de l'exécution, après avoir affiché les chaînes "Début du programme" et " ...Avant incident", le programme s'arrête et la java machine signale une erreur.
- Voici ci-dessous l'affichage obtenu sur la console lors de l'exécution :



The screenshot shows a Java IDE console window with the following text:

```
<terminated> UseAction1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Exception in thread "main" Début du programme.
.... avant incident
java.lang.ArithmeticException: / by zero
    at C6.Action1.meth(UseAction1.java:6)
    at C6.UseAction1.main(UseAction1.java:15)
```

Programme avec gestion de l'exception

- Java possède une **instruction de gestion des exceptions**, qui permet d'intercepter des exceptions dérivant de la classe Exception :

try ... Catch

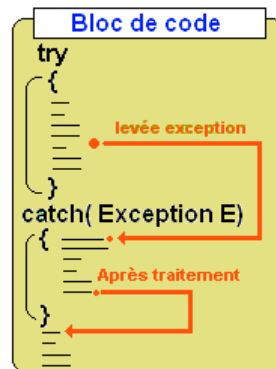
- Syntaxe minimale d'un tel gestionnaire :

```
try {
    <lignes de code à protéger>
}catch ( UneException e ) {
    <lignes de code réagissant à l'exception UneException >
}
```

Le type **UneException** est obligatoirement une classe qui **hérite de la classe Exception**.

Schéma du fonctionnement du `try ...catch`

- Le gestionnaire d'exception "déroute" l'exécution du programme vers le bloc d'interception `catch` qui traite l'exception, puis renvoie et continue l'exécution du programme vers le code situé après le gestionnaire lui-même.



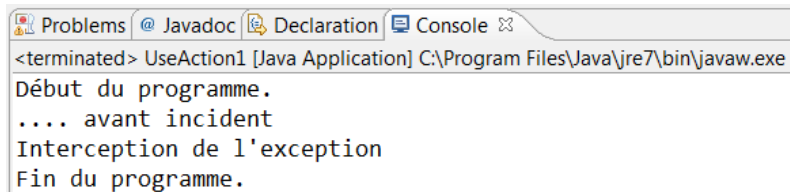
Programme avec gestion de l'exception

```
class Action1{
    private int x;
    public void meth(){
        System.out.println(".... avant incident");
        x = 1/0; ← Engendre une exception
        System.out.println(".... après incident");
    }
}

public class UseAction1 {
    public static void main(String args[]){
        Action1 obj = new Action1();
        System.out.println("Début du programme.");
        try{
            obj.meth(); ← Levée d'ArithmeticException
        }catch(ArithmeticException e){ ← 
            System.out.println("Interception de l'exception");
        }
        System.out.println("Fin du programme."); ← Traitement puis poursuite de l'exécution
    }
}
```


Programme avec gestion de l'exception

- Voici ci-dessous l'affichage obtenu sur la console lors de l'exécution :



```
<terminated> UseAction1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Début du programme.
.... avant incident
Interception de l'exception
Fin du programme.
```

Interceptions de plusieurs exceptions

- Dans un gestionnaire `try...catch`, il est en fait possible d'intercepter plusieurs types d'exceptions différentes et de les traiter.

```
try {
    < bloc de code à protéger >
}
catch ( TypeException1 e ) { <Traitement TypeException1 > }
catch ( TypeException2 e ) { <Traitement TypeException2 > }
.....
catch ( TypeExceptionk e ) { <Traitement TypeExceptionk > }
```

Où `TypeException1`, `TypeException12`, ... , `TypeExceptionk` sont des classes d'exceptions obligatoirement toutes **distinctes**.

Seule une seule clause `catch (TypeException E) { ... }` est exécutée (celle qui correspond au bon type de l'objet d'exception instancié).

Interceptions de plusieurs exceptions

- Supposons que la méthode `meth()` puisse lever 3 types différents d'exceptions: `ArithmeticException`, `ArrayStoreException`, `ClassCastException`.

```
public class UseAction2 {
    public static void main(String args[]){
        Action2 obj = new Action2();
        System.out.println("Début du programme.");
        try{
            obj.meth();
        }catch(ArithmeticException e){
            System.out.println("Interception ArithmeticException");
        }catch(ArrayStoreException e){
            System.out.println("Interception ArrayStoreException");
        }catch(ClassCastException e){
            System.out.println("Interception ClassCastException");
        }
        System.out.println("Fin du programme.");
    }
}
```

Prof. Asmaa El Hannani

ENSA-El Jadida

318

Déclenchement manuel d'une exception

- JAVA peut soulever une exception automatiquement comme dans l'exemple de la levée d'une `ArithmeticException` lors de l'exécution de l'instruction "`x = 1/0 ;`".
- Mais JAVA peut aussi soulever une exception à votre demande suite à la rencontre d'une instruction **`throw`**.

Prof. Asmaa El Hannani

ENSA-El Jadida

319

Déclenchement manuel d'une exception

```
class Action3{
    private int x;
    public void meth(){
        System.out.println(".... avant incident");
        if(x==0)
            throw new ArithmeticException("Mauvais calcul!");
        System.out.println(".... après incident");
    }
}

public class UseAction3 {
    public static void main(String args[]){
        Action3 obj = new Action3();
        System.out.println("Début du programme.");
        try{
            obj.meth();
        }catch(ArithmeticException e){
            System.out.println("Interception de l'exception: " + e.getMessage());
        }
        System.out.println("Fin du programme.");
    }
}
```

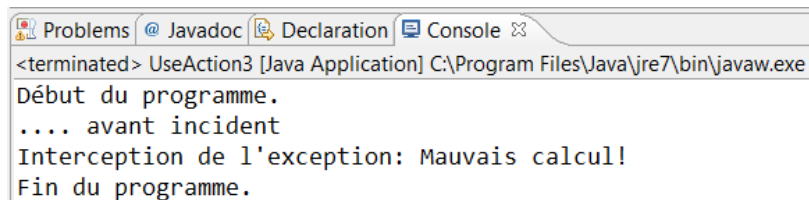
Prof. Asmaa El Hannani

ENSA-El Jadida

320

Déclenchement manuel d'une exception

- Voici ci-dessous l'affichage obtenu sur la console lors de l'exécution :



```
<terminated> UseAction3 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Début du programme.
.... avant incident
Interception de l'exception: Mauvais calcul!
Fin du programme.
```

Prof. Asmaa El Hannani

ENSA-El Jadida

321

Méthode propageant des exceptions

- Toute méthode susceptible de déclencher une exception (autre que la classe `RuntimeException` ou une dérivée) **qu'elle ne traite pas localement doit mentionner son type** dans une clause **throws** figurant dans son en-tête.

```
<modificateurs> <type> <identificateur> ( <liste param > )  
    throws < liste d'exceptions > { ..... }
```

Exemple :

```
protected static void meth ( int x, char c )  
    throws IOException, MonException { ..... }
```

Méthode propageant des exceptions

```
import java.io.*;  
class Action4{  
    private int x;  
    public void meth() throws IOException{  
        System.out.println(".... avant incident");  
        if(x==0)  
            throw new IOException("Problème d'E/S!");  
        System.out.println(".... après incident");  
    }  
}  
  
public class UseAction4 {  
    public static void main(String args[]){  
        Action4 obj = new Action4();  
        System.out.println("Début du programme.");  
        try{  
            obj.meth();  
        }catch(IOException e){  
            System.out.println("Interception de l'exception: " + e.getMessage());  
        }  
        System.out.println("Fin du programme.");  
    }  
}
```

Signaler l'exception susceptible d'être propagée

Interception de l'exception dans le bloc englobant

Redéfinition d'une méthode propageant des exceptions vérifiées

■ Principe de base :

La partie **throws < liste d'exceptions >** de la signature de la méthode qui redéfinit une méthode de la super-classe

- peut comporter moins de types d'exception.
- ne peut pas propager plus de types ou des types différents de ceux de la méthode de la super-classe.

Exception personnalisée

- Pour une exception personnalisée, le mode d'action est strictement identique, il vous faut seulement auparavant créer une nouvelle classe **héritant obligatoirement de la classe `Exception`** ou de n'importe laquelle de ses sous-classes.

Exception personnalisée: Exemple (1/3)

```
public class Equation {
    private int a;
    private int b;
    private int c;

    public Equation(int a, int b, int c) {
        this.a = a; this.b = b; this.c = c;
    }

    public double delta() {
        return b*b - 4*a*c;
    }

    public double solution() throws PasDeSolution {
        double discr = delta();
        if (discr<0) throw new PasDeSolution( this , "cette équation n'a pas de solution!!");
        return (b + Math.sqrt(discr))/(2*a);
    }

    @Override
    public String toString(){
        return a + "x^2 + " + b + "x + " + c;
    }
}
```

Prof. Asmaa El Hannani

ENSA-El Jadida

326

Exception personnalisée: Exemple (2/3)

```
public class PasDeSolution extends Exception {
    private Equation eq;

    public PasDeSolution(String s){
        super(s);
    }

    public PasDeSolution(Equation eq, String s){
        super(s);
        this.eq = eq;
    }

    public String getDetailedMessage(){
        return "L'équation: " + this.eq + ", L'erreur: " + getMessage();
    }

    // Autres méthodes au besoin ....
}
```

Prof. Asmaa El Hannani

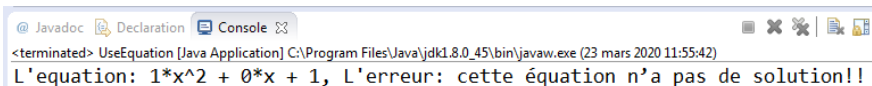
ENSA-El Jadida

327

Exception personnalisée: Exemple (3/3)

```
public class UseEquation {
    public static void main(String args[]){
        try {
            Equation eq = new Equation(1,0,1);    // x2 + 1 = 0
            double resultat = eq.solution();
            System.out.println("La solution: " + resultat );
        }
        catch(PasDeSolution p) {
            System.out.println(p.getDetailedMessage());
        }
    }
}
```

- Voici ci-dessous l'affichage obtenu sur la console lors de l'exécution :



The screenshot shows a Java IDE with a console window. The console output is as follows:

```
<terminated> UseEquation [Java Application] C:\Program Files\Java\jdk1.8.0_45\bin\javaw.exe (23 mars 2020 11:55:42)
L'equation: 1*x^2 + 0*x + 1, L'erreur: cette équation n'a pas de solution!!
```

Prof. Asmaa El Hannani

ENSA-El Jadida

328

Clause finally

- Java permet d'introduire, à la suite d'un bloc **try**, un bloc particulier **d'instructions qui seront toujours exécutées** :
 - soit après la fin "naturelle" du bloc **try**, si aucune exception n'a été déclenchée,
 - soit après le gestionnaire d'exception **catch** (à condition, bien sûr, que ce dernier n'ait pas provoqué d'arrêt de l'exécution).
- Ce bloc est introduit par le mot-clé **finally** et doit obligatoirement être **placé après le dernier gestionnaire (catch)**.

Prof. Asmaa El Hannani

ENSA-El Jadida

329

Clause finally

```
try {
    <code à protéger>
}
catch (Exception1 e) {
    <traitement de l'exception1>
}
catch (Exception2 e) {
    <traitement de l'exception2>
}
...
finally {
    <action toujours effectuée>
}
```

Redéclenchement d'une exception

- Dans un gestionnaire d'exception, il est possible de demander que, malgré son traitement, **l'exception soit retransmise à un niveau englobant**, comme si elle n'avait pas été traitée.
- Il suffit pour cela de la relancer en appelant à nouveau l'instruction **throw** :

```
try
{ .....}
catch (Excep e) // gestionnaire des exceptions de type Excep
{
    .....
    throw e ; // on relance l'exception e de type Excep
}
```


Exercice 9

- Y a-t-il un problème avec cette interception telle que codée ci-dessous ? Compilera-t-elle ?

```
try {  
    ...  
} catch (Exception e) {  
    ...  
} catch (ArithmeticException a) {  
    ...  
}
```

Exercice 10

- Quels résultats fournit ce programme ?

```
class Except extends Exception {}  
  
public class Finally {  
    public static void f(int n){  
        try{  
            if (n!=1)  
                throw new Except ();  
            System.out.println ("dans try - n = " + n)  
        }catch (Except e){  
            System.out.println ("catch dans f - n = " + n) ;  
            return ;  
        }  
        finally{  
            System.out.println ("dans finally - n = " + n) ;  
        }  
    }  
    public static void main (String args[]){  
        Finally.f(1);  
        Finally.f(2) ;  
    }  
}
```

Exercice 11

- A. Réaliser une classe **EntNat** permettant de manipuler des entiers naturels (positifs ou nuls).
- Pour l'instant, cette classe disposera simplement :
 - d'un constructeur à un argument de type *int* qui générera une exception de type **ErrConst** (type classe à définir) lorsque la valeur reçue ne conviendra pas,
 - d'une méthode `getter`.
 - Ecrire **ErrConst** de manière à disposer dans le gestionnaire d'exception du type `ErrConst` de la valeur fournie à tort au constructeur.
- B. Écrire un petit programme d'utilisation qui traite l'exception `ErrConst` en affichant un message et en interrompant l'exécution.