

Traitement des erreurs en Java

Les Exceptions

I. Définition et terminologie

Prévoir les erreurs d'utilisation

- Certains cas d'erreurs peuvent être prévus à l'avance par le programmeur.
 - erreurs d'entrée-sortie (I/O fichiers)
 - erreurs de saisie de données par l'utilisateur
- Le programmeur peut :
 - «Laisser planter» le programme à l'endroit où l'erreur est détectée
 - Manifester explicitement le problème à la couche supérieure
 - Tenter une correction

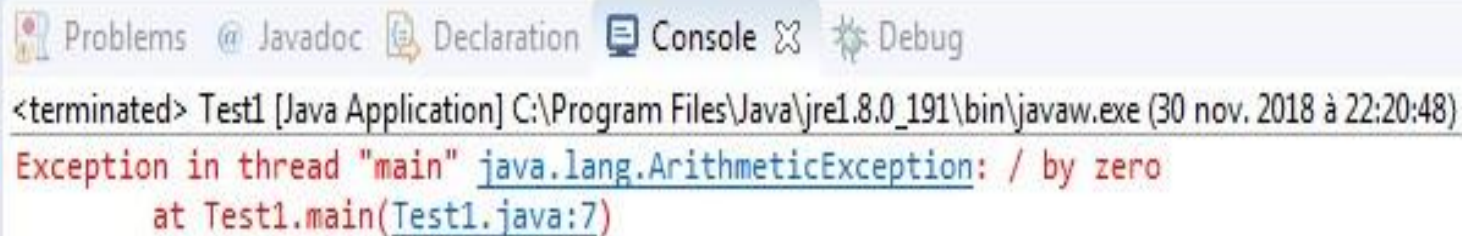
Définition

- Une exception est une **erreur** qui se produit lors de l'**exécution** d'un programme et qui en perturbe le déroulement normal.
- Cependant, vous pouvez gérer ces problèmes au sein de votre programme et prendre les mesures correctives qui s'imposent pour assurer la bonne marche de l'exécution (gestion des exceptions).

Exemple : Division par zéro

```
2 public class Test1 {  
3     public static void main (String[] args )  
4     {  
5         int a = 0;  
6         int b = 5;  
7         double c = b/a; Quitter le programme!  
8         System.out.println("a par b égale" + c);  
9     }  
10 }
```

Toutes les instructions qui suivent
ne seront pas exécutées



The screenshot shows the 'Console' tab of an IDE. The output text is as follows:

```
<terminated> Test1 [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (30 nov. 2018 à 22:20:48)  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at Test1.main(Test1.java:7)
```

Résumé de l'exemple

- La méthode main :
 - a déclaré et initialisé à la valeur 0,
 - b déclaré et initialisé à la valeur 5,
 - c déclaré et initialisé à la valeur b/a
- Dès que l'instruction $c=b/a$ est exécutée , elle provoque un incident:
 - ➡ Une exception de la classe **ArithmeticException** a été 'levée' et un objet de cette classe a été instancié par la JVM.
 - ➡ La JVM arrête le programme immédiatement à cet endroit puisqu'elle n'a pas trouvé de code d'interception (traitement) de cette exception qui a été automatiquement levée.

Notion d'exception

- En Java, les erreurs se produisent lors d'une exécution sous la forme d'exceptions.
- Une exception :
 - est un objet, instance d'une **classe d'exception**
 - provoque la sortie d'une méthode
 - correspond à un type d'erreur
 - contient des informations sur cette erreur
- Deux solutions alors :
 - laisser le programme se terminer avec une erreur,
 - essayer, malgré l'exception, de continuer l'exécution normale.
- Pour continuer l'exécution normale :
 - *Lever une exception consiste à signaler quelque chose d'exceptionnel.*
 - *Capter l'exception consiste à essayer de la traiter.*

II. Gestion des exceptions

Traiter des exceptions

- Une façon de traiter les exceptions consiste à les éviter d'office, tout simplement.

Exemple:

- une **logique conditionnelle classique** permet d'éviter l'exception

ArithmeticException

➔ Faire un test pour voir si la condition se posera avant de tenter l'opération risquée.

```
int divisor = 0;

if(divisor == 0){
    System.out.println("Ne peut pas être nul !");
}
else{
    System.out.println( 5 / divisor );
}
```

L'interception des exceptions : bloc try...catch

- Il n'est toujours pas possible d'empêcher toutes les exceptions en utilisant une logique conditionnelle car on ne peut pas prévoir toutes les erreurs possibles.
- Java possède une instruction de gestion des exceptions permettant **d'intercepter** des exceptions de la classe Exception :

bloc **try/catch**

- Si un **code est susceptible de générer une exception**, on peut l'écrire dans un bloc **try** spécial.
 - Associez des **gestionnaires d'exceptions** au bloc try en plaçant des blocs **catch** après le try.
 - Chaque bloc catch gère le type d'exception mentionné par son argument.
 - Le type d'argument **ExceptionType** déclare le type d'exception.

Bloc try ... catch

```
try{ ..... }
```

Si une erreur
se produit
ici.... **(lever
l'exception)**

```
catch (.....) {  
    .....  
    .....  
}
```

On tente de récupérer là.
Capter l'exception

Si le bloc `try` réussit, aucune exception n'est générée.

```
try {
```

```
//Code risqué susceptible de générer  
//une exception
```

```
}
```

```
catch(ExceptionType ex) {
```

```
//Code de gestion des exceptions
```

```
}
```

```
2
```

```
System.out.println("Nous avons réussi");
```

Le bloc `try` est exécuté en premier, suivi du code placé après le bloc `catch`.

Si le bloc `try` mène à un **échec**, une exception est générée.

```
try {
```

```
//Code risqué susceptible de générer  
//une exception
```

Le bloc `try` est exécuté, une exception est générée et le reste du bloc `try` n'est pas traité.

```
}
```

```
catch(ExceptionType ex) {
```

```
//Code de gestion des exceptions
```

Le bloc `catch` est exécuté, puis le reste du code.

```
}
```

```
System.out.println("Nous avons réussi");
```

Exemple : Traitement de l'exception

```
1
2 public class Test1 {
3     public static void main (String[] args )
4     {
5         int a = 0;
6         int b = 5;
7         try
8         {
9             double c = b/a;
10            System.out.println("a par b égale" + c);
11        }
12        catch(Exception E)
13        {System.out.println("Exception " + E.getMessage() + " est traitée");}
14        System.out.println("Au revoir");
15    }
16 }
```

1. Engendrer une exception
2. Levée d'ArithmeticException

3. Capture et traitement de l'exception

4. Poursuivre l'exécution normale du programme

Problems Javadoc Declaration Console Debug Coverage

<terminated> Test1 [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (30 nov. 2018 à 23:47:55)

Exception / by zero est traitée

Au revoir

Exemple : suite

- Dès qu'une **exception est levée** (instanciée), la JVM **stoppe immédiatement** l'exécution normale du programme à la **recherche d'un gestionnaire d'exception** susceptible d'intercepter (attraper) et traiter cette exception.
- Dans l'exemple précédent on a traité l'objet Exception (Générique)
- On peut spécifier le type d'exception susceptible d'être déclenchée (dans ce cas ArithmeticException)

```
2 public class Test1 {  
3     public static void main (String[] args )  
4     {  
5         int a = 0;  
6         int b = 5;  
7         try  
8         {  
9             double c = b/a;  
10            System.out.println("a par b égale" + c);  
11        }  
12        catch(ArithmeticException E)  
13        {System.out.println("Exception " + E.getMessage() + " est traitée");}  
14        System.out.println("Au revoir");  
15    }  
16 }
```

Exemples d'exception

- java.lang.ArrayIndexOutOfBoundsException

Tentative d'accès à un index de tableau inexistant

- java.lang.NullPointerException

Tentative d'utilisation d'une référence d'objet non instanciée

- java.io.IOException

Echec ou interruption des opérations d'E/S

- *Tentative de forçage de type illégale* : ClassCastException

- *Tentative de création d'un tableau de taille négative*

NegativeArraySizeException

Comprendre les exceptions courantes

– Exemple : `ArrayIndexOutOfBoundsException`

```
01  int[] intArray = new int[5];  
02  intArray[5] = 27;
```

– Trace de pile :

```
Exception in thread "main"  
    java.lang.ArrayIndexOutOfBoundsException: 5  
        at TestErrors.main(TestErrors.java:17)
```

NullPointerException

- Cette exception non vérifiée est générée si une application tente d'utiliser la valeur null alors qu'un objet est nécessaire.
- Exemples de circonstances :
 - Appel de la méthode d'instance d'un objet null
 - Accès au champ d'un objet null ou modification du champ

Appel de la
méthode `length`
sur un objet null

```
public static void main(String[] args) {  
  
    String name=null;  
    System.out.print("Longueur de la chaîne"+  
name.length());  
  
}
```

IOException

```
public static void main(String[] args) {  
    try {  
        File testFile = new File("//testFile.txt");  
        testFile.createNewFile();  
        System.out.println("testFile existe :"  
+ testFile.exists());  
    }  
    catch (IOException e) {  
        System.out.println(e);  
    }  
}
```

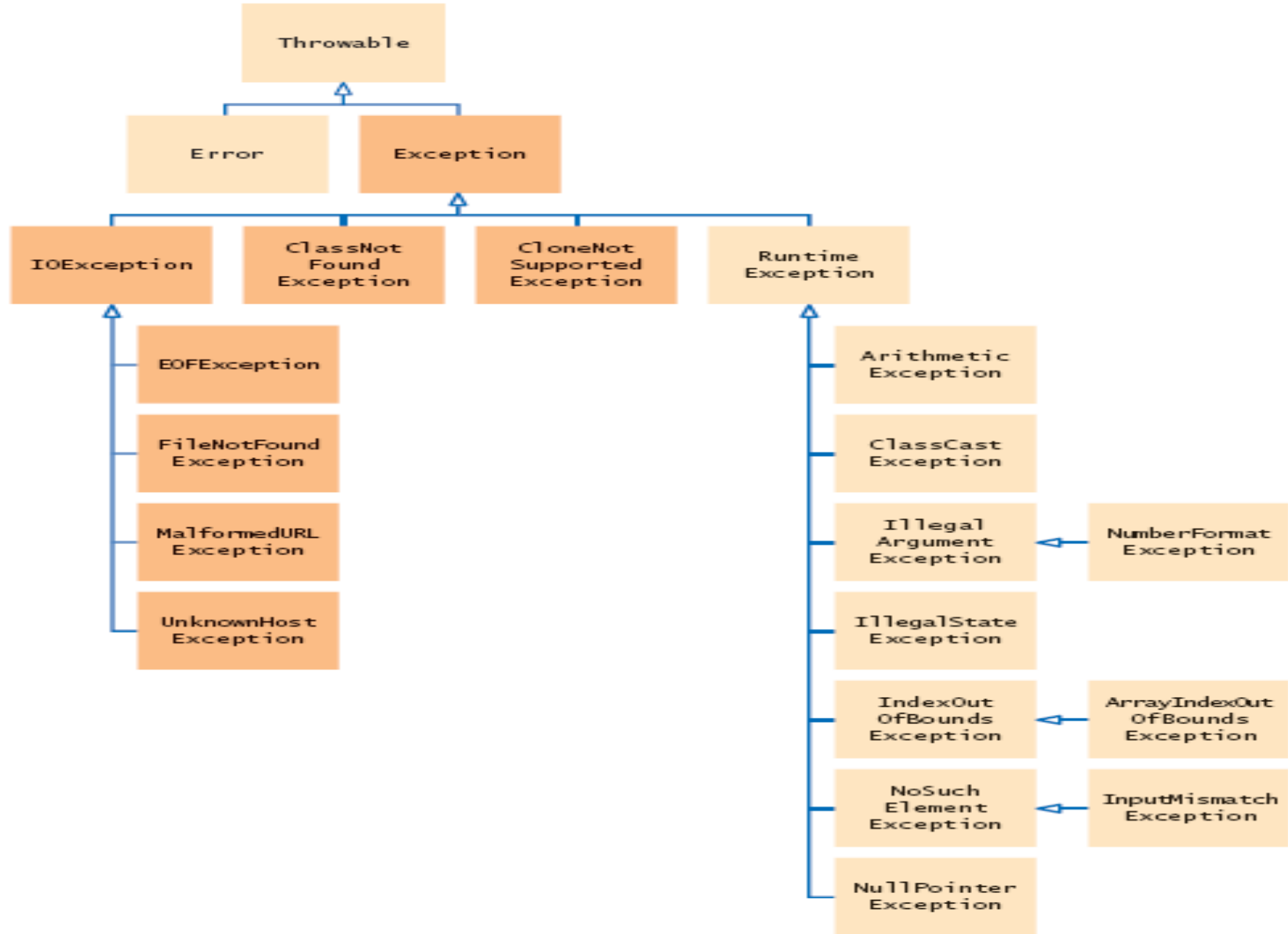


Figure 1 The Hierarchy of Exception Classes

Nature des exceptions

- En Java, les exceptions sont des objets ayant 3 caractéristiques:
 - Un type d'exception (défini par la classe de l'objet exception)
 - Une chaîne de caractères (option), (hérité de la classe *Throwable*).
 - Un « instantané » de la pile d'exécution au moment de la création.
- Les exceptions construites par l'utilisateur étendent la classe *Exception*
- *RuntimeException*, *Error* sont des exceptions et des erreurs prédéfinies et/ou gérées par Java

Quelques méthodes de la classe Exception

Méthode	Utilité
getMessage()	Retourne un message décrivant l'exception
printStackTrace()	Imprime le contenu de la pile dans le fichier d'erreur err
String toString()	Message donnant une description spécifique de l'exception

try / catch / finally

```
try
{
    ...
}
catch (<une-exception>)
{
    ...
}
catch (<une_autre_exception>)
{
    ...
}

...

finally
{
    ...
}
```

- ⇒ Autant de blocs **catch** que l'on veut
- ⇒ Bloc **finally** facultatif.

Traitement des exceptions

- Le bloc **try** est exécuté jusqu'à ce qu'il se termine avec succès ou bien qu'une exception soit levée.
- Dans ce dernier cas, les clauses **catch** sont examinées l'une après l'autre dans le but d'en trouver une qui traite cette classe d'exceptions (ou une superclasse).
- *Les clauses catch doivent donc traiter les exceptions de la plus spécifique à la plus générale.*
- Si une clause **catch** convenant à cette exception a été trouvée et le bloc exécuté, l'exécution du programme reprend son cours.

Bloc finally

- Un bloc **finally** permet au programmeur de définir un ensemble d'instructions qui est toujours exécuté, que l'exception soit levée ou non, capturée ou non.
- La seule instruction qui peut faire qu'un bloc **finally** ne soit pas exécuté est **System.exit()**.

Plusieurs Catch : Exemple 1

```
3 public class test2 {  
4  
5 public static void main(String[] args)  
6 {  
7     int T[] = {0,1};  
8     try  
9     {  
10        System.out.println(T[2]/0);  
11    }  
12    catch(ArithmeticException E) {System.out.println("Division par Zéro impossible!");}  
13    catch(IndexOutOfBoundsException E) {System.out.println("Taille dépassée!Élément inexistant");}  
14  
15 }  
16  
17 }
```

1. Levée de l'exception: IndexOutOfBoundsException

1. Capture de l'exception

Plusieurs Catch : Exemple 1

- La première exception déclenchée par le programme sera levée
- Dans ce cas `T[2]` déclenche **`IndexOutOfBoundsException`** qui sera recherchée séquentiellement dans la liste des catch.
- L'exception **`ArithmeticException`** ne sera pas déclenchée puisque le programme s'arrêtera au niveau de `T[2]`.

Exemple 2 : Interception ClassCastException

Interception d'une ClassCastException :

```
class Action2 {  
    public void meth(){  
        // une exception est levée ...  
    }  
}
```

ClassCastException



```
class UseAction2{  
    public static void main(String[] Args) {  
        Action1 Obj = new Action1();  
        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.");  
    }  
}
```

Vérification séquentielle de l'exception

Non!

Non!

Oui!

Example 3

```
public class HelloWorld{  
  
    public static void main(String []args){  
  
        int t[]={0,1};  
  
        try  
        {  
  
            System.out.println(t[2]/0);  
  
        }  
  
        catch(Exception e)  
        {  
            System.out.println(e);  
        }  
  
        catch (IndexOutOfBoundsException e)  
        {  
            System.out.println(e);  
        }  
    }  
}
```

```
HelloWorld.java:18: error: exception IndexOutOfBoundsException has already been caught  
    catch (IndexOutOfBoundsException e)  
    ^
```

1 error

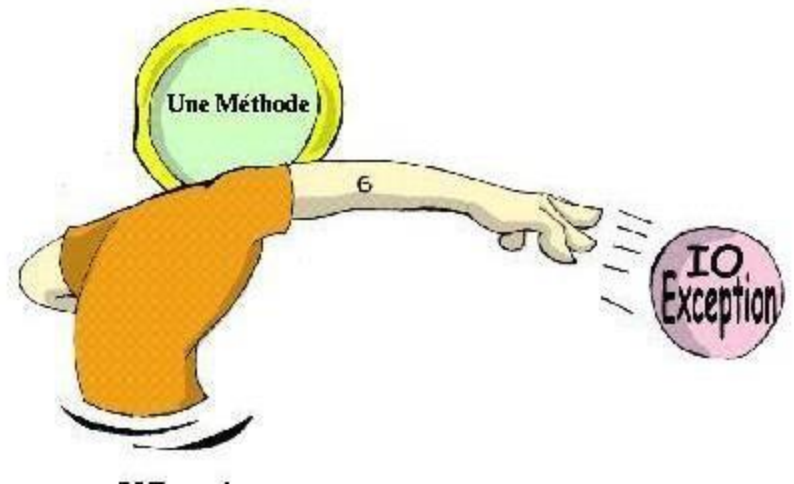
Exemple 3:correction -de la plus spécifique vers la plus générale

```
public class HelloWorld{  
  
    public static void main(String []args){  
  
        int t[]={0,1};  
  
        try  
        {  
  
            System.out.println(t[2]/0);  
  
        }  
  
        catch (IndexOutOfBoundsException e)  
        {  
            System.out.println(e);  
        }  
  
        catch(Exception e)  
        {  
            System.out.println(e);  
        }  
    }  
}
```

java.lang.ArrayIndexOutOfBoundsException: 2

Interception des exceptions : Résumé

- Le bloc **try** est exécuté jusqu'à ce qu'il se termine avec succès ou bien qu'une exception soit **levée**.
- Dans ce dernier cas, les clauses **catch** sont examinées l'une après l'autre dans le but d'en trouver une qui **traite** cette classe d'exceptions.
- Les clauses **catch** doivent donc traiter les exceptions de la plus spécifique à la plus générale.
- Si une clause **catch** convenant à cette exception a été trouvée et le bloc exécuté, l'exécution du programme reprend son cours.
- Le bloc **finally** permet au programmeur de définir un ensemble d'instructions qui est toujours exécuté, que l'exception soit levée ou non, capturée ou non.
- La seule instruction que peut faire qu'un bloc **finally** ne soit pas exécuté est `System.exit()`.



III. Levée manuelle des exceptions : Throw

Propagation des exceptions : Throws

Interception vs propagation

- Si une méthode peut émettre une exception, il faut :
 - 1) soit **intercepter** et traiter l'exception (try ... catch).
 - 2) soit **propager** l'exception (la méthode doit l'avoir déclarée)
au niveau supérieur pour être traitée;

1) Interception d'une exception dans une méthode

```
import java.io.*;

class Action4 {
    public void meth(){
        int x=0;
        System.out.println("    ..Avant incident");
        try{
            if (x==0)
                throw new IOException("Problème d'E/S !");
            catch(IOException E){
                System.out.println("Interception exception : "+E.getMessage());
            }

            System.out.println("    ...Après incident");
        }
    }
}

class UseAction4{
    public static void main(String[] Args) {
        Action4 Obj = new Action4();
        System.out.println("Début du programme.");
        Obj.meth();
        System.out.println("Fin du programme.");
    }
}
```

Interception de l'exception dans la méthode

Appel ordinaire de la méthode dans le bloc englobant

1) Interception d'une exception dans une méthode

Résultat de l'exécution :

Début du programme.

...Avant incident

Interception exception : Mauvais calcul !

Fin du programme.

2) Levée manuelle et propagation d'une exception prédéfinie

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

→ Signaler les exceptions susceptibles d'être déclenchées

↓ Levée manuel d'une exception prédéfinie Et instantiation deArithmeticException

```
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 exception : "+E.getMessage());  
        }  
        System.out.println("Fin du programme.");  
    }  
}
```

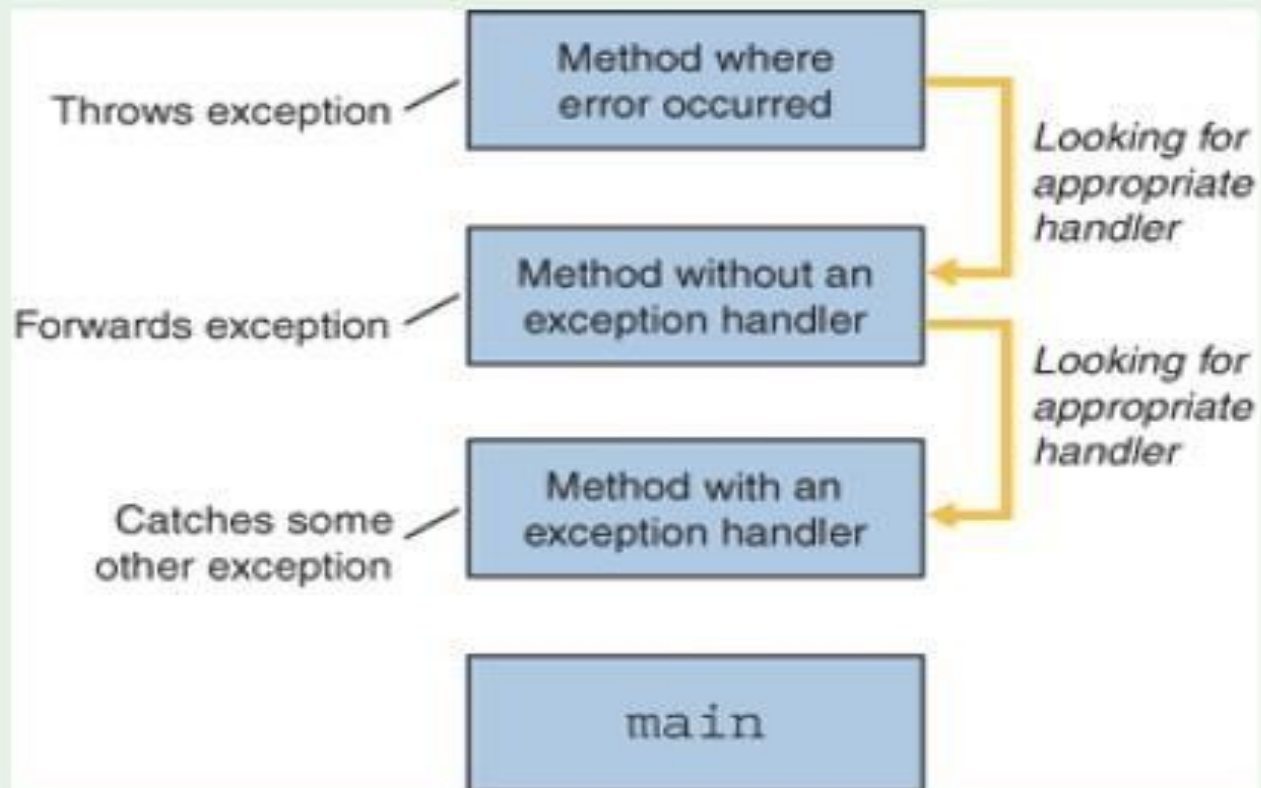
Interception et capture de l'exception prédéfinie

Throws

- Le mot-clé “**Throws**” s’utilise au niveau de la **signature** d’une méthode pour préciser que celle-ci est susceptible de lancer une exception
- Si une exception n’est pas attrapée dans le bloc où elle a été lancée elle est transmise au bloc de niveau supérieur (la méthode qui invoque cette dernière) (**récurivement**)
- Si une exception n’est jamais attrapée
 1. propagation jusqu’à la méthode main()
 2. affichage des messages d’erreurs et la pile d’appels
 3. arrêt de l’exécution du programme

Gestion d'une exception

Recherche d'un exception handler dans la call stack



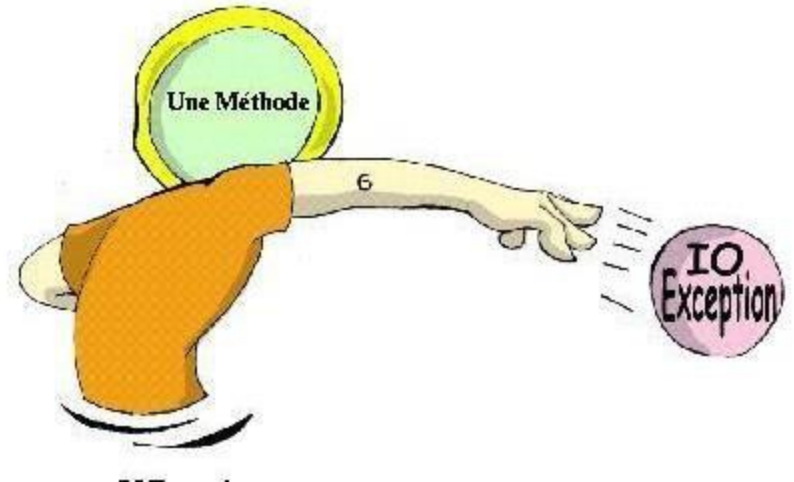
throws

- Pour "laisser remonter" à la méthode appelante une exception qu'il ne veut pas traiter, le programmeur rajoute le mot réservé **throws** à la déclaration de la méthode dans laquelle l'exception est susceptible de se manifester.

```
public void uneMethode() throws IOException
{
    // ne traite pas l'exception IOException
    // mais est susceptible de la générer
}
```

throws

- Les programmeurs qui utilisent une méthode connaissent ainsi les exceptions qu'elle peut lever.
- La classe de l'exception indiquée peut tout à fait être une super-classe de l'exception effectivement générée.
- Une même méthode peut tout à fait "**laisser remonter**" **plusieurs types d'exceptions (séparés par des ,)**.
- Une méthode doit traiter ou "laisser remonter" toutes les exceptions qui peuvent être générées dans les méthodes qu'elle appelle.



IV. Exceptions personnalisées

Levée d'exceptions personnalisées ou exceptions utilisateurs

- Le programmeur peut définir ses propres codes d'erreurs (exceptions personnalisées)
- Le mode d'action *est le même que les exceptions prédéfinies,*
- Il faut seulement créer une nouvelle classe *héritant obligatoirement de la classe **Exception*** ou de n'importe laquelle de ses sous-classes.

```
throw new MonException("Mon exception s'est produite!!!");
```

Syntax: Example

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

Les exceptions personnalisées : Exemple1

- Classe Equation permet de résoudre les équations de deuxième degré
 - $\Delta > 0$ ➡ $\frac{-b + \sqrt{\Delta}}{2a}$
 - $\Delta < 0$ ➡ Pas de solution : On génère une Exception personnalisée

Les exceptions personnalisées : Exemple1

```
public class Equation
{
    private double a,b,c ;
    Equation (double ap , double bp , double cp )
    {
        a = ap;
        b = bp;
        c = cp;
    }
    public double delta()
    { return ((b*b)-(4*a*c)); }
    public double solution() throws PasDeSolution
    {
        double discr = delta();
        if ( discr <0) throw new PasDeSolution () ;
        return ( -b + Math.sqrt (discr ))/(2*a);
    }
}
```

3. Instanciation de la classe PasDeSolution
Et Propagation de l'exception à la méthode main



```
class PasDeSolution extends Exception {Définition d'une exception personnalisée
    public String toString ()
    {return "L'équation n'a pas de solution";}
}
```

Les exceptions personnalisées : Exemple 1

```
public class test3 {  
    public static void main (String[] args )  
    {  
        // Méthode appelante  
        try  
        {  
            Equation eq = new Equation(1,0,1);    1.Instanciation de la classe Equation  
            double resultat = eq.solution ();      2.Appeler la méthode solution de la classe  
        }  
        catch( PasDeSolution p) {System.out.println (p.toString ()); }  
    }  
    4.Capture de l'exception PasDeSolution et affichage du message  
}
```

Résultats de l'exécution :

L'équation n'a pas de solution

Les exceptions personnalisées : Exemple 2

```
class ArithmeticExceptionPerso extends ArithmeticException{
    ArithmeticExceptionPerso(String s){
        super(s);
    }
}

class Action3 {
    public void meth() Throws ArithmeticExceptionPerso {
        int x=0;
        System.out.println(" ...Avant incident");
        if (x==0)
            throw new ArithmeticExceptionPerso("Mauvais calcul !");
        System.out.println(" ...Après incident");
    }
}

class UseAction3{
    public static void main(String[] Args) {
        Action3 Obj = new Action3();
        System.out.println("Début du programme.");
        try{
            Obj.meth();
        }
        catch(ArithmeticExceptionPerso E){
            System.out.println("Interception exception : "+E.getMessage());
        }
        System.out.println("Fin du programme.");
    }
}
```

1. Propagation de l'exception, à la couche supérieure : main()

2. Capture l'exception

Les exceptions personnalisées : Exemple 2

- Dans le programme précédent, on a créé une classe d'exception `ArithmeticExceptionPerso` héritant de la classe des `ArithmeticException`

Résultats de l'exécution :

Début du programme.

...Avant incident

Interception exception : Mauvais calcul!

Conclusion

- Grâce aux exceptions, Java possède un mécanisme sophistiqué de gestion des erreurs permettant d'écrire du code robuste.
- Le programme peut déclencher des exceptions au moment opportun.
- Le programme peut capturer et traiter les exceptions grâce au bloc d'instructions **try ... catch ... finally**
- Le programmeur peut définir ses propres classes d'exceptions

Conclusion

- Du code Java valide doit nécessairement inclure la gestion des exceptions et respecter ainsi le bloc **try ... catch**
- Tout code susceptible de renvoyer une exception doit être contenu soit :
 - Dans une instruction **try ... catch** qui définit ainsi le handler (gestionnaire) approprié.
 - Dans une méthode spécifiant explicitement qu'elle peut renvoyer une exception : il faut inclure le mot clé **throws**, suivi du type d'exception renvoyée, dans la signature de la méthode.

```
Exemple : public void setAge(int age)  
          throw NegativeAgeException
```