

EXCEPTIONS

Java Standard Edition



Objectif de ce cours



A la fin de cours vous serez capable de :

- **Utiliser les** exceptions et sécuriser votre code
- **Créer** et lancer vos propres exceptions

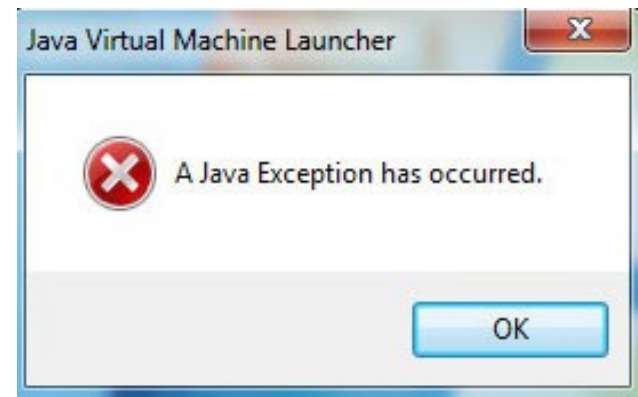
Plan du cours



- Introduction
- En dérivant de la classe Throwable
- Classe error
- Classe exception : exception surveillée / non surveillée
- Blocs try/catch/finally
- Throw/throws keywords
- Stack Trace.
- Créer vos propres exceptions

Exceptions

INTRODUCTION



Qu'est ce qu'une exception ?

Une exception est un évènement (exceptional event) qui :

- 1) interrompt l'exécution normale des instructions d'un programme quand il rencontre une erreur
- 2) code la réaction adéquate à ces erreurs

nature des erreurs

- division par zéro,
- conversion d'un type vers un autre,
- accès à indice de tableau se trouvant en dehors des bornes,
- ouverture d'un fichier qui n'existe pas,
- la mémoire qui manque lors de la création d'un objet.

Pourquoi gérer les exceptions

- séparer le code applicatif, du code de traitements des erreurs.
 - la méthode où l'exception est levée
 - elle est transmise à une classe/méthode spécialisée
- Permet de poursuivre l'exécution du code
- notifie la nature de l'erreur
- gère le traitement de l'erreur

Permet poursuite exécution du code

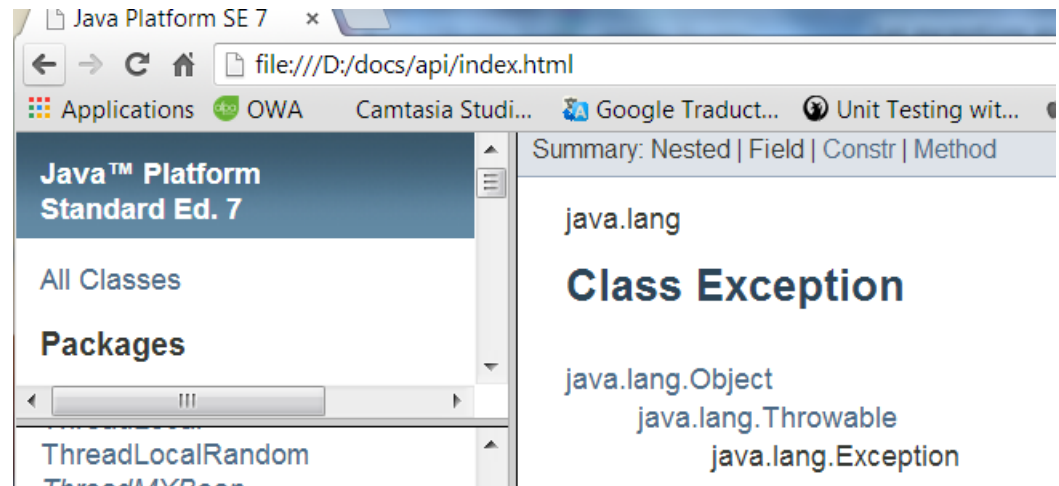
```
int numerateur = 10 , resultat = 0 ;  
resultat = numerateur/0 ;  
System.out.println("Le programme ne se poursuit pas" );
```

```
try {  
    resultat = numerateur/0 ;  
}  
catch (Exception e )  
{  
    System.out.println("Division Par Zéro");  
}
```

```
System.out.println("Le programme se poursuit" );
```


Qui génère les exceptions

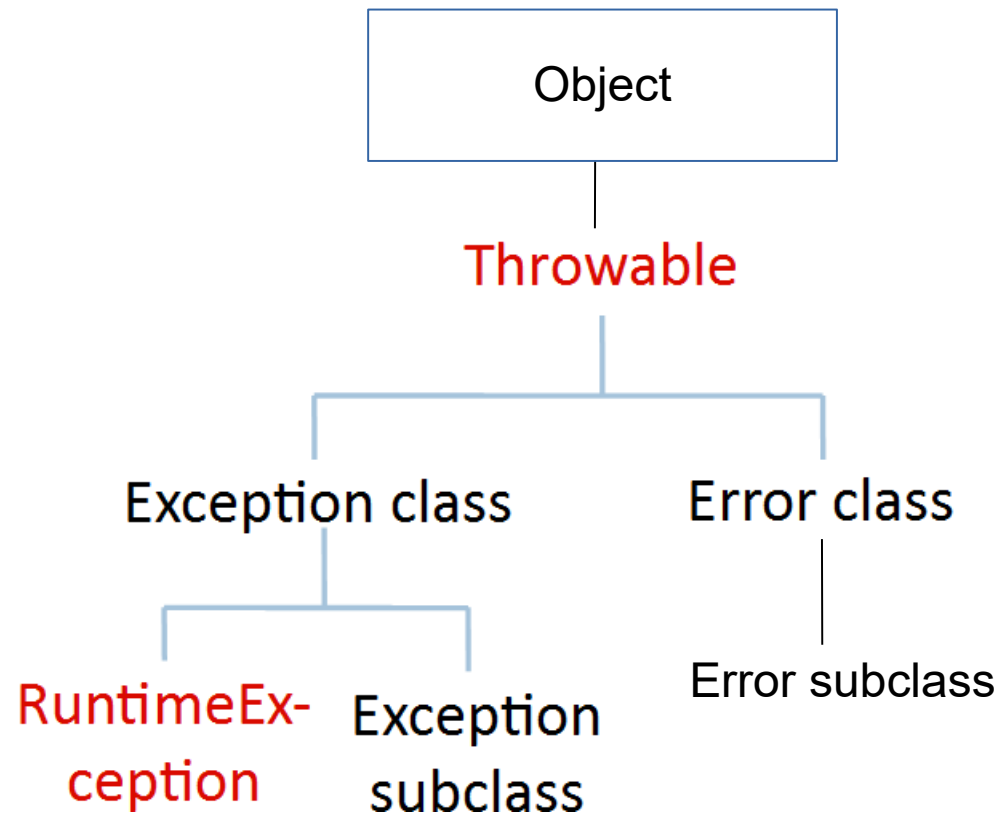
- Java génère des exceptions pour notifier au développeur les erreurs qu'il aurait oublié de traiter. Il existe dans Java des centaines de classes d'exception, chacune dédiée à un cas d'erreur particulier.



- Mais le développeur peut aussi créer ses propres classes d'exception...

Hiérarchie de classe

- Exceptions sont des Objets
- Tous dérivé de **Throwable**



Hiérarchie des exceptions

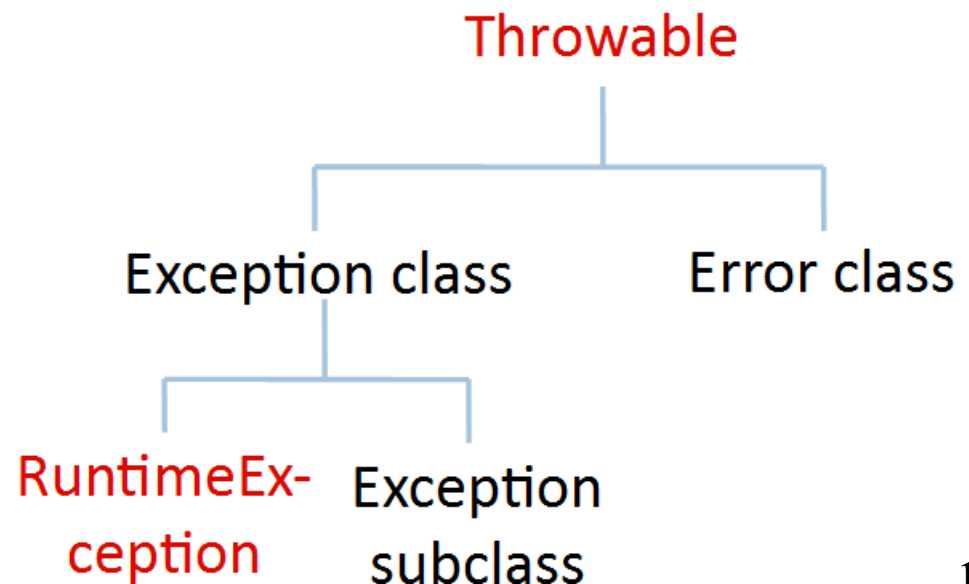
Throwable : Classe de base de toutes les exceptions.

Error class : erreur grave (en particulier, exceptions asynchrones) qu'il n'est pas raisonnable de chercher à récupérer.

Exception class : Exceptions méritant d'être détectées et traitées.

RuntimeException : Exceptions pouvant survenir durant le fonctionnement normal de la machine Java (indice de tableau hors des bornes, acces a un membre d'une reference null, division par zéro.....)

Exception subclass : Vos propres classes exceptions viennent ici



Exceptions

En dérivant de la classe **Throwable**

la classe **Error**

- représente des **erreurs** graves conduisant à l'arrêt du programme
- **Les erreurs** ne sont pas surveillées au moment de la compilation et n'ont pas la vocation à (mais peuvent) être capturées (catch) et gérées.

classe **Error**

<i>AssertionError</i>	Levée lorsqu'une assertion a échoué.
<i>ExceptionInInitializerError</i>	Levée lorsqu'une erreur dans un initialiseur static survient.
<i>VirtualMachineError</i>	Levée pour indiquer une erreur de la JVM.
<i>OutOfMemoryError</i>	Levée pour indiquer un manque de mémoire lors de l'allocation d'un objet.
<i>NoClassDefFoundError</i>	Levée lorsque la JVM ne peut trouver la définition d'une classe.
<i>StackOverflowError</i>	Levée lorsqu'un dépassement de la pile est atteint.

classe **Exception**

- représente les erreurs gérables
(les instructions présentes après l'erreur sont exécutées)
par du code Java.
- de 2 types:
 - les exceptions surveillées (check exceptions)
 - les exceptions non surveillées / runtime error

exception surveillée

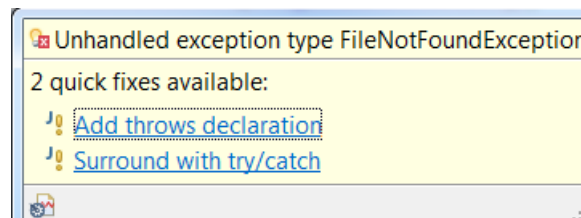
<i>ClassNotFoundException</i>	Levée lorsque une classe ne peut pas être chargée car sa définition ne peut être trouvée.
<i>IOException</i>	Levée lorsqu'une opération d'entrée/sortie s'interrompt ou échoue. Deux sous-classes communes d' <i>IOException</i> : <i>EOFException</i> et <i>FileNotFoundException</i> .
<i>FileNotFoundException</i>	Levée lorsqu'une ouverture se fait sur un fichier non trouvé.
<i>SQLException</i>	Levée pour une erreur liée à la base de données.
<i>InterruptedException</i>	Levée lorsqu'un thread est interrompu.
<i>NoSuchMethodException</i>	Levée lorsqu'une méthode appelée n'est pas trouvée.
<i>CloneNotSupportedException</i>	Levée lorsque la méthode <i>clone()</i> est appelée sur un objet non clonable.

exception surveillée

Obligation de coder leur gestion:

- soit encadrer le code avec try / catch
- soit ajouter l'instruction **throws** dans la signature de la méthode

sinon le programme ne compilera pas



exception surveillée : obligation gestion

Exemple: ouverture de fichier

```
public class mainCoursExceptSurveille {  
    public static void main(String[] args) {  
        FileReader lecteur = new FileReader("java.txt");  
    }  
}
```

Exception in thread "main" java.lang.Error: Unresolved compilation problems:

Traitement local de l'exception →

```
Try { FileReader lecteur = new FileReader("java.txt"); }  
catch (Exception e) { e.printStackTrace(); }
```

Propagation exception(voire section du cours throw/throws)→

```
public static void main(String[] args) throws FileNotFoundException{  
    FileReader lecteur = new FileReader("java.txt");  
}
```

java.io.FileNotFoundException: java.txt (Le fichier spécifié est introuvable)

exception non surveillée

pas obligatoire de coder un traitement spécifique

le code est compilable, l'erreur apparaîtra à l'exécution du code

→ de type runtime exception

exemple : division par zéro

```
System.out.println("le resultat de la division est:" + 10/0);
```

Lorsque l'erreur survient la J.V.M par **défaut** :

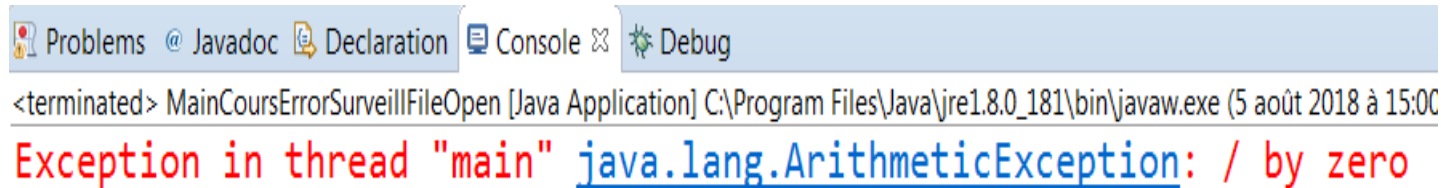
- crée une exception caractéristique de l'événement

- le thread est stoppé

- le comportement affiche le diagnostic de l'erreur



exceptions non surveillée : diagnostique s'affiche



The screenshot shows an IDE interface with tabs for Problems, Javadoc, Declaration, Console, and Debug. The Console tab is active, displaying the following text:
<terminated> MainCoursErrorSurveillFileOpen [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (5 août 2018 à 15:00
Exception in thread "main" java.lang.ArithmeticException: / by zero

L'erreur a été traitée. Java nous renseigne qu'une exception de classe :

java.lang.ArithmeticException
a été émise et déclenchée.

la trace de la pile d'exécution, au moment de l'exception indique que seule la méthode main est impliquée.

at runtimeException.RuntimeException.main(RuntimeException.java:12)

exemple erreurs non surveillées

<i>ArithmeticException</i>	Levée lorsqu'une erreur arithmétique survient.
<i>ArrayIndexOutOfBoundsException</i>	Levée lorsqu'un indice de tableau hors-plage est utilisé.
<i>ClassCastException</i>	Levée lors d'un casting d'un objet vers une sous-classe dont il n'est pas une instance.
<i>IllegalArgumentException</i>	Levée pour indiquer qu'un argument invalide a été transmis à une méthode.
<i>IndexOutOfBoundsException</i>	Levée lorsqu'un indice hors-plage est utilisé.
<i>NullPointerException</i>	Levée lorsque le code référence un objet <i>null</i> alors qu'un non <i>null</i> est attendu.
<i>NumberFormatException</i>	Levée pour indiquer qu'une conversion d'une chaîne vers un type numérique a échoué.

Exceptions

LES TRY/CATCH/FINALLY BLOCS

pour gérer les exceptions

FINALITE DES BLOCS

- Le bloc **try**:

encapsule le code à risque: `int a = 2/0;`

jamais seul: suivi d'un block catch ou/et finally

- Le bloc **catch**:

gère l'exception

contient le code à exécuter à la place d'un crash ☹️

les blocs catch sont en nombre quelconque

- Le bloc **finally** :

optionnel, si présent catch devient optionnel

s'exécute si l'exception est lancée ou pas

SYNTAXE

try

{ ... instructions dangereuses ... }

catch (exception e1)

{ ... code traitement e1 ... }

catch (exception e2)

{ ... code traitement e2 ... }

...

finally

{ s'exécute à tous les coups }

Exemple division par 0

```
int numerateur = 10 , resultat = 0 ;

try {
    resultat = numerateur/0 ;
    println("Suite du bloc try");//Instruction non exécutée
}
catch (Exception e ){//java.lang.Exception
    //La classe exception fournit la méthode getMessage() informant sur la nature de l'erreur
    println("Message de l'exception"+ e.getMessage());
    println("Message Personnalisé Division Par Zéro");
}
finally{
    System.out.println("action faite systématiquement");
}

println("Le programme se poursuit" );
```

```
Message de l'exception:/ by zero
Message Personnalisé Division Par Zéro
action faite systématiquement
Le programme se poursuit|
```

Exemple division par 0

Sur l'appel d'une méthode exemple la méthode Division()

```
public class coursTryCatchMethodCall {
    public static void main(String[] args)
    {
        int a = 15, b = ? ;
        try {
            Division(a,b);
        }
        catch(ArithmeticException Ex) { // b = 0
            System.out.print("Error : " + Ex.getMessage());
        }
    }
    public static void Division
    {
        System.out.print("resultat: " + a / b); //b = 1
    }
}
```

Error : / by zero

resultat: 15

Elargir le type d'erreur à gérer

Si, dans le bloc try, plusieurs catégories d'exceptions peuvent potentiellement être levées, alors, on peut placer autant de blocs catch, que de catégorie d'exception :

```
/* exemple d'erreur
   -un dénominateur différent de 0
   -un type non numérique */

System.out.println("Veuillez saisir un nombre :");
Scanner sc = new Scanner(System.in);
try {
    int denominateur = sc.nextInt();
    System.out.println("resultat: "+12/denominateur);
}
catch (ArithmeticException e){ //java.lang.ArithmeticException
    System.out.println("pas de divison par zero");
}
catch (InputMismatchException e){//java.util.InputMismatchException
    System.out.println("la val saisie n'est pas un nombre ");
}
```

Exceptions

throw/throws

Centralize handling and throw exception



Différencier les mots clé throw/throws

Dans un programme Java, toute exception jetée(throw) explicitement doit être soit propagée vers la méthode appelante, soit traitée localement (bloc try catch)

- **throw** se place
 - dans la méthode
 - gère l'exception localement (capture pas de propagation)
- **throws** se place
 - au niveau de l'en tête (la signature) de la méthode
 - propage l'exception

```
public void methodeOpenFile() throws IOException {  
    throw new IOException("File missing");  
}
```



la clause **throw** (se place dans la méthode)

Dans le code déclenchement d'exceptions en utilisant le mot clé **throw** suivi d'une instance issue de l'arborescence des exceptions ici de type exception (new Exception..)

```
System.out.println("nombre ?");
Scanner sc = new Scanner(System.in);
int v = sc.nextInt();
try {
    if (v < 0) throw new Exception("negative");
    if (v == 0) throw new Exception("nulle");
    if (v > 0) throw new Exception("positive");
}
catch (Exception e) {
    if(e.getMessage()=="negative")
        System.out.println("val negative");
    if(e.getMessage()=="nulle")
        System.out.println("val nulle");
    if(e.getMessage()=="positive")
        System.out.println("val positive");
}
```

la clause **throws** : propager une exception

- Leur entête précise la clause **throws** avec la/les exceptions que la méthode décide de ne pas traiter
- Indique les méthodes pouvant lever une exception (obligatoire pour les exceptions surveillées)
- la méthode décide de pas gérer localement l'exception
 - elle ne fournit pas le bloc try catch
 - l'exception est propagée à la méthode appelante pour y être traitée



la clause **throws** : propager une exception

Exception surveille -> obligation de traiter ou propager l'exception

```
public static void main(String[] args) {  
    methode1();  
}
```

```
public static void methode1() {  
    try {  
        methode2();  
    } catch FileNotFoundException fnfe) {  
        System.err.println(fnfe.getMessage());  
    }  
}
```

java.txt (Le fichier spécifié est introuvable)

```
public static void methode2() throws FileNotFoundException {  
    methode3();  
}
```

```
public static void methode3() throws FileNotFoundException {  
    FileReader lecteur = new FileReader("java.txt");  
}
```




Propagation de l'exception / clause throws

l'exception n'est pas traitée dans la méthode où elle se produit, mais dans une des méthode appelante (ici main)

```
public static void main(String[] args) {
    int a = 2 , b = 0;
    try {
        methode1(a,b);
    } catch (ArithmeticException e) {
        System.out.println("division par 0");
    }
}

static void methode1(int a, int b) throws ArithmeticException
{
    methode2(a,b);
}

static void methode2(int a, int b){

    System.out.println(a/b);

}
```

Console

division par 0

A noter ArithmeticException n'est pas une exception surveillée
le code sans traitement de l'exception reste compilable
La présence du mot clé throws n'est pas obligatoire

Exceptions

Read an exception stacktrace

We'll understand the cause of an exception.



Presentation

Une exception peut être levée non pas par une méthode appelée directement par votre code, mais par une autre méthode plus loin dans la pile des appels (stack trace).

Dans ce cas, Java déroule la pile à la recherche d'une méthode avec un bloc catch pour le type d'exception concerné

- La classe Throwable conserve la pile des appels
- Visible dans la console

avec la méthode: **void printStackTrace()**

Exemple

```
4 public static void main(String[] args) {  
5     int a = 2 ,b=0;  
6     try {  
7         methode1(a,b);  
8     } catch (ArithmeticException e) {  
9         e.printStackTrace();  
10    }  
11 }  
12 static void methode1(int a, int b){  
13     methode2(a,b);  
14 }  
15 static void methode2(int a, int b){  
16     System.out.println(a/b);  
17 }  
18 }
```

Console Problems Javadoc Declaration Debug Expressions

<terminated> mainPropagatExcept [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (9 janv. 2019 à 14:47:52)

[java.lang.ArithmeticException: / by zero](#)

- at coursPropagaExcept.mainPropagatExcept.methode2([mainPropagatExcept.java:16](#))
- at coursPropagaExcept.mainPropagatExcept.methode1([mainPropagatExcept.java:13](#))
- at coursPropagaExcept.mainPropagatExcept.main([mainPropagatExcept.java:7](#))

Exceptions

Définir ses propres classes d'exception



Exemple saisir un entier positif

```
public class mainCoursCreateYourOwnExcept {
    public static void main(String[] args){
        try {
            methodeGereNegNumber(-2);
        } catch (NegativeNumberException e) {
            System.out.println(e.getMessage());
        }
    }

    public static void methodeGereNegNumber(int a) throws NegNumberExcept
    {
        if (a < 0) throw new NegativeNumberException(a);
    }
}

class NegNumberException extends Exception {

    public NegativeNumberException(int num) {
        super("The number "+num+" is negative");
    }
}
```

A la construction d'un objet

Dans notre classe Point nous lançons une **exception** pour traiter l'invalidité des arguments fournis pour construire un point:

```
class Point
{
    static final int XMAX = 1024, YMAX = 768;
    int x, y;
    Point(int a, int b) throws Exception
    {
        if (a < 0 || a >= XMAX || b < 0 || b >= YMAX){
            throw new Exception();//le point n'est pas instancié
        }
        x = a; y = b;
    }
    //constructeur par défaut.....
}
```

Dans le main nous surveillons la construction du point

```
try { Point Pt0 = new Point(2345,0);}
catch (Exception e1) { println("instance hors limites"); }
finally{ if(Pt0 == null) Pt0 = new Point();}
```



Résumé

Une exception est un événement lié à une **situation anormale** qui modifie ou interrompt l'exécution d'un programme.

Java fournit un gestionnaire d'exceptions pour remédier à une telle situation.

La gestion des exceptions doit être découplée du déroulement normal d'un programme **Java**.

Les exceptions sont gérées en Java par un bloc *try*, *catch*, *finally*.

Le bloc *try* contient le code susceptible de lever des exceptions.

The try statement should contain at least one catch block or a finally block and may have multiple catch blocks.

Le(s) bloc (s) *catch* contient du code pour gérer les exceptions capturées

Le bloc *finally*

un bloc de code qui est obligatoirement exécuté, endroit idéal utilisé pour le nettoyage et la libération des ressources (exemple la fermeture de fichiers)

Une exception peut être levée non pas par une méthode appelée directement par votre code, mais par une autre méthode plus loin dans la pile des appels.

Dans ce cas, Java déroule la pile à la recherche d'une méthode avec un bloc catch pour le type d'exception concerné.



Quizz

- Fill the blanks:

Exceptions can have **2** kinds.

In order to manage an exception, we HAVE TO use the **try** keyword.

Then, we must add at least one of the following keywords : **catch or finally**

You can use several **catch** . blocks in one exception handling.

The **finally** block ensure you that its instructions will be executed whatever happens.



Quizz

- Can you replace the class name here?

