

POO : Séance 9

Les Exceptions en Java

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 des données par l'utilisateur
- Le programmeur peut :
 - «Laisser planter» le programme à l'endroit où l'erreur est détectée
 - Tenter une correction

Les Exceptions : Définition

- Une exception est une erreur qui se produit lors de l'exécution d'un programme et qui perturbe son déroulement normal.
- Cependant, on peut gérer ces problèmes au sein du programme et prendre les mesures correctives 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

Problems @ Javadoc Declaration Console  Debug

<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 la variable a,
 - a déclaré et initialisé la variable b,
 - a déclaré la variable c,
 - a tenté de diviser a par b.
- 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 de l'exécution sous la forme d'exceptions.
- Une exception :
 - Est un objet, instance de la classe **Exception**,
 - Provoque la sortie d'une méthode
 - Correspond à un type d'erreur : arithmétique, E/S, etc.
 - Contient des informations sur cette erreur
- Deux solutions possibles :
 - 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**.

Traiter les exceptions

- Une façon de traiter les exceptions consiste à les éviter tout simplement.
- **Exemple:**
 - Une logique conditionnelle permet d'éviter l'exception ***ArithmeticException***.
 - Faire un test pour voir si la condition se posera avant de tenter l'opération risquée.

```
1
2 public class Test1 {
3     public static void main (String[] args )
4     {
5         int a = 0;
6         int b = 5;
7         if(a==0)
8             System.out.println("Division par zéro impossible!");
9         else
10            {
11                double c = b/a;
12                System.out.println("a par b égale" + c);
13            }
14    }
15 }
```

L'interception des exceptions

Bloc try...catch

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 : Syntaxe

Try

{

.....

}


Catch

{

.....

.....

}



Si une erreur se
produit ici
(Lever l'exception)



On tente de la récupérer ici
(capture d'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
```

```
}
```

```
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
```

```
}
```

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

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

```
}
```

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

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

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

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.Engendre 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 }
```

Traitement généralisée des exceptions

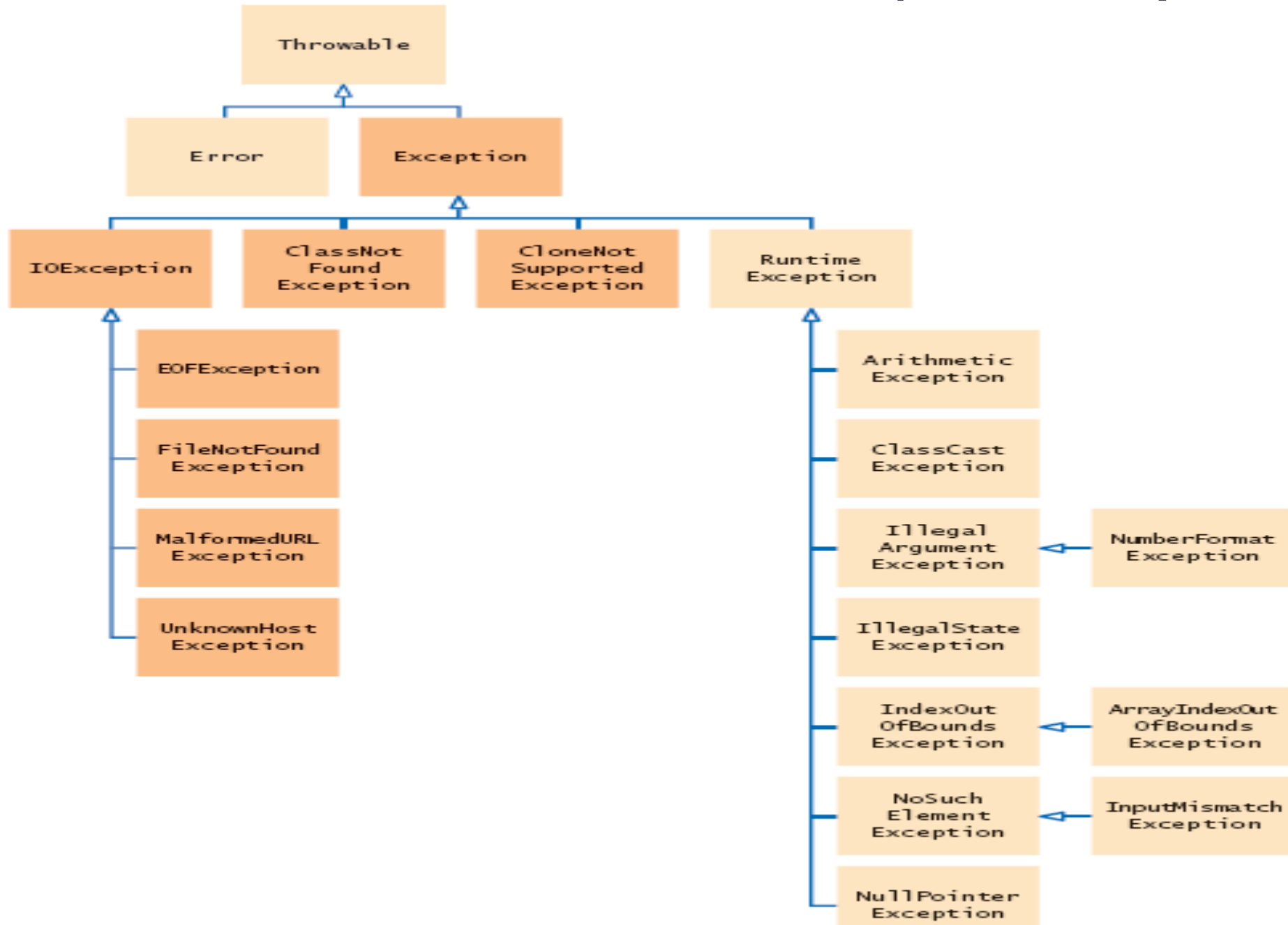
- Il est possible de traiter toutes les exceptions susceptibles d'être déclenchées par le programme de la même façon dans une seule clause catch

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

**Cast entre FileNotFoundException
et Exception**

- FileNotFoundException** sera déclenché
- FileNotFoundException** hérite de **IOException** qui hérite de la classe **Exception**

Hiérarchie des classes d'exceptions en JAVA



Les objets exceptions

- La classe **Exception** hérite de La classe **Throwable**.
- 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, héritée de la classe *Throwable*:
 - Ce champ est utilisé pour stocker le message décrivant l'exception,
 - Ce message peut être récupéré par la méthode `getMessage()`
 - Un «instantané» de la pile d'exécution au moment de la création.
- *RuntimeException*, *Error* sont des exceptions et des erreurs prédéfinies gérées par Java.
- Les exceptions construites par l'utilisateur étendent la classe *Exception*.

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

Exemples d'exception prédéfinies

- **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

- **ClassCastException**

Tentative de forçage de type illégale

- **NegativeArraySizeException**

Tentative de création d'un tableau de taille négative

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());  
  
}
```

Identifier une exception `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);  
    }  
}
```

Try...Catch...finally

try

{

...

}

catch (<TypeException E1>)

{

...

}

catch (<TypeExceptionE2>)

{

...

}

finally

{

...

}

⇒ Autant de blocs **catch** que l'on veut.

⇒ Bloc **finally** facultatif.

⇒ **TypeException** sont des classes d'exceptions obligatoirement toutes **distinctes**.

Interception de plusieurs exceptions

- Dans un gestionnaire try...catch, il est possible d'intercepter et de traiter plusieurs types d'exceptions différentes.
- Dès qu'une exception intervient dans le bloc try, la JVM vérifie séquentiellement toutes les clauses catch de la première jusqu'à la nième
- Si l'exception actuellement levée est d'un des types présents dans la liste des clauses catch, le traitement associé est effectué, la vérification des clauses catch est abandonnée et le programme poursuit son exécution après le gestionnaire.

Plusieurs Catch : Exemple I

```
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

2. Capture de l'exception

Plusieurs Catch : Exemple I

- 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]`.

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()`.

Levée manuelle des exceptions :Throw

Propagation des exceptions :Throws

Les méthodes déclenchant des exceptions

- 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 traité;

Levée manuel d'une exception : Throw

- Le programmeur peut lever des exceptions **prédéfinies** ou ses propres exceptions (**personnalisées**) à l'aide du mot réservé throw.
- **Throw** prend en paramètre un objet instance de Throwable ou d'une de ses sous-classes (Exception).

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

- Les objets exception sont souvent instanciés dans l'instruction même qui assure leur lancement avec New.
- Une méthode susceptible de lever des exceptions est identifiée par le mot-clé **throws** suivi du type de l'exception exemple:

```
public void ouvrirFichier(String name) throws  
    MonException  
    {  
        if (name==null) throw new MonException();  
        else  
            {...}  
    }
```

Propagation d'une exception :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
- Pour remonter une exception à la méthode appelante, le programmeur rajoute le mot réservé throws à la déclaration de la méthode dans laquelle l'exception est susceptible de se manifester.
- Si une exception n'est pas attrapée dans le bloc où elle a été levée, elle est transmise au bloc de niveau supérieur (récursivement)
- Si une exception n'est jamais attrapée (catch)
 - 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

Propagation d'une exception :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 "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.

Les exceptions personnalisées : 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.

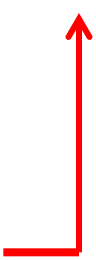
Les exceptions personnalisées : Exemple I

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

Les exceptions personnalisées : Exemple I

```
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 I

```
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

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.