

Support de cours

Cours:

## Introduction à la programmation orientée objet (en Java)

Vidéo:

### W16-02-exceptionsyntaxe-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Prochain bloc catch. Bloc try. Sous-classe de la classe throwable. Instance de la classe exception. Méthodes intéressantes. Instruction throw. Tel bloc. Différents types d'erreur. Pile d'appels. Seule sous-classe. Méthode getmessage. Classe exception. Gestion des exceptions. Exception générale. Exceptions possibles.**



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en Java).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

# Gestion des exceptions : syntaxe

## (Partie 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s

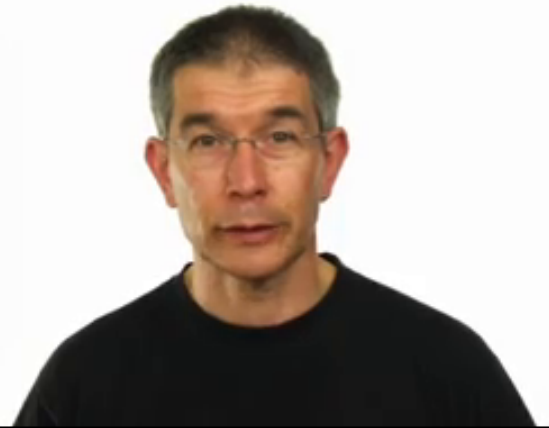


`throw`, en « lançant » une exception, interrompt le cours normal d'exécution et :

- ▶ saute au bloc `catch` du bloc `try` directement supérieur, si il existe ;
- ▶ quitte le programme si l'exécution courante n'était pas dans au moins un bloc `try`.

Exemple :

```
try {  
    // ...  
    throw new Exception("Quelle erreur !");  
    // ...  
}  
catch (Exception e) {  
    // ...  
}
```



Qu'est-ce que ça veut dire concrètement "lancer une exception" ?

notes

---

---

---

---

---

---

---

---

---

---

résumé

---

---

---

---

---

---

---

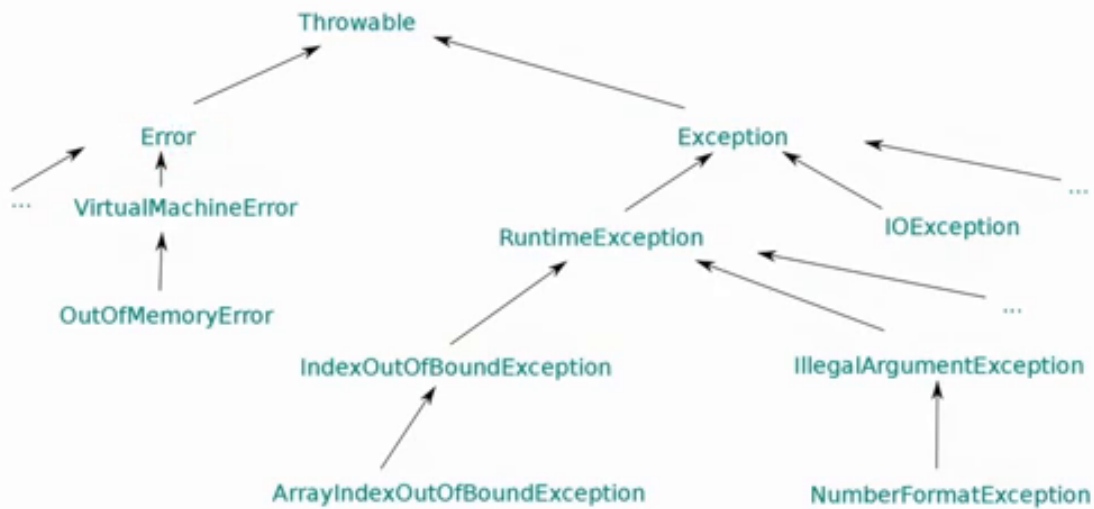
---

---

---

0m 1s





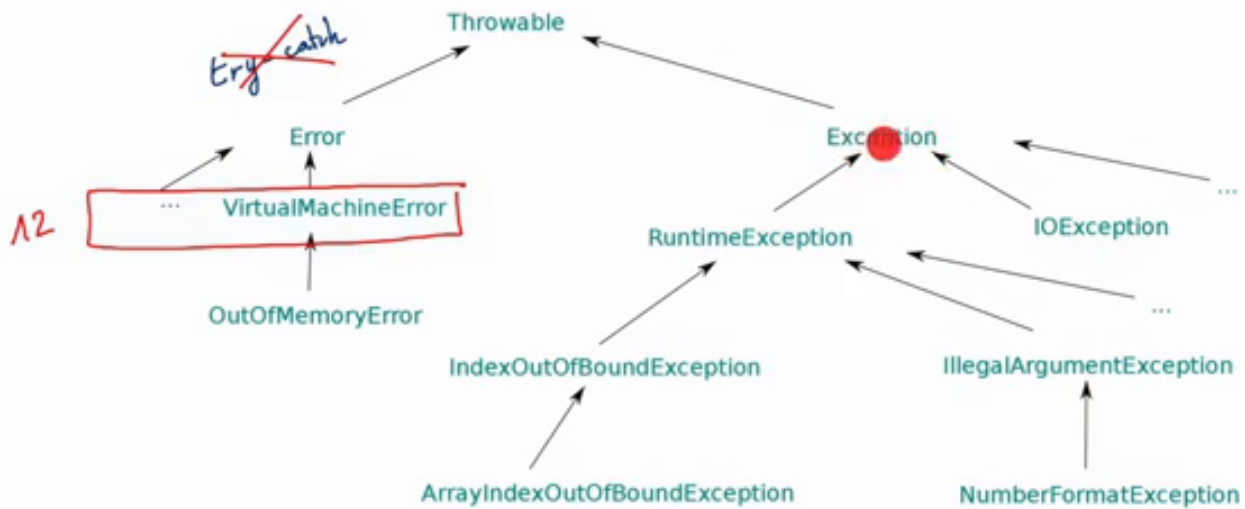
Ca veut dire que l'instruction throw va quitter le cours normal d'exécution du programme et va sauter soit au prochain bloc catch directement lié à un bloc try. On verra catch et try dans un instant. Pour l'instant ce qu'il faut retenir c'est que ça va sauter à un autre endroit du programme. Soit, s'il n'y a pas un tel bloc try/catch, va alors complètement quitter complètement le programme en terminant par ce qu'on appelle une exception. Throw lance donc une instance de la classe Exception. Cette classe Exception est elle-même une sous-classe de la classe Throwable qui hérite elle-même d'Object. Exception n'est pas la seule sous-classe de throwable qui possède une multitude de sous-classes dont le but est d'indiquer différents types d'erreur ou d'exceptions possibles. Nous avons par exemple la sous-classe Error

notes

résumé

0m 6s





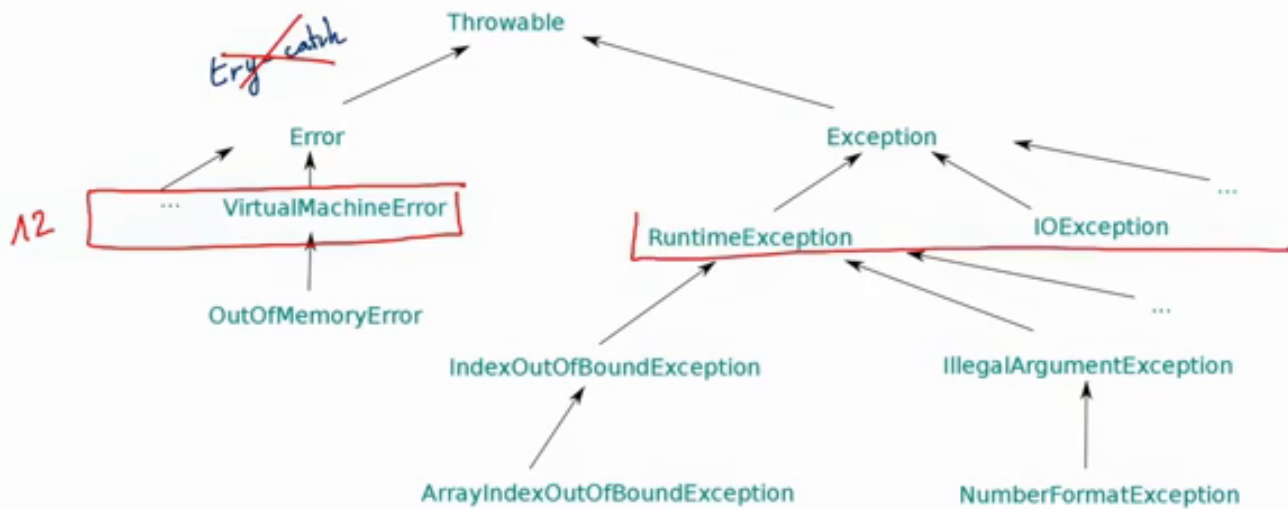
laquelle possède elle-même 12 sous-classes directes et qui sert par exemple à indiquer des erreurs mémoires. La classe Error sert à indiquer des erreurs fatales qui ne sont pas censées être utilisées, traitées par le programmeur avec un bloc try catch.

notes

résumé

1m 2s





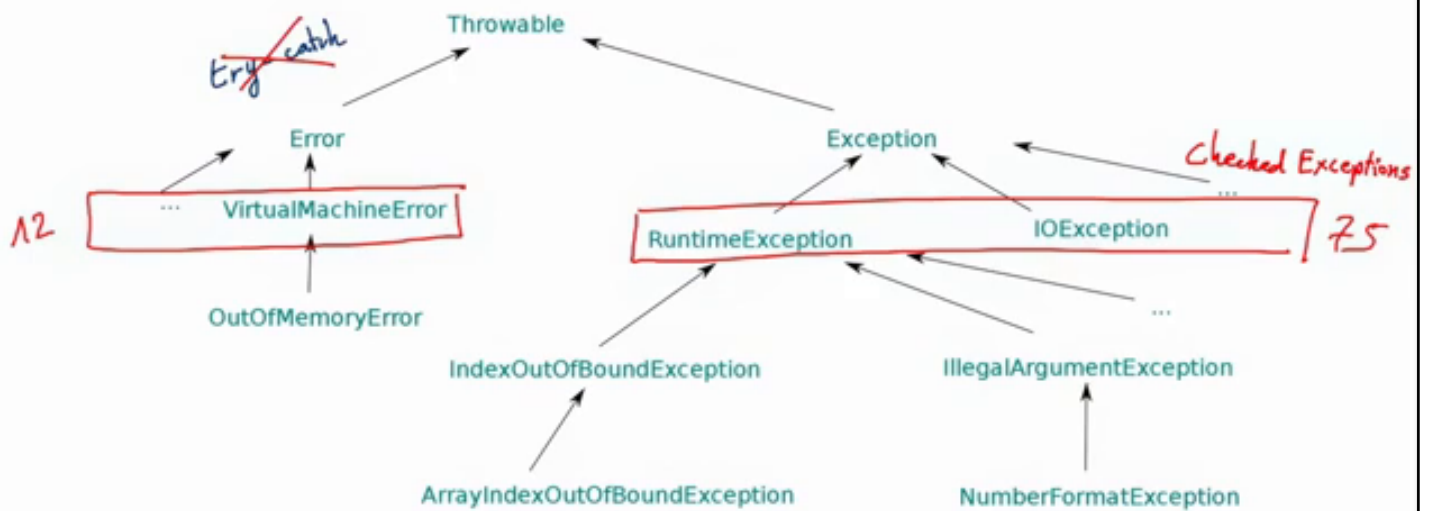
De l'autre côté, nous avons les Exception qui doivent ou peuvent être traitées par le programmeur.

notes

résumé

1m 25s





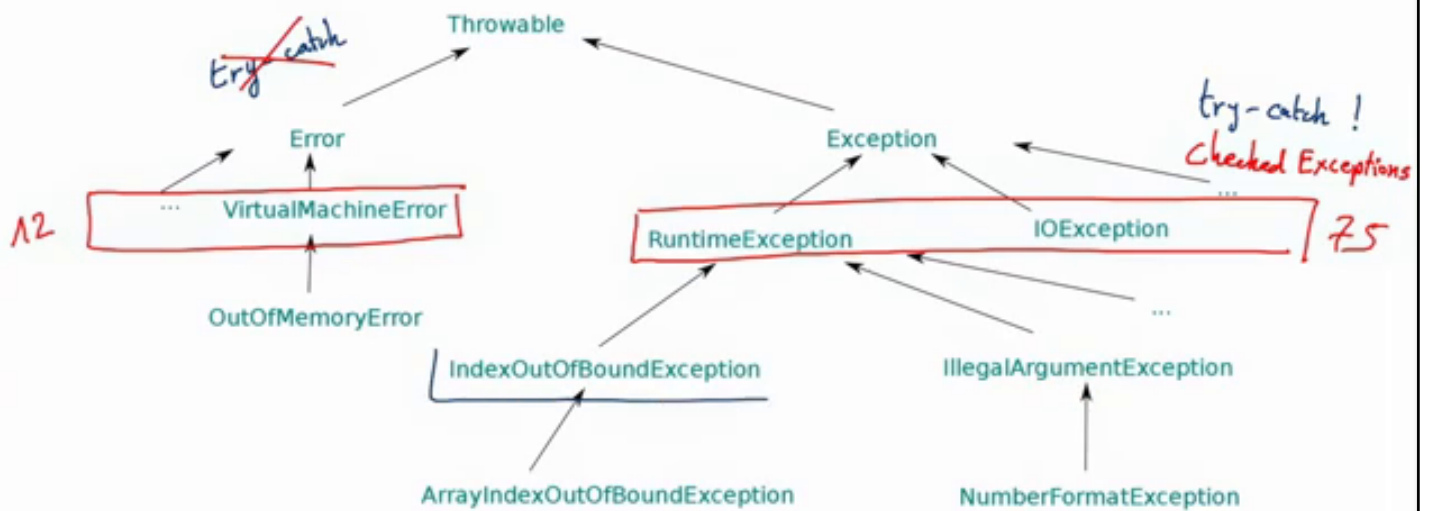
La classe Exception a 75 sous-classes directes parmi lesquelles nous avons 74 Checked Exceptions qui doivent être traitées par le programmeur. Elles indiquent des circonstances exceptionnelles,

notes

résumé

1m 32s





souvent des erreurs mais pas forcément toujours. Elles doivent donc être traitées au niveau du programme par un try/catch Nous avons aussi la sous-classe RunTimeException dont le traitement n'est pas forcément vérifié par le programmeur on appelle ça des Unchecked Exception,

notes

résumé

1m 49s







elle-même ayant 49 sous-classes directes pour traiter différents cas d'erreurs évitables par de la bonne programmation, comme par exemple des indices dépassant des tableaux, comme par exemple des divisions par 0, etc.

notes

## résumé

2m 7s



```
public class Throwable extends Object
```

► Deux constructeurs:

- Erreur avec ou sans message

```
public Throwable()  
public Throwable(String message)
```

► deux méthodes (parmi d'autres) :

- Accès au message d'erreur
- Affichage du chemin vers l'erreur

```
public String getMessage()  
public void printStackTrace()
```

Ce genre d'exception, il n'est pas exclu que le programmeur puisse le traiter, mais ce n'est pas strictement nécessaire. La vidéo suivante reviendra d'ailleurs sur ces Checked/Unchecked Exceptions et la façon de les gérer. Au niveau le plus général, tout en haut de la hiérarchie, la classe Throwable offre deux constructeurs, un constructeur par défaut et un constructeur avec possibilité d'indiquer un message d'erreur sous la forme d'une string. Parmi d'autres, elle offre deux méthodes intéressantes,

## notes

---

---

---

---

---

---

---

---

---

---

## résumé

2m 22s



---

---

---

---

---

---

---

---

---

---



notes

## résumé

2m 49s



`try` (*lit.* « essaye ») introduit un **bloc réceptif aux exceptions** lancées par des instructions, ou des méthodes appelées à l'intérieur de ce bloc (ou même des méthodes appelées par des méthodes appelées par des méthodes... ... à l'intérieur de ce bloc)

Exemple :

```
try {  
    // ...  
    y = f(x); // f pouvant lancer une exception  
    // ...  
}
```

Voilà donc pour l'instruction `throw`. Passons maintenant à l'instruction `try`. Nous avons vu que l'instruction `Throw`, si elle n'était pas attrapée c'est à dire, pas gérée par un `try/catch`, terminait le programme. Donc, on va vouloir dans beaucoup de cas, en particulier pour les Checked Exceptions, on va vouloir les traiter par un bloc sensible à la gestion des exceptions. C'est exactement ce à quoi sert `try`. `try` sert donc à rendre des portions de codes sensibles à la gestion des exceptions. Supposons que je fasse un appel à une méthode `f` qui peut lancer des exceptions, soit la méthode peut lancer des exceptions directement, soit elle peut appeler des méthodes qui appellent des méthodes

notes

résumé

3m 8s



`catch` est le mot-clé introduisant un **bloc dédié à la gestion** d'une ou plusieurs **exceptions**.

Tout bloc `try` doit toujours être suivi d'au moins un bloc `catch` gérant les exceptions pouvant être lancées dans ce bloc `try`.

Si une exception est lancée mais n'est pas interceptée par le `catch` correspondant, le programme s'arrête (message « **Exception in thread ...** » et affichage de la trace d'exécution).

#### Syntaxe :

```
catch (type nom) { ... }
```

intercepte toutes les exceptions de type `type` lancées depuis le bloc `try` précédent  
`type` peut-être une classe prédéfinie de la hiérarchie d'exceptions de Java ou une classe d'exception créée par le programmeur.

qui appellent de méthodes qui lancent des exceptions. Donc peu importe à quelle profondeur d'appel, dans cet appel à la méthode `f`, on a un `throw`. Mais quelque part, en dessous de cet appel, on va avoir un `throw`. Alors à ce moment là, si une telle expression, si un tel appel à une méthode qui peut lancer des exceptions est dans un bloc qui est déclaré comme un bloc `try`, alors à ce moment là, on va pouvoir avoir mise en place du mécanisme de gestion des exceptions et donc le `throw` qui se trouve là quelque part, en-dessous, va finalement sauter au prochain bloc `catch` associé au bloc `try` dans lequel on est en train de demander la gestion des exceptions. `catch` est donc justement ce mot clé qui sert à introduire l'ensemble des instructions que l'on voudra exécuter pour pouvoir gérer les exceptions. Tout bloc `try` doit avoir au moins un bloc `catch` associé, on verra dans un instant qu'il peut en avoir plusieurs,

#### notes

#### résumé

3m 49s



```
try {
    // ...
    if (age >= 150)
    { throw new Exception("valeur trop grande"); }
    // ...
    if (x == 0.0)
    { throw new ArithmeticException("Division par zero"); }
    // ...
}

catch (ArithmeticException e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}

catch (Exception e) {
    System.out.println("Qui peut vivre si vieux?");
}
```

qui va donc gérer les exceptions lancées dans ce bloc try. Si jamais une exception est lancée par un throw, mais qui n'est pas interceptée par un catch, s'il n'y a pas de catch correspondant ni de try pour attraper l'exception qui est lancée, à ce moment là, le programme va s'arrêter en indiquant qu'il y a eu une exception qui a été lancée et afficher l'endroit où elle a été lancée, indiquant que cette exception a été lancée et n'a pas été interceptée. La syntaxe du bloc catch est donc de commencer par le mot catch puis de faire suivre comme un en-tête de méthode, entre parenthèses, le type et l'équivalent d'un paramètre. On aura qu'un seul paramètre pour cet en-tête de catch. Puis derrière, ici le bloc avec les instructions que l'on veut faire exécuter pour toutes les exceptions du type que l'on aura précisé. Le mieux c'est de donner un exemple. On a ici un bloc d'instructions que l'on veut rendre sensible à la gestion des exceptions avec le try, et juste derrière, on aura catch pour attraper ici des exceptions de type ArithmeticException et un autre bloc catch qui se rapporte aussi à ce bloc try pour attraper des exceptions très générales de type Exception. Donc ici, dans ce bloc, on pourrait avoir une variable age, si l'âge est trop élevé, à ce moment là, on va lancer une "exception" très générale. Puis si l'âge est inférieur à 150, on continuerait l'exécution. On aurait une autre variable x. Si x était égal à 0, alors on lance une ArithmeticException. Sinon on continue. Cette ArithmeticException va être interceptée par ce catch-là. On exécutera donc les instructions qui sont ici, en affichant le message d'erreur que l'on avait mis dans l'ArithmeticException, puis par exemple,

## notes

## résumé

4m 49s

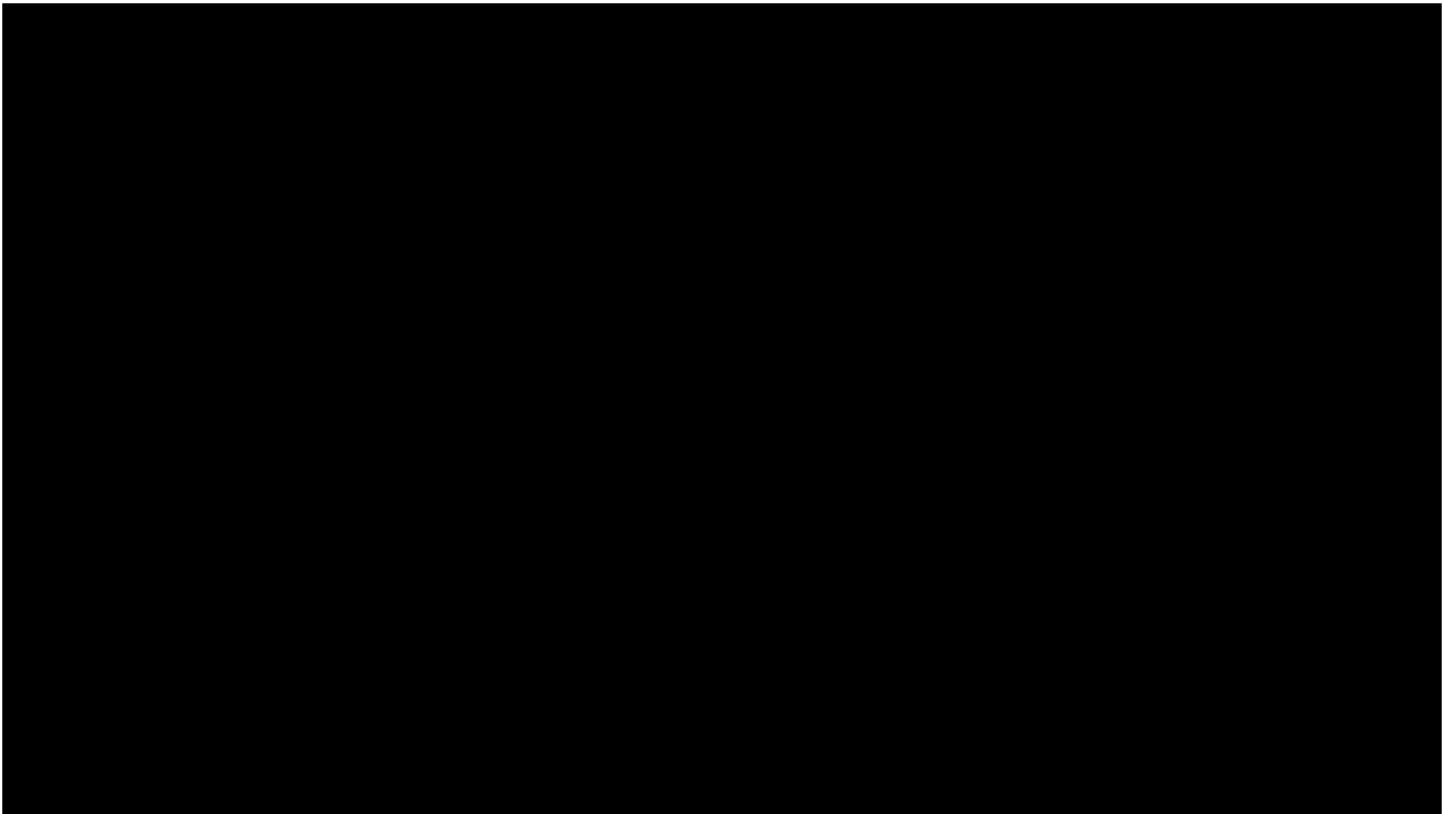


```
try {  
    // ...  
    if (age >= 150)  
    { throw new Exception("valeur trop grande"); }  
    // ...  
    if (x == 0.0)  
    { throw new ArithmeticException("Division par zero"); }  
    // ...  
}  
  
catch (ArithmeticException e) {  
    System.out.println(e.getMessage());  
    e.printStackTrace();  
}  
  
catch (Exception e) {  
    System.out.println("Qui peut vivre si vieux?");  
}
```

en imprimant la pile d'appels. Alors que si c'est cette exception générale qui a été lancée

notes

résumé



c'est ce catch qui va être exécuté. A ce moment là, on affichera un message, par exemple : " Qui peut vivre aussi vieux ?" Ce qui est important de noter, c'est que les catch ici associés à un bloc try vont être ordonnés par niveau d'exception de plus en plus général. On aura d'abord les exceptions les plus spécifiques dans la hiérarchie, puis ensuite les exceptions les plus générales. Je vous rappelle qu'on a une hiérarchie d'exceptions, comme ceci. Le traitement dans le catch devra se faire en commençant par des exceptions les plus spécifiques, puis ensuite en allant vers les plus générales. Si vous placez les blocs catch avec les exceptions les plus générales en premier, vous aurez une erreur de compilation. vous aurez une erreur de compilation.

#### notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

#### résumé

6m 49s



.....

.....

.....

.....

.....