

Support de cours

Cours:

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

Vidéo:

### W16-01-exceptionsintro-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Situations d'erreur. Situations anormales. Gestion des exceptions. Utilisation de ces concepts. Valeur de la donnée entrante. Cas de division. Cas échéant. But de cette séquence. Code d'erreur. Situation d'erreur. Utilisateur potentiel de la fonction. Partie de son rôle normal. Utilisation de la méthode. Fait arbitraire. Dernière possibilité.**



[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 : introduction**

## **(Partie 1)**

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

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





La gestion des exceptions est un volet important de la programmation de base

notes

---

---

---

---

---

---

---

---

---

---

résumé

0m 1s



---

---

---

---

---

Les **exceptions** permettent d'*anticiper les erreurs* qui pourront potentiellement se produire lors de l'utilisation d'une portion de code.

Exemple : on veut écrire une fonction qui calcule l'inverse d'un nombre réel quand c'est possible :

f
entrée : $x$ sortie : $1/x$
<b>Si</b> $x = 0$ <i>erreur</i> <b>Sinon</b> <i>retourner <math>1/x</math></i>

☞ Mais que faire **concrètement** en cas d'erreur ?

qui va permettre de rendre les programmes plus robustes plus résistants face à des situations anormales ou des situations d'erreur ; la gestion des exceptions n'est pas à proprement parler un concept de l'orienté-objet, néanmoins en Java, sa mise en œuvre implique l'utilisation de ces concepts. Le but de cette séquence est de vous présenter les aspects fondamentaux liés à la gestion des exceptions en Java. Nous nous intéresserons dans ce volet à comment gérer des erreurs ou des situations anormales dans un programme.

notes

résumé

0m 6s



① retourner une valeur choisie à l'avance :

```
double f(double x) {  
    if (x != 0.0) {  
        return 1.0 / x;  
    } else {  
        return Double.MAX_VALUE;  
    }  
}
```

Nous allons voir que les exceptions vont nous permettre d'anticiper les erreurs ou les situations anormales et d'y répondre de façon appropriée le cas échéant. Commençons par illustrer l'utilité de cette notion des exceptions au travers d'un petit exemple simple : nous voulons écrire une fonction « f » qui calcule l'inverse d'un nombre donné. Nous voulons que cette fonction gère le cas où la valeur de la donnée entrante est 0 comme une situation anormale, une situation d'erreur et nous nous posons la question de comment gérer cette erreur à proprement parler. Première tentative,

notes

résumé

0m 37s



① retourner une valeur choisie à l'avance :

```
double f(double x) {  
    if (x != 0.0) {  
        return 1.0 / x;  
    } else {  
        return Double.MAX_VALUE;  
    }  
}
```

on peut faire en sorte que lorsque la donnée entrante vaut 0 on retourne une valeur choisie à l'avance, pour mimer le fait que le résultat serait proche de l'infini, on pourrait imaginer de façon tout à fait arbitraire de retourner le plus grand « double » possible ;

notes

résumé

1m 9s



① retourner une valeur choisie à l'avance :

```
double f(double x) {  
    if (x != 0.0) {  
        return 1.0 / x;  
    } else {  
        return Double.MAX_VALUE;  
    }  
}
```

Mais cela

1. **n'indique pas** à l'utilisateur potentiel qu'il a fait une erreur
2. retourne de toutes façons un **résultat inexact** ...
3. suppose une **convention arbitraire** (la valeur à retourner en cas d'erreur)

il existe d'autres valeurs prédéfinies en java « infinity » pour mimer l'infini mais ce n'est pas ici le but de notre propos. Cette façon de procéder est néanmoins loin d'être satisfaisante;

notes

résumé

1m 25s



② afficher un message d'erreur :

```
double f(double x) {  
    if (x != 0.0) {  
        return 1.0 / x;  
    } else {  
        System.out.println("Erreur dans f : division par 0");  
        return ???;  
    }  
}
```

mais que retourner effectivement en cas d'erreur ?...  
...on retombe en partie sur le cas précédent

premièrement cela n'indique pas du tout à l'utilisateur potentiel de la fonction « f » qu'il se trouve dans une situation d'erreur ou une situation anormale lorsque 'x' est nul, à nul moment cet utilisateur n'est averti de ce fait. Le résultat retourné est forcément inexact et l'utilisation de « f » implique une convention tout à fait arbitraire qui n'est pas forcément connue des autres programmeurs ; typiquement le choix de cette valeur « retourner ».

notes

résumé

1m 34s





## ② afficher un message d'erreur :

```
double f(double x) {  
    if (x != 0.0) {  
        return 1.0 / x;  
    } else {  
        System.out.println("Erreur dans f : division par 0");  
        return ???;  
    }  
}
```

mais que retourner effectivement en cas d'erreur ?...

...on retombe en partie sur le cas précédent

De plus, cela est **très mauvais** car produit des *effets de bord* : affichage dans le terminal alors que ce n'est pas du tout dans le rôle de `f` !

Pour palier au manque d'informations qu'a l'utilisateur de la fonction « `f` » en cas de division par zéro on peut imaginer de faire afficher à notre fonction « `f` » un message d'erreur ; dans le cas où la donnée entrante serait nulle, on ferait afficher à notre fonction qu'il y a une situation d'erreur, une division par 0. Cela ne résout pas tous les problèmes de la version précédente puisque nous avons toujours le problème de quoi retourner exactement en cas d'erreur. Cette solution est aussi critiquable par le fait qu'elle génère ce qu'on appelle des effets de bord, c'est-à-dire que la fonction « `f` » dont l'unique rôle est normalement de calculer l'inverse d'un nombre, ce met tout à coup à produire des affichages dans le terminal ce qui ne fait pas du tout partie de son rôle normal, de plus on imagine le cas où nous utiliserions « `f` » dans le cadre d'un programme graphique nous ne voudrions pas qu'une situation anormale soit signalée par un message sur le terminal

## notes

## résumé

2m 1s



### ③ retourner un code d'erreur :

```
boolean f(double x, Double resultat) {  
    if (x != 0.0) {  
        resultat = 1.0 / x;  
        return true;  
    } else {  
        return false;  
    }  
}
```

mais plutôt par l'ouverture d'une fenêtre d'alerte par exemple. Enfin la fonction « f » qui va être la partie du programme, qui détecte la situation d'erreur, n'a pas forcément toutes les cartes en main pour gérer proprement cette situation-là : il se peut qu'une division par 0 soit quelque chose de très préjudiciable qui nécessite d'arrêter le programme il se peut aussi qu'une division par 0 puisse être traitée d'une autre manière dépendamment du contexte d'utilisation; c'est plutôt la partie du programme qui appelle la fonction « f » qui est informée du contexte et qui va pouvoir apporter les réponses appropriées au cas où on a une division par 0.

### notes

### résumé

2m 49s



## ③ retourner un code d'erreur :

```
boolean f(double x, Double resultat) {  
    if (x != 0.0) {  
        resultat = 1.0 / x;  
        return true;  
    } else {  
        return false;  
    }  
}
```

Cette solution est déjà **meilleure** car elle laisse à la fonction qui appelle **f** le soin de décider quoi faire en cas d'erreur.

Cela présente néanmoins l'inconvénient d'être assez lourd à gérer pour finir :

- ▶ cas de l'appel d'appel d'appel.... ..d'appel de fonction,
- ▶ mais aussi écriture peu intuitive :  
`if (f(x,y))... // le résultat de la division est dans y`  
au lieu de  
`y=f(x);`

Une dernière possibilité enfin serait de réécrire à la fonction « f » de sorte à ce qu'elle retourne plutôt un code d'erreur indiquant si la division s'est bien passée ou s'est mal passée et qu'elle restitue le résultat de la division dans le cadre d'une utilisation normale sans erreur au travers d'un paramètre. Cette façon de procéder est meilleure que celles précédemment décrites, dans la mesure où elle laisse à la fonction qui appelle « f » le soin de décider quoi faire en cas d'erreur ça n'est pas à la fonction « f » elle-même de décider ce qu'il faut faire, elle informe celui qui l'appelle du résultat de son exécution et c'est à l'appelant de décider quoi faire en cas d'erreur. Avec cette façon de procéder l'utilisation de la méthode devient cependant beaucoup moins intuitive au lieu de simplement écrire quelque chose comme ceci qui veut dire stockons dans 'y' le résultat de l'inversion de 'x' calculé par la méthode « f », et bien il faudrait passer par des tournures de ce style qui impliquent donc de savoir que « f » retourne un code d'erreur et que le résultat normal d'utilisation est stocké dans le second paramètre ce qui est clairement beaucoup moins intuitif que cette façon d'écrire les choses ; de plus lorsque une méthode en appelle une autre qui à son tour en appelle une autre et ainsi de suite, chaque appel peut potentiellement produire un code d'erreur il faut donc à ce moment-là gérer la combinaison de tous les codes d'erreurs possibles; cela devient rapidement très lourd à gérer. Nous souhaitons donc à ce stade disposer d'un mécanisme

## notes

## résumé

3m 25s



Il existe une solution permettant de *généraliser* et d'*assouplir* cette dernière solution : déclencher une **exception**

- ☛ mécanisme permettant de *prévoir une erreur* à un endroit et de **la gérer à un autre endroit**

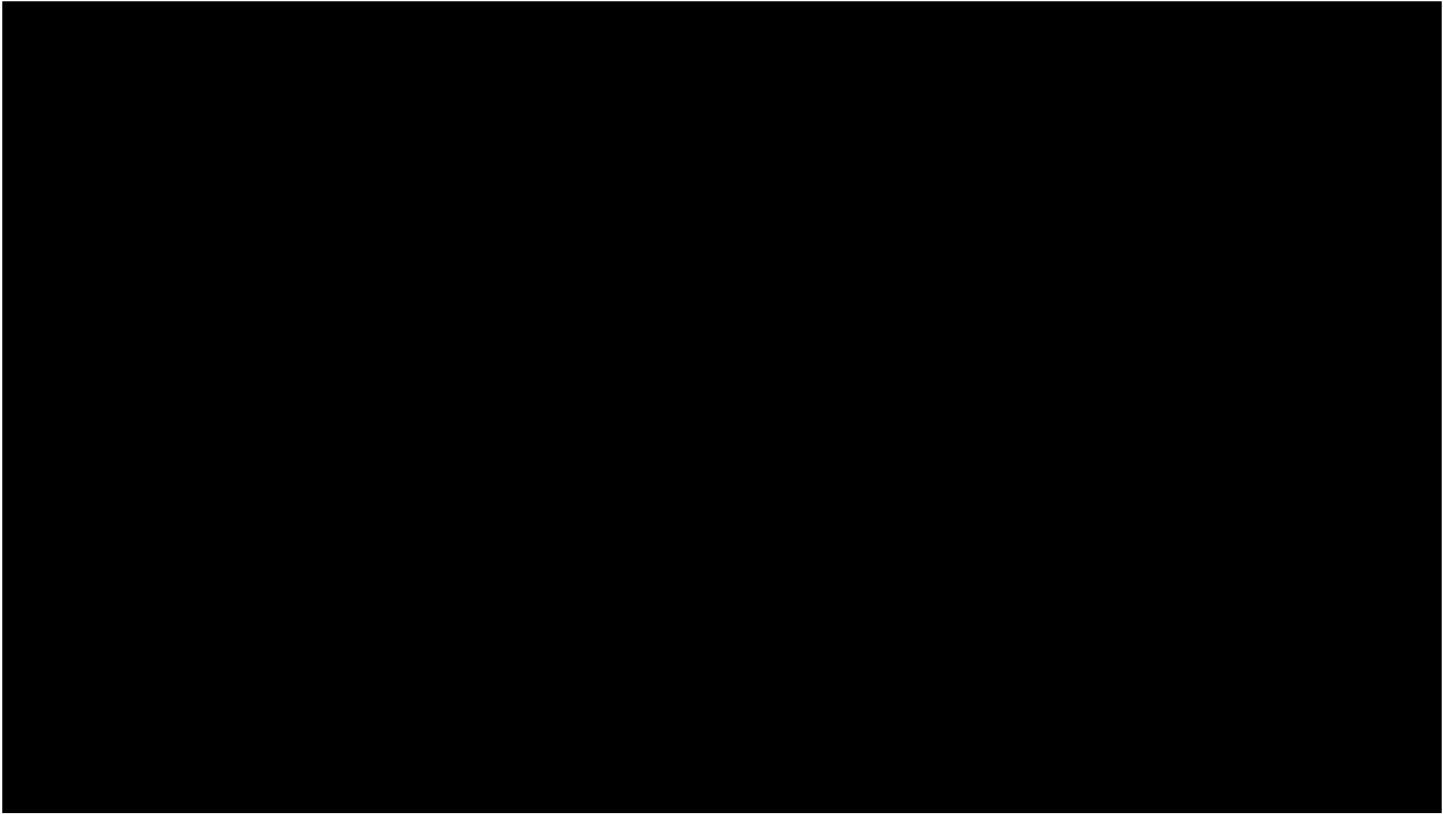
qui permettrait de signaler une situation d'erreur, sans pour autant produire des « effets de bords » indésirés et aussi un mécanisme qui permettrait de gérer une erreur à un autre endroit qu'à l'endroit où elle serait détectée ; et ceci tout en gardant une souplesse d'utilisation, une utilisation intuitive des différentes fonctionnalités que l'on veut avoir dans le programme.

notes

résumé

4m 49s





La gestion de ces exceptions répond précisément à ce besoin, elle va nous permettre à l'endroit où l'on détecte l'erreur de déclencher une exception et de la gérer à un autre endroit potentiellement dans le programme et ce sans intégrer explicitement de codes d'erreur dans les différentes fonctionnalités que l'on souhaite mettre en oeuvre. que l'on souhaite mettre en oeuvre.

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

5m 9s



.....

.....

.....

.....

.....