

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W16-01-exceptionsintro-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Endroit du programme. Mécanisme de gestion des exceptions. Exécution du bloc réceptifs. Exemple concret. Reste du programme. Situation anormale. Programme principal. Graphique de l'inverse. Gestion des exceptions. Ensemble d'instructions capable. Contexte de l'erreur. Erreur. Endroits réceptifs. Moyen du mot. Exemple d'un appareil de mesure.



[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 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



Principe :

- ▶ lorsque qu'une erreur a été détectée à un endroit, on la signale en « *lançant* » *un objet* contenant toutes les informations que l'on souhaite donner sur l'erreur (« lancer » = créer un objet disponible pour le reste du programme)
- ▶ à l'endroit où l'on souhaite gérer l'erreur (au moins partiellement), on peut « *attraper* » *l'objet* « *lancé* » (« attraper » = utiliser)
- ▶ si un objet « lancé » n'est pas attrapé du tout, cela provoque l'arrêt du programme : *toute erreur non gérée provoque l'arrêt.*

Un tel mécanisme s'appelle « gestion des exceptions ».

Le mécanisme de gestion des exceptions va fonctionner selon le principe suivant

notes

résumé

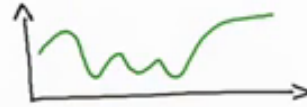
0m 1s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiescer(temperatures);  
    graphiqueInverse(temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse(double[] temps) {  
    for (double t : temps) {  
        afficher(inverse(t));  
    }  
}
```

```
double inverse(double x) {  
    return 1/x;  
}
```

donc selon les trois points qui sont développés ici. Lorsqu'une erreur ou une situation anormale est détectée à un endroit du programme eh bien il faut la signaler au reste du programme en lançant un objet qui va contenir les informations utiles pour pouvoir éventuellement traiter cette erreur ou cette situation anormale ailleurs dans le programme. Lancer ou déclencher une exception veut dire concrètement créer un objet qui va devenir disponible pour le reste du programme. Cet objet peut être attrapé, utilisé, à un autre endroit du programme qui se chargera de gérer la situation anormale. Complètement ou éventuellement de façon partielle. et finalement si l'objet lancé n'est jamais attrapé alors ceci doit provoquer l'arrêt du programme une erreur n'a pas été gérée et donc va provoquer l'arrêt du programme. Illustrons ceci sur un exemple concret.

notes

résumé

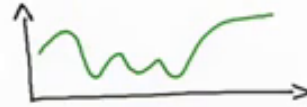
0m 6s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiesir (temperatures);  
    graphiqueInverse (temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse (double[]  
    temps) {  
    for (double t : temps) {  
        afficher (inverse (t));  
    }  
}
```

```
double inverse (double x) {  
    return 1/x;  
}
```

Imaginons par exemple que nous ayons un programme principal qui travaille avec des températures; des températures acquises au travers par exemple d'un appareil de mesure. Ces températures sont stockées dans un tableau et ce programme a pour rôle de dessiner un graphique de l'inverse de toutes les températures acquises. imaginons que les températures soient acquises au travers d'un appareil de mesure qui n'est pas tout à fait fiable, et que de temps en temps il ne soit pas capable de mesurer correctement les températures en question ce qui se traduirait par des valeurs nulles dans notre tableau de température.

notes

résumé

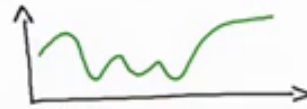
1m 1s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiescer (temperatures);  
    graphiqueInverse (temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse (double[] temps) {  
    for (double t : temps) {  
        afficher (inverse (t));  
    }  
}
```

```
double inverse (double x) {  
    return 1/x;  
}
```

Le programme principal ici fait appel pour réaliser ces traitements à une autre fonctionnalité, « graphiqueInverse », qui va itérer sur l'ensemble des températures du tableau fournies et qui ven afficher l'inverse. Cette méthode fait donc appel à son tour à une autre méthode calculant cette fois l'inverse d'un double. La méthode inverse n'est pas du tout informée du contexte de son utilisation elle ne sait pas concrètement quoi faire, si la valeur qui lui est fournie est 0 donc ce qui correspondrait à cette valeur dans le tableau par contre le programme principal sait dans quel contexte est appelée cette méthode

notes

résumé

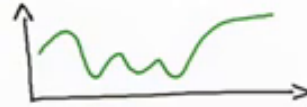
1m 35s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiesir (temperatures);  
    graphiqueInverse (temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse (double[]  
    temps) {  
    for (double t : temps) {  
        afficher (inverse (t));  
    }  
}
```

```
double inverse (double x) {  
  
    return 1/x;  
}
```

et c'est lui qui est à même de résoudre le problème par exemple décider que si le tableau de température n'a pas pu être affiché correctement, c'est que les températures acquises sont corrompues et qu'il faut les acquérir à nouveau.

notes

résumé

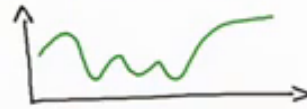
2m 13s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiescer(temperatures);  
    graphiqueInverse(temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse(double[] temps) {  
    for (double t : temps) {  
        afficher(inverse(t));  
    }  
}
```

```
double inverse(double x) {  
    return 1/x;  
}
```

→ lancement d'une excep

L'idée derrière la gestion des exceptions est que la partie du programme qui détecterait une situation anormale sans pour autant savoir la traiter localement la signale au reste du programme en lançant une exception,

notes

résumé

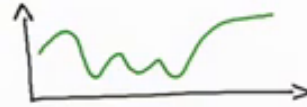
2m 27s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiescer(temperatures);  
    graphiqueInverse(temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse(double[] temps){  
    for (double t : temps){  
        afficher(inverse(t));  
    }  
}
```

```
double inverse(double x){  
    return 1/x;  
}
```

→ lancement d'une exception
(objet)

donc il s'agit d'un objet particulier qui va être lancé au reste du programme si cet objet lancé n'est jamais rattrapé par aucune des autres parties du programme, le programme s'arrête : on n'a pas traité l'erreur, mais on peut aussi bien sûr faire en sorte qu'une autre partie du programme plus informée du contexte de l'erreur rattrape cet objet et le traite de façon appropriée. Le programme principal devrait alors ici indiquer qu'il est réceptif aux objets lancés qu'il est capable de les recevoir

notes

résumé

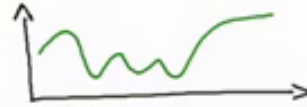
2m 45s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiescer(temperatures);  
    graphiqueInverse(temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
void graphiqueInverse(double[] temps) {  
    for (double t : temps) {  
        afficher(inverse(t));  
    }  
}
```

```
double inverse(double x) {  
    return 1/x;  
}
```

→ lancement d'une exception
(objet)

et dans ce cas-là il va pouvoir rattraper l'objet et le traiter,

notes

résumé

3m 13s



On cherche à remplir 4 tâches élémentaires :

1. signaler une erreur
2. marquer les endroits réceptifs aux erreurs
3. leur associer (à chaque endroit réceptif) un moyen de gérer les erreurs qui se présentent
4. éventuellement, « faire le ménage » après un bloc réceptif aux erreurs

On a donc 4 mots du langage Java dédiés à la gestion des exceptions :

`throw` : indique l'erreur (i.e. « lance » l'exception)

`try` : indique un bloc réceptif aux erreurs

`catch` : gère les erreurs associées (i.e. les « attrape » pour les traiter)

`finally` : (optionel) indique ce qu'il faut faire après un bloc réceptif

s'il a été lancé bien sûr pendant l'exécution du bloc réceptifs aux erreurs.

notes

résumé

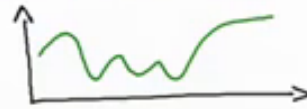
3m 23s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiescer(temperatures);  
    graphiqueInverse(temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
void graphiqueInverse(double[] temps) {  
    for (double t : temps) {  
        afficher(inverse(t));  
    }  
}
```

```
double inverse(double x) {  
    return 1/x;  
}
```

→ lancement d'une exception
(objet)

Pour mettre en œuvre un tel processus il faut pouvoir réaliser quatre tâches élémentaires. La première est le signalement d'une erreur, c'est ce que faisait notre méthode inverse qui lançait un objet pour alerter le reste du programme d'une situation anormale.

notes

résumé

3m 32s



On cherche à remplir 4 tâches élémentaires :

1. signaler une erreur
2. marquer les endroits réceptifs aux erreurs
3. leur associer (à chaque endroit réceptif) un moyen de gérer les erreurs qui se présentent
4. éventuellement, « faire le ménage » après un bloc réceptif aux erreurs

On a donc 4 mots du langage Java dédiés à la gestion des exceptions :

`throw` : indique l'erreur (i.e. « lance » l'exception)

`try` : indique un bloc réceptif aux erreurs

`catch` : gère les erreurs associées (i.e. les « attrape » pour les traiter)

`finally` : (optionel) indique ce qu'il faut faire après un bloc réceptif

Signaler une erreur était donc ce qui se passait à cette étape de notre exemple. la seconde tâche est de marquer les endroits réceptifs aux erreurs, c'est ce que faisait ici notre programme principal pour indiquer qu'il est capable de rattraper et de traiter un objet correspondant à une situation anormale; à chaque endroit réceptif aux erreurs, aux situations anormales, il faut fournir un moyen de gérer ces erreurs, donc de rattraper proprement l'objet et de le traiter; dans notre exemple à ce bloc réceptif aux erreurs est associé un ensemble d'instructions capable de traiter de rattraper l'objet proprement.

notes

résumé

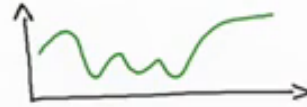
3m 47s



Exceptions

```
-- main (-) {  
    double[] temperatures = new ...;  
    acquiescer(temperatures);  
    graphiqueInverse(temperatures);  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
void graphiqueInverse(double[] temps) {  
    for (double t : temps) {  
        afficher(inverse(t));  
    }  
}
```

```
double inverse(double x) {  
    return 1/x;  
}
```

→ lancement d'une exception
(objet)

Enfin dans certains cas il peut être nécessaire de faire le ménage après un bloc réceptif aux erreurs c'est-à-dire par exemple libérer un certain nombre de ressources qui auraient été sollicitées mais pas proprement libérées en raison d'une situation anormale. A chacune de ces tâches va correspondre un mot-clé du langage Java. Pour la tâche du signalement d'erreur, le mot-clé « throw », pour celle de marquer les endroits réceptifs aux erreurs, « try », pour associer à un bloc réceptif aux erreurs un bloc de traitement de l'erreur, le mot-clé « catch », et enfin pour faire le ménage, le mot-clé « finally ». La description détaillée de l'utilisation de ces mots-clés vous sera présentée dans la séquence vidéo suivante; en clair

notes

résumé

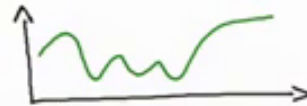
4m 25s



Exceptions

```
-- main (-) {  
try {  
    double[] temperatures = new ...;  
    acquiescer (temperatures);  
    graphiqueInverse (temperatures);  
} catch {  
    rattraper et traiter  
    l'objet  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse (double[]  
    temps) {  
    for (double t : temps) {  
        afficher (inverse (t));  
    }  
}
```

```
double inverse (double x) {  
    throw .....  
    return 1/x;  
}
```

→ lancement d'une exception
(objet)

l'endroit du programme qui détecte l'erreur va lancer l'objet en utilisant le mot-clé réservé « throw », nous verrons un peu plus loin qu'est ce qu'il va exactement lancer. L'endroit réceptif aux objets lancés doit être à signalé au moyen du mot réservé, « try »,

notes

résumé

5m 14s



Notez bien que :

- ▶ L'indication des erreurs (`throw`) et leur gestion (`try/catch`) sont le plus souvent à *des endroits bien séparés* dans le code
- ▶ Chaque bloc `try` possède son/ses `catch` associé(s)

donc le bloc réceptif aux erreurs est indiqué au moyen de ce mot clé et à chaque bloc réceptif aux erreurs doit correspondre un bloc capable de traiter l'erreur qui va être signalée au moyen du mot clé, « `catch` », dont nous verrons la syntaxe précise dans les séquences suivantes.

notes

résumé

5m 37s



Une exception est un moyen de signaler un événement nécessitant une attention spéciale au sein d'un programme, comme :

- ▶ une **erreur grave**
- ▶ une **situation inhabituelle** devant être traitées de façon particulière

But : **améliorer la robustesse des programmes** en :

- ▶ séparant le code de traitement des erreurs du code « effectif »
- ▶ fournissant le moyen de forcer une réponse à des erreurs particulières

Comme vous l'aurez sans doute constaté dans le cadre de notre petit exemple, le signalement des erreurs au travers du mot clé, « throw », et leur gestion au travers des blocs « try » et « catch », sont le plus souvent à des endroits bien séparés dans le code. C'est ce qui fait d'ailleurs l'intérêt principal de la gestion des exceptions dans notre exemple le « throw » était dans la méthode inverse, les blocs « try » et « catch » étaient dans le programme principal. Nous verrons dans les séquences qui suivent qu'à chaque « try » doit correspondre un « catch », ou un certain nombre de « catch » qui lui sont associés. pour résumer une exception est donc un moyen de signaler un événement qui nécessite une attention particulière dans le programme, cet événement peut-être une erreur grave qui nécessite d'arrêter le programme,

notes

résumé

5m 54s



Avantages de la gestion des exceptions par rapports aux codes d'erreurs retournés par des fonctions :

- ▶ écriture plus facile, plus intuitive et plus lisible $y = \text{inverse}(x,) ;$
- ▶ propagation **automatique** de l'exception aux niveaux supérieurs d'appel (fonction appelant une fonction appelant ...)
plus besoin de gérer obligatoirement l'erreur au niveau de la fonction appelante
- ▶ une erreur peut donc se produire à n'importe quel niveau d'appel, elle sera toujours reportée par le mécanisme de gestion des exceptions

Note : si une erreur peut être gérée localement, le faire et ne pas utiliser le mécanisme des exceptions.)

mais ça peut être aussi simplement une situation inhabituelle qu'il faudra traiter de façon appropriée. C'était le cas dans notre exemple précédent où une division par zéro correspondait à une situation inhabituelle, un ensemble de mesure qu'il faudrait reprendre par exemple, et qui ne correspond pas à une situation dramatique nécessitant d'arrêter le programme. La gestion des exceptions va donc permettre d'améliorer la robustesse des programmes et va permettre de séparer le code qui est dédié au traitement des erreurs du code qui réalise les autres traitements qui se passent habituellement sans situation anormale. La gestion des exceptions va également fournir un moyen de forcer une réponse à des erreurs particulières, comme par exemple ici une situation inhabituelle qu'il faudrait gérer de façon appropriée. Par opposition à la solution présentée en début de séquence où l'on traitait les situations anormales en faisant en sorte que les fonctions ou méthodes retournent des codes d'erreurs, la gestion des exceptions offre un certain nombre d'avantages; l'un des plus importants est une écriture plus facile, plus intuitive, plus lisible, par exemple, notre méthode calculant l'inverse peut continuer à être utilisée de façon très naturelle, il n'est pas nécessaire d'en polluer l'écriture en ajoutant

notes

résumé

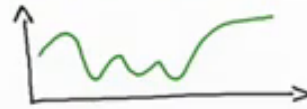
6m 37s



Exceptions

```
-- main (-) {  
try {  
    double[] temperatures = new ...;  
    acquiescer (temperatures);  
    graphiqueInverse (temperatures);  
} catch (...) {  
    rattraper et traiter  
    l'objet  
}
```

{ 35.0, 35.7, 37.2, 0.0, 38.2 ... }



```
Void graphiqueInverse (double[]  
    temps) {  
    for (double t : temps) {  
        afficher (inverse (t));  
    }  
}
```

```
double inverse (double x) {  
    throw .....  
    return 1/x;  
}
```

→ lancement d'une exception
(objet)

d'autres paramètres uniquement dédiés à la gestion des erreurs. la propagation de l'exception à des niveaux supérieurs d'appel va se faire de façon automatique, il n'est pas nécessaire que chaque niveau intermédiaire d'appel se charge du traitement de l'erreur;

notes

résumé

7m 49s



Avantages de la gestion des exceptions par rapports aux codes d'erreurs retournés par des fonctions :

- ▶ écriture plus facile, plus intuitive et plus lisible $y = \text{inverse}(x)$;
- ▶ propagation **automatique** de l'exception aux niveaux supérieurs d'appel (fonction appelant une fonction appelant ...)
plus besoin de gérer obligatoirement l'erreur au niveau de la fonction appelante
- ▶ une erreur peut donc se produire à n'importe quel niveau d'appel, elle sera toujours reportée par le mécanisme de gestion des exceptions

Note : si une erreur peut être gérée localement, le faire et ne pas utiliser le mécanisme des exceptions.)

par exemple si l'exception lancée par la méthode inverse n'est pas rattrapée par la méthode appelante directe « graphiqueInverse », l'objet va être automatiquement relancé au niveau supérieur, qui pourra éventuellement les traiter comme si le programme principal.

notes

résumé

8m 6s



Une erreur peut donc désormais se produire à n'importe quel niveau d'appel tout en restant reportée pour le reste du programme. Comme nous venons de le voir la gestion des exceptions présente donc de nombreux avantages pour traiter des situations d'erreurs des situations anormales dans un programme. Les mécanismes sous-jacents ont cependant un certain coût, et il en résulte une consigne méthodologique que nous aurons l'occasion de rediscuter dans les séquences suivantes, et qui est la suivante : Si une erreur peut être gérée localement c'est-à-dire qu'on est complètement informé à un endroit donné sur la façon de résoudre l'erreur, alors il faut le faire localement, tout de suite, et ne pas utiliser le mécanisme des exceptions. le mécanisme des exceptions.

8m 21s

