

Support de cours

Cours:

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

Vidéo:

### W16-03-exceptionscomplements-JAVA-pt4

Concepts (extraits des sous-titres générés automatiquement) :

**Gestion des exceptions. Travers d'une méthode. Nombre de températures mesurées. Nombre de mesures. Exécution de cette méthode. Tableau dynamique de doubles. Exception de type arithmeticexception. Déclenchement d'une exception. Appel de cette méthode. Bloc réceptif. Utilisateur de ce programme. Petit exemple d'introduction. Valeur nulle. Inverse des températures. Tableau de mesures.**



[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 : compléments

## (Partie 4)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Voilà, nous en avons terminé avec les compléments

notes

---

---

---

---

---

---

---

---

---

---

résumé

0m 1s



---

---

---

---

---

```
public static void main(String[] args) {  
    int nbEssais = 0;  
    final int MAX_ESSAIS = 2;  
    ArrayList<Double> mesures = new ArrayList<Double>();  
  
    do {  
        nbEssais++;  
  
        acquerirTemp(mesures);    // remplit le tableau  
  
        try {  
            plotTempInverse(mesures);  
        }  
        // ...  
    }  
}
```

que nous souhaitons apporter sur la gestion des exceptions en java. Pour conclure sur ce volet, nous allons simplement vous présenter le petit exemple d'introduction, mais cette fois, codé en java. Alors il s'agit de notre programme qui avait pour but d'afficher l'inverse d'un certain nombre de températures mesurées, commençons par l'écriture du programme principal tel qu'on pourrait l'imaginer en java. Le programme principal va stocker un certain nombre de mesures dans un tableau dynamique de doubles. Ses températures sont acquises aux travers d'une méthode, qui va permettre le remplissage du tableau, et nous voulons donc afficher l'inverse des températures ainsi stockées dans le tableau. Si le tableau de mesures contient des valeurs nulles, l'affichage des températures inverses va produire une erreur. Nous souhaitons cependant donner plusieurs chances à l'utilisateur de ce programme, et lui permettre de saisir à nouveau son tableau de mesures au cas où une situation anormale se serait présentée. Nous lui accordons dans cet exemple deux essais au maximum, pour tenter de saisir un tableau de mesure qui ne contient aucune valeur nulle. L'idée est donc la suivante : sachant que l'appel de cette méthode peut résulter dans le déclenchement d'une exception, nous allons indiquer que cette instruction est un bloc réceptif aux erreurs, donc l'encadrer par un bloc try, si le tableau de mesures contient une mesure nulle,

## notes

## résumé

0m 6s



```
catch (ArithmeticException e) {  
    if (nbEssais < MAX_ESSAIS) {  
        System.out.println("Ressaisir les valeurs !");  
    } else {  
        System.out.println("Il y a déjà eu au moins "  
            + MAX_ESSAIS + " essais.");  
        System.out.println(" -> abandon");  
    }  
}  
} while (nbEssais < MAX_ESSAIS);  
}
```

l'exécution de cette méthode va lancer une exception, on suppose ici que c'est une exception de type `ArithmeticException`, laquelle sera rattrapée par un bloc `catch` correspondant. Si nous n'avons pas encore atteint le nombre d'essai maximal, on va simplement signifier à l'utilisateur qu'il doit ressaisir des valeurs, sinon, par un autre message, nous lui indiquons qu'il y a eu suffisamment d'essais et qu'il y a donc abandon de l'acquisition des mesures. Lorsque nous terminons l'exécution de ce bloc `catch`, l'exécution se poursuit en séquence, l'exécution va tomber sur l'évaluation de la condition de continuation du `while`. Le `while` continuera à s'exécuter

## notes

## résumé

1m 25s



```
private static void plotTempInverse(ArrayList<Double> t)
throws ArithmeticException
{
    for(int i = 0; i < t.size(); i++) {
        try {
            plot(inverse(t.get(i)));
        } catch (ArithmeticException e) {
            System.out.println("Problème à l'indice :" + i);
            // RELANCEMENT
            throw e;
        }
    }
}
```

si le nombre d'essais n'est pas encore atteint. Ceci va donc permettre de ressaisir le tableau de mesures, tant que le nombre maximal d'essais n'est pas atteint. La méthode en charge d'afficher les inverses de températures va donc prendre un tableau de température en paramètres, elle va itérer sur chacune des entrées de ce tableau, et afficher pour chaque entrée  $i$ , l'inverse de la température stockée à la position  $i$ . Comme le calcul de l'inverse peut lancer une exception, que nous supposons ici, de type `ArithmeticException`, nous prenons le soin d'indiquer que cette instruction est un bloc réceptif aux erreurs, donc entouré par un bloc `try`. Ceci permettra de signaler que le problème s'est produit au niveau de l'indice  $i$ . Comme nous ne savons pas complètement résoudre le problème à ce niveau là, nous relançons l'exception, et ici, nous relançons exactement l'exception déjà reçue.

## notes

## résumé

2m 1s



```
private static void plot(double x) {  
    // fait le dessin  
}  
  
private static double inverse(double x)  
    throws ArithmeticException // PAS NECESSAIRE  
                                //RunTimeException  
{  
    if (x == 0.0) {  
        throw new ArithmeticException("Division par 0 !");  
    }  
    return 1.0/x;  
}
```

On aurait pu en créer une autre. Vous aurez peut-être remarqué au passage que nous avons déclaré l'exception potentiellement lancée dans `plotTemplInverse`, était-ce nécessaire ici ? Alors en effet, il n'est pas nécessaire de déclarer l'exception lancée ici. Pourquoi ? Parce que `ArithmeticException` est de type `RunTimeException`, et qu'elle n'est donc pas soumise à la règle déclarée ou traitée. Simplement ici, le fait de la déclarer donne un petit plus d'informations, quelqu'un qui lirait l'entête de cette méthode verrait qu'elle peut lancer une `ArithmeticException`, et donc prendre ses dispositions en conséquence. Enfin, pour aller jusqu'au bout de l'exemple, on supposera bien sûr qu'il existe une méthode de dessin, à proprement parler, qui est capable de dessiner un point, donc on imagine qu'il existe une méthode graphique capable de faire ce traitement,

## notes

## résumé

2m 49s



```
public static void main(String[] args) {  
    int nbEssais = 0;  
    final int MAX_ESSAIS = 2;  
    ArrayList<Double> mesures = new ArrayList<Double>();  
  
    do {  
        nbEssais++;  
  
        acquerirTemp(mesures);    // remplit le tableau  
  
        try {  
            plotTempInverse(mesures);  
        }  
        // ...  
    }  
}
```

puis bien sûr notre fameuse méthode de calcul d'inverses, qui calcule l'inverse d'un nombre, nous avons choisi ici aussi, de déclarer qu'elle lançait une `ArithmeticException`, à nouveau ça n'est pas nécessaire puisqu'il s'agit d'une `RuntimeException`, donc dans le cas où le nombre donné est nul, cette méthode va lancer une exception de type `ArithmeticException` avec un message approprié.

notes

résumé

3m 37s





```
private static void plot(double x) {  
    // fait le dessin  
}  
  
private static double inverse(double x)  
    throws ArithmeticException // PAS NECESSAIRE  
                                //RuntimeException  
{  
    if (x == 0.0) {  
        throw new ArithmeticException("Division par 0 !");  
    }  
    return 1.0/x;  
}
```

Le flot d'exécution sera le suivant, imaginez que dans le tableau de mesures, il existe une mesure nulle, nous allons donc appeler plotTemplInverse, laquelle va appeler la méthode qui calcule l'inverse d'un nombre,

notes

résumé

4m 0s



```
private static void plotTempInverse(ArrayList<Double> t)
throws ArithmeticException
{
    for(int i = 0; i < t.size(); i++) {
        try {
            plot(inverse(t.get(i)));
        } catch (ArithmeticException e) {
            System.out.println("Problème à l'indice :" + i);
            // RELANCEMENT
            throw e;
        }
    }
}
```

et qui si ce nombre est nul, va donc lancer une exception de type `ArithmeticException`,

notes

résumé

4m 13s



```
public static void main(String[] args) {
    int nbEssais = 0;
    final int MAX_ESSAIS = 2;
    ArrayList<Double> mesures = new ArrayList<Double>();

    do {
        nbEssais++;

        acquerirTemp(mesures);    // remplit le tableau

        try {
            plotTempInverse(mesures);
        }
        // ...
    }
```

cette exception va donc être capturée au niveau supérieur, donc dans `plotTempInverse`, et donc à ce moment là, le message "Problème à l'indice i" va s'afficher, puis l'exception est relancée,

notes

résumé

4m 19s



```
catch (ArithmeticException e) {  
    if (nbEssais < MAX_ESSAIS) {  
        System.out.println("Ressaisir les valeurs !");  
    } else {  
        System.out.println("Il y a déjà eu au moins "  
            + MAX_ESSAIS + " essais.");  
        System.out.println(" -> abandon");  
    }  
}  
}  
} while (nbEssais < MAX_ESSAIS);  
}
```

pour être cette fois capturée au niveau supérieur, c'est plotTemplInverse, qui cette fois a envoyé,

notes

résumé

4m 32s



**La gestion d'une exception coûte** beaucoup plus en temps de calcul qu'un simple `if.. then.. else`

- ☛ **Si l'erreur peut être traitée là où elle est découverte, il faut le faire sans passer par les exceptions**

Lancer des exceptions spécifiques est plus informatif et utile !

a lancé l'exception, du coup, on va brancher sur le bloc catch correspondant dans le programme principal, et à ce moment là on commence à comptabiliser les essais de notre utilisateur. Donc voici comment se déroulerait l'exécution d'un tel programme lorsque des lancements d'exceptions ont lieu. Voilà, pour clore sur le chapitre des exceptions, il faut savoir que la gestion des exceptions est un moyen extrêmement souple et pratique pour gérer les situations anormales,

notes

résumé

4m 40s



La gestion d'une exception coûte beaucoup plus en temps de calcul qu'un simple `if... then... else`

- Si l'erreur peut être traitée là où elle est découverte, il faut le faire sans passer par les exceptions

Lancer des exceptions spécifiques est plus informatif et utile !

mais qu'il est néanmoins beaucoup plus coûteux qu'un simple `if..then..else`. La consigne est donc que si nous avons le moyen de traiter l'erreur à l'endroit où nous la découvrons, alors il faut le faire, sans passer par la gestion des exceptions. Imaginez par exemple que dans un programme on demande à l'utilisateur de saisir des valeurs entre 0 et 100, et que l'utilisateur introduise une valeur supérieure à 100, alors nous savons gérer localement cette erreur, en signifiant à l'utilisateur qu'il doit réintroduire des valeurs, nous n'avons pas besoin d'utiliser le mécanisme des exceptions, dans ce cas là. Enfin, lorsque l'on a le choix de lancer plusieurs types d'exceptions, choisir des exceptions plus spécifiques sera généralement plus informatif et plus utile. Et ceci termine cette séquence sur les exceptions. sur les exceptions.

notes

résumé

5m 1s

