

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W16-03-exceptionscomplements-JAVA-pt3

Concepts (extraits des sous-titres générés automatiquement) :

Propres classes d'exceptions. Type d'exception. Propres classes d'exception. Constructeurs de la classe exception. Exemple concret. Biais des exceptions. Semblant utile. Classe exception. Température anormale. Situations anormales. Exemple d'utilisation. Large palette de type d'exceptions. Méthode getmessage. Seuil critique. Constructeurs suivants.



[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 3)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Java offre une très large palette de type d'exceptions possibles, qu'il est recommandé d'utiliser en étant le plus informatif possible, en utilisant le type d'exception qui cible au mieux le type de problème que vous voulez cerner ou gérer par le biais des exceptions. Néanmoins, dans certaines situations, il est peut être utile de définir soi-même ses propres classes d'exceptions, et c'est une possibilité qui est offerte par le langage.

notes

résumé

0m 1s



Il est possible de programmer **ses propres classes d'exception**

☛ sous-classe de `Exception` (ou d'une autre sous-classe d'exception existante)

Contenu minimal :

```
class MonException extends Exception
{
    public MonException() {
        super("mon message par défaut");
    }
    public MonException(String message) {
        super(message);
    }
}
```

☛ permet de préserver le comportement attendu de `getMessage()`

Pour définir ses propres classes d'exception, il suffit de les faire hériter soit de la classe exception, ou de l'une de ces sous-classes. Comme vous avez eu l'occasion de le voir dans une séquence précédente, la classe exception offre une méthode `getMessage`, qu'il est utile d'invoquer pour donner des informations sur la nature de l'exception. Le message retourné par `getMessage` est initialisé par les constructeurs de la classe exception. Pour que le message associé à vos propres classes d'exception, soit correctement initialisé, il est donc recommandé qu'une classe d'exception personnalisée qui n'ait pas d'autre contenu, contienne au minimum les deux constructeurs suivants, l'un par défaut qui initialisera le message avec une valeur par défaut, et un autre qui prendra une chaîne avec laquelle initialiser ce message. Et ceci permettra donc de préserver

notes

résumé

0m 25s




```
class TropChaudException extends Exception
{
    private double temperatureAnormale;
    private String consigne;

    public TropChaudException(double uneTemperature, String uneConsigne) {
        super("Température trop élevée");
        temperatureAnormale = uneTemperature;
        consigne = uneConsigne;
    }
    public double getTemperature() {
        return temperatureAnormale;
    }
    public String getConsigne() {
        return consigne;
    }
}
```

le comportement attendu de getMessage. Alors bien évidemment, assez souvent, si l'on définit des classes d'exception personnalisées, c'est pour mettre davantage de contenu à l'intérieur, et on est tout à fait libre à ce niveau d'ajouter autant de méthodes, d'attributs, qu'il semble souhaitable de le faire. Par exemple, il pourrait être utile de mettre à l'intérieur de la classe des attributs qui indiquent des codes d'erreurs, ou bien certaines informations sur le contenu de détection de l'exception, et ainsi de suite. Voyons ceci sur un exemple concret. Par exemple, imaginons que l'on souhaite créer une classe d'exceptions qui est capable de traiter des situations anormales par rapport à des températures, donc cette classe d'exceptions contiendrait une information

notes

résumé

1m 13s




```
class TropChaudException extends Exception
{
    private double temperatureAnormale;
    private String consigne;

    public TropChaudException(double uneTemperature, String uneConsigne) {
        super("Température trop élevée");
        temperatureAnormale = uneTemperature;
        consigne = uneConsigne;
    }
    public double getTemperature() {
        return temperatureAnormale;
    }
    public String getConsigne() {
        return consigne;
    }
}
```

indiquant quelle température anormale a été relevée, et donnerait des consignes par rapport à cette température, qu'est-ce qu'il faut faire concrètement dans le cas de telle ou telle température anormale. Vous noterez au passage qu'il est de bon temps en Java d'appeler ses propres classes d'exceptions en terminant leur nom par le terme exceptions. Nous atteignons ici les limites de l'esthétique quant au mélange anglais et français, mais nous allons vivre avec ça. La classe peut alors comporter tout contenu semblant utile

notes

résumé

1m 49s




```
class TropChaudException extends Exception
{
    private double temperatureAnormale;
    private String consigne;

    public TropChaudException(double uneTemperature, String uneConsigne) {
        super("Température trop élevée");
        temperatureAnormale = uneTemperature;
        consigne = uneConsigne;
    }
    public double getTemperature() {
        return temperatureAnormale;
    }
    public String getConsigne() {
        return consigne;
    }
}
```

pour gérer ce type d'exceptions, et typiquement ici, un constructeur capable d'initialiser la température relevée

notes

résumé

2m 13s




```
class TropChaudException extends Exception
{
    private double temperatureAnormale;
    private String consigne;

    public TropChaudException(double uneTemperature, String uneConsigne) {
        super("Température trop élevée");
        temperatureAnormale = uneTemperature;
        consigne = uneConsigne;
    }
    public double getTemperature() {
        return temperatureAnormale;
    }
    public String getConsigne() {
        return consigne;
    }
}
```

et la consigne qui y est associée. Ce constructeur devra bien évidemment invoquer aussi le constructeur de la super classe pour initialiser le message associé. Lequel pourrait tout à fait aussi être un paramètre, ici nous avons fait le choix de lui attribuer une valeur par défaut.

notes

résumé

2m 19s




```
try {
    //...
    if (temperature > TEMP_MAX){
        throw new TropChaudException(temperature,
            "Vérification de l'appareil de mesure");
    }
}

catch(TropChaudException e){
    System.out.print(e.getMessage() + " : " );
    System.out.println(e.getTemperature());
    System.out.println("Consigne -> " + e.getConsigne());
}
```

Exemple d'exécution du bloc `catch` :

Température trop élevée : 150.0

Consigne -> Vérification de l'appareil de mesure

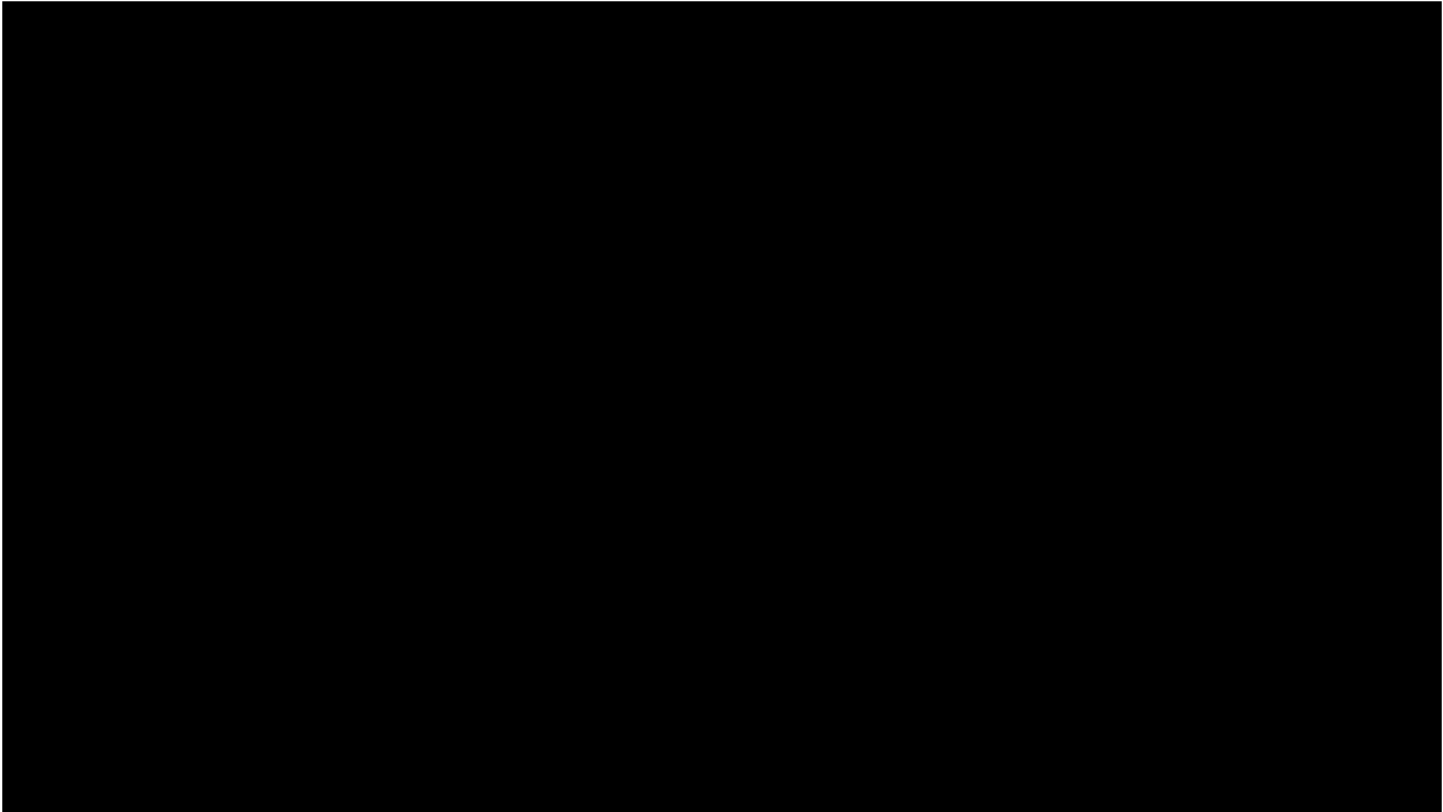
Et l'on peut ajouter toute autre méthode semblant utile, ici, nous avons simplement fait le choix d'ajouter un guetteur pour chacun des attributs, donc un pour la température et un pour la consigne. Et ici, un exemple d'utilisation, si par exemple, la température atteint un seuil critique, on peut imaginer de lancer une exception du nouveau type qu'on a défini, en l'initialisant avec la température anormale relevée, et une consigne particulière à adopter

notes

résumé

2m 37s





lorsque cette température a été rencontrée. Donc le bloc qui va intercepter cette exception va pouvoir fournir un certain nombre d'informations intéressantes sur la nature de l'exception, il va pouvoir indiquer la température anormale qui a été relevée, et il va pouvoir par exemple ici, donner des consignes sur ce qu'il faut faire par rapport à cette température là. On peut ainsi imaginer donner des messages plus informatifs, comme par exemple ici, nous avons relevé une température trop importante de 150 degrés, et la consigne dans ce cas, est d'aller vérifier votre appareil de mesure. vérifier votre appareil de mesure.

notes

résumé

3m 1s