Développement en JAVA

Comprendre la programmation objet

O.Cros - ISEN Lille

Exceptions

Exceptions

Problématique

Comment anticiper les erreurs d'un programme ?
Comment garantir la fiabilité de l'exécution d'un programme même en cas d'erreur ?

Solution

Traiter les cas d'erreurs par un la gestion d'exceptions

Exceptions

Définition

Une exception est une information renvoyée par un programme lors d'une erreur à l'exécution
En Java, on représente chaque exception par une instance d'un objet dédié

Convention

Chaque exception est représentée par une classe spécifique
Exemples : NullPointerException, DataStreamException, BadXMLFormatException, ...
5 mots-clés : try, catch, throw, throws, finally

Erreurs et exceptions

Throwable

Classe mère de toutes les exceptions

Erreurs

Les erreurs sont des problèmes liés au noyau Java, à la JVM...
Elles causent un arrêt immédiat de Java

Exceptions

Ce sont les erreurs renvoyées par les programmes tournant à l'intérieur de la machine virtuelle

RuntimeExceptions

Exceptions génériques (non interceptées)

Try/catch

Principe

On anticipe les erreurs possibles d'une méthode avec l'instruction try/catch

Syntaxe

```
1
2           try {
3                   //Instructions
4
5           }
6           catch(Exception e} {
7                   //Cas d'erreur
8           }
```

Exceptions multiples

Il est possible d'anticiper plusieurs types d'exceptions en utilisant catch !

```
1
2           try {
3                   doStuff();
4           }
5           catch(ExceptionA e) {}
6           catch(ExceptionB e) {}
```

Uncaught exception

Toute exception non interceptée renverra une RuntimeException et causera l'arrêt immédiat du programme

Finally

Mot-clé

Le mot-clé **finally** est utilisé pour exécuter un code à la fin d'une procédure où d'une exception
Même dans le cas d'une exception, finally permet de définir un code qui sera exécuté
Cela permet de clore certains processus (fermeture de fichiers, connexions, ...) même en cas d'exception

Finally

Syntaxe

```
1  try {
2          //Instructions
3  }
4  catch(Exception e) {
5          //Cas d'erreur
6  }
7  finally {
8          // Instructions de fin
9  }
```

Java Stack Trace

?

La stack trace est la trace d'exécution de Java. Elle représente un listing détaillé des causes et des sources d'une erreur lors d'une exception.
Il s'agit du premier moyen d'étude d'une erreur dans un programme Java
Chaque appel de méthode y est indiqué (numéro de ligne, fichier, ...)
Le type d'exception y est indiqué

Java Stack Trace

```
1 Exception in thread "main" java.lang.NullPointerException
2     at myproject.MyThirdClass.funcThird(Book.java:133)
3     at pack.MySecondClass.funcSecond(MySecondClass.java:34)
4     at pack.MyFirstClass.funcFirst(MyFirstClass.java:108)
5     at pack.Main.main(Main.java:14)
```

Try/catch imbriqués

```
1                try {
2                    //Some code
3                    try {
4                        //Some other stuff
5                    }
6                    catch(ExceptionA e) {}
7                }
8                catch(ExceptionB e) {}
```

Nested

Chaque try s'ajoute à la Stack Trace

Exceptions et méthodes

Throws

Le mot-clé **throws** permet de spécifier qu'une méthode est susceptible de renvoyer une exception dans l'une de ses sous-méthodes

Syntaxe

```
1         public void doStuff() throws MyException {
2         doOtherStuff();
3         }
4
5         public void doOtherStuff() {
6         try {
7             //Something
8         }
9         catch(MyException e) {}
10        }
```

Exceptions et méthodes

Throw

Le mot-clé **throw** permet de renvoyer une exception créée

Syntaxe

```
1           if(cat.age == null) {
2                   throw new CatNoAgeException();
3           }
```

Exceptions

Créer ses propres exceptions

Principe

Toute exception doit hériter de la classe Exception

Syntaxe

```
1 public class CatNoAgeException extends exception {
2         public CatNoAgeException(s) {
3         System.out.println("Age␣undefined");
4         }
5 }
```

NullPointerException

Définition

Exception sur pointeur nul

Détails

Appel d'un attribut ou d'une méthode sur un objet non instancié

NullPointerException

Exemple

```
1 Cat c;
2 c.meow(); // Renvoie une NullPointerException
```

Correction

```
1 Cat c = new Cat();
2 c.meow();
```

Quelques exemples

- ArithmeticException : erreur arithmétique (division par zéro)
- ArrayIndexOutOfBoundsException : dépassement de la taille d'un tableau
- ClassCastException : Erreur d'upcast/downcast
- NumberFormatException : Erreur de conversion String $\rightarrow$ Numeric
- StringIndexOutOfBounds : dépassement de la taille d'une String
- SecurityException : Violations des clauses de sécurité d'un programme

Classes Internes

Classes internes

Définition

Une classe interne est une classe définie au sein d'une autre classe.

Utilisation

Définir une classe dans un contexte
Alternative à l'héritage
Ensemble d'attributs ou de constantes

Classes internes

```
1          public class OuterClass {
2                  private class InnerClass {
3                          public int getValue();
4                  }
5          }
6
7          OuterClass oc = new OuterClass();
8          InnerClass ic = oc.new InnerClass();
9
10         oc.getValue();
```

Classes internes

Compilation

Deux fichiers de bytecode
OuterClass.class, OuterClass$InnerClass.class

Détails

Pas d'élément statique
Pas de main

Classes internes

Classes statiques

Une classe interne statique est indépendante de sa classe englobante
Classe embarquée

```
1 public class OuterClass {
2     public static class InnerClass {}
3 }
4
5 InnerClass ic = new InnerClass();
```

Classes internes

Scope d'une classe

public : Tous les packages
protected : package + sous-classes
private : package uniquement
Le scope d'une classe interne est similaire aux attributs

Classes anonymes

Définition

Une classe anonyme est une classe définie à la volée au sein d'une autre classe.

Utilisation

Dédiée à une utilisation particulière
Implémentation locale d'une interface (handlers, listeners, ...)

Classes anonymes

Exemple

```
1  Cat myCat = new Cat() {
2  @Override
3          void purr() {
4                  System.out.println("Rrrrrrr");
5          }
6  };
7
8  myCat.purr();
```

Classes anonymes

Méthodes

Il est possible de définir une classe en interne d'une méthode
Son scope sera limité à la méthode

Attention

Il est nécessaire de redéfinir les variables partagées comme final.

Classes anonymes

```
1  class Outer {
2          void doStuff() {
3                  final int value = 42;
4
5                  class Inner {
6                          void print() {
7                                  System.out.prinln(
8                                  ""+value);
9                          }
10                 }
11
12         }
13
14 }
```

Generics

Type safety

Définition

Protection contre les erreurs de typage
Cohérent des types

Détails

Garantir que deux objets de type non commun ne peuvent être assimilés identiques
Typage fort/faible

Generics

Principe

Chaque classe peut être paramétrée par un type particulier
Défini à l'instanciation
Ce type peut varier d'une instance à une autre
Application : collections

Exemple

```
1 ArrayList<String> names = new ArrayList<String>();
```

Generics

En appliquant de l'upcast, on peut génériser la création d'une collection

Exemple

```
1 List<String> names = new ArrayList<String>();
```

Intérêt

Génériser le type de collection
Ne pas dévoiler l'implémentation interne de la collection
Alléger le code

Syntaxe en diamant

Principe

Il est possible de génériser le type d'une collection

Exemple

```
1 List<String> names = new List<>();
```

Attention

Uniquement valable depuis Java 7

Generics

Principe

Java Generics est une solution pour traiter des données de manière **type-safe**

Génériser les données traitées par une classe

Exemples d'application

Génériser le contenu d'une collection

Ne pas pré-typer

Simplifier la gestion des ensembles

Generics sur une classe

Java Generics désigne la paramétrisation d'un type dans une classe

Exemple

```
1  public class Cat<T> {
2          private final T stuff;
3
4          public Cat(T stuffP) {
5                  this.stuff = stuffP;
6          }
7  }
8
9  Cat<String> myCat = new Cat<>();
```

Compilation

Un seul modèle de classe est créé par le compilateur
Il ne s'agit pas d'une déclinaison de classe pour chaque type

Generics sur une classe

Une classe generics peut s'appliquer à plusieurs types

Exemple

```
1 public Cat<A, B> {}
```

Generics et héritage

Principe

Il est possible de contraindre un type générique par héritage
Intérêt : Spécifier des contraintes sans forcer le type

Exemple

```
1 public class Cat<String, S extends Number> {}
```

Generics et héritage

Héritage

On peut rajouter des types génériques par héritage
On peut également spécifier des types génériques par héritage

Exemple

```
1 public class Cat<A, B> extends Feline<A> {}
2 public class Cat extends Feline<A> {}
3 public class Cat<String> extends Feline<A> {}
```

Generics et héritage

Spécification d'un type générique par héritage

Exemple

```
1 public class Feline<A> {}
2 public class Cat<String> extends Feline<String> {}
```

Generics et méthodes

Toute méthode d'une classe peut être définie comme opérant sur des Generics

```
1 class Generics<T> {
2     public T getVal() {}
3
4     void setVal (T val) {}
5 }
```

Generics et méthodes

Il est possible de définir une méthode avec un Generics, sans avoir besoin de le spécifier dans la classe.

Exemple

```
1 public class Cat {
2         public <P> List<P> doStuff() {}
3 }
```

Wildcards

Les limites de Generics

```
1 List<Integer> li;
2 List<Number> ln;
```

Problème

Deux types paramétrés liés par l'héritage n'entraînent pas de lien d'héritage avec leurs classes generics

Wildcards

Problématiques

Comment génériser le contenu d'une collection ?
Comment garantir la cohérence des types ?

Solution

Travailler avec des collections de type inconnu : les wildcards

Wildcards

Type inconnu

Le type inconnu, représenté par le symbole **?**, symbolise l'utilisation de n'importe quel type.

Exemple

```
1 Collection<?> col = new ArrayList<Cat>();
```

Wildcards

Héritage

Il est possible de contraindre les wildcards par héritage

Objectif : spécifier les types utilisés

Exemple

```
1 Collection<? extends Feline> col
2         = new ArrayList<>();
3
4 col.add(new Feline()); // OUI
5 col.add(new Cat()); // OUI
6 col.add(new Dog()); //NON
```

Conclusion

- Générisation des collections
- Définir une classe dans un contexte
- Différent entre héritage et classe interne