Gestion d'erreurs du paquetage tec

Équipe Optiplex
Yannick Levif, Joffrey Diebold, Mohamed Akdim, Jérôme Le Bot

1 Gestion d'erreurs et instanciation

Les cas limites de l'instanciation de JaugeNaturel sont rencontrés lorsque

$$VigieMax = VigieMin = ValeurouVigieMax < VigieMin$$

```
public void testExceptionCasLimites()
{
JaugeNaturel inverse = new JaugeNaturel(78, 13, 55);

JaugeNaturel egale = new JaugeNaturel(-42, -42, -42);
}
```

Nous avons pris le soin de modifier le code de l'instanciation de la JaugeNaturel egale car le résultat est incohérent dans le cas donné dans l'énoncé.

Le résultat de l'instanciation est :
Pour la Jauge inverse :
estRouge() : true
estVert() : false
estBleu() : true
L'état est donc bien incohérent.

Pour la Jauge egale :
estRouge() : true
estVert() : false
estBleu() : true
L'état est donc bien incohérent.

1.1 Lever une exception

On ne relève que la première exception car cela fonctionne comme le mécanisme des assertions : la première exception est relevée mais non capturée.

1.2 Capturer une exception

Pour vérifier que les deux exceptions sont levées, il est nécessaire de créer un bloc try...catch()... par test. Chaque bloc try catch permet de gérer l'exception déclenchée dans le try et de la gérer grâce à l'instruction du bloc catch. Dans la partie catch les variables n'ont pas de valeur car si on passe dans le bloc catch, cela signifie que l'instanciation de JaugeNaturel à échoué.

2 Paquetage tec

Les exceptions contrôlées sont des exceptions gérée par le développeur. C'est lui qui va indiquer les opérations à effectuer dans le cas ou l'exception est levée. C'est lui qui va créer la classe de l'exception, avec son constructeur et l'ensemble des messages à afficher.

Les exceptions contrôlées ne se produisent que dans des circonstances spécifiques et bien définies. Si on écrit une méthode qui lève une exception contrôlée, on doit utiliser une clause throws afin de déclarer l'exception au sein de la signature de la méthode. On parle d'exception contrôlée par le compilateur Java effectue un contrôle afin de s'assurer qu'il ya une déclaration dans les signatures des méthodes. Il provoquera une erreur de compilation si ce n'est pas le cas.

2.1 Exception contrôlée

On dispose de deux constructeur pour la classe TecInvalidException car il faut gérer deux types d'exceptions :
- le cas où elle est levée lors d'une erreur de conversion dans la méthode monterDans()
- le cas où elle est levée lors de la détection d'un état incohérent chez l'un des passagers

Dans le premier cas, l'exception sera instanciée avec un message en paramètre uniquement. Dans le second cas, l'exception sera instanciée avec un message et une instance de l'exception IllegalStateException levée par les interfaces Bus et Passager.

```
public TecInvalidException(String message) {
super(message);
}

public TecInvalidException(String message, Throwable cause) {
super(message,cause);
}
```

Pour garder une trace de l'exception de départ, l'instance de TecInvalidException a pour cause l'instance de IllegalStateException.
Le message d'erreur affiché est celui défini par l'instance de IllegalStateException (voir la documentation des constructeurs de la classe Throwable).

TecInvalidException ne doit pas hériter de la classe Error ni de la classe RuntimeEception, car c'est un exception contrôlée. Elle va hériter de Exception pour utiliser direction ses constructeurs.

```
public class TecInvalidException extends Exception {
```

Le prototype de ces deux méthodes dans les sous-types de ces interfaces n'a pas besoin d'être modifiée. Tant que le code redéfini ne lève pas cette exception contrôlée, il n'est pas nécessaire de préciser la clause de propagation.

La compilation des fichiers du répertoire src ne provoque aucune erreur. Par contre, la compilation du test de recette provoque maintenant une erreur car les méthodes monterDans() et allerArretSuivant() sont utilisées à travers les deux interfaces publique.

Pour la mise en oeuvre de l'exception contrôlée les methodes allerArretSuivant() et monterDans deviennent

```
public void allerArretSuivant() throws TecInvalidException

final public void monterDans(Transport t) throws TecInvalidException
```

2.2 Exception contrôlée dans la méthode monterDans()

```
if( !(t instanceof Bus))
throw new TecInvalidException("Echec Conversion") ;
Bus b = (Bus) t;
```

Ce bloc conditionnel permet de tester si l'exception de l'echec de conversion a bien été levé lors de l'execution.
Pour celà, on construit une classe anonyme qui est sous-type de Transport sur laquelle sera appliqué le test unitaire verifiant la présence de l'echec de la conersion.
L'operateur instanceof se charge de vérifier la concordance des types entre t et bus. Ainsi on peut detecter l'echec de conversion et déclencher l'exception.

2.3 Exception contrôlée dans la méthode allerArretSuivant()

Dans la classe TestAutobus, on rajoute la méthode testException() qui va tester la levée de l'exception contrôlée TecInvalidException.
Pour cela, on va créer une classe anonyme héritant de classe factice FauxPassager. Dans cette classe anonyme on redéfinie la méthode nouvelArret() pour qu'elle lève l'exception IllegalStateException.
Ensuite on va réécrire le code de la méthode allerArretSuivant() de manière à transformer l'exception IllegalStateException en TecInvalidException tout en gardant comme «cause» la première exception.
Pour cela on va utiliser le deuxième constructeur de la classe d'exception créée, qui va récupérer la première exception .

```
try {
Passagers.get(i).nouvelArret(this,nbArret);
} catch (IllegalStateException e){
throw new TecInvalidException("nouvelArret",e);
}

try {
this.choixPlaceMontee(b);
} catch(IllegalStateException e){
throw new TecInvalidException("choixPlaceMontee", e);
}
}
```

Enfin, il ne faut pas oublier de tester si l'exception a bien été capturée puis transfomée. Pour cela on va essayer de capturer directement TecInvalidException dans notre méthode testException().

```
try{
b.allerArretSuivant();
assert(false);
} catch(TecInvalidException e){
System.out.println("Exception Levée! Illegal->Invalid");
}
A l'exécution, l'exception a bien été levée.
```

2.4 Exceptions non contrôlées

Pour gérer les demandes incohérentes des passager, nous avons levé des exceptions IllegalStateException dans les méthodes demanderChangerEnAssis() et demanderChangerEnDebout() de la classe Autobus si le passager était déjà Assis ou Debout en fonction de la méthode.

Nous avons levé aussi des exceptions IllegalArgumentException au niveau de l'instanciation, pour eviter par exemple, une instanciation d'un passagerAbstrait avec un nom vide , et une instanciation d'un bus avec aucune place, ou que des places assises ou debout.

3 Boutez vos neurones

3.1 Java Collection Framework

Les interfaces Collection, Map, List, Set représentent les containers standards : tableau associatif, liste et ensemble. Plusieurs implémentations possibles de ces containers sont possible, d'où l'intérêt de les interfacer. C'est notamment le cas de Map qui a plusieurs méthodes possibles d'association.

On sépare le parcours de la structure de données car un même container peut être parcouru à plusieurs endroits en même temps. Les informations propre au parcours ne peuvent donc pas être contenues dans le container même.

L'iterateur n'est pas fournir par le conteneur car l'utilisateur ne sait pas comment est implémenté le conteneur, il ne peut donc pas savoir comment itérer le contenu.

Ce framework utilise le mécanisme de "type paramétré" afin de pouvoir manipuler n'importe quel type d'objet.