Les Exceptions et les Tests

Le langage java inclut un mécanisme de gestion d'erreurs appelé les exceptions. Ce mécanisme permet de propager une erreur dans la pile d'appel sans utiliser le traditionnel retour de méthode.

Une exception est un objet. C'est une instance d'une classe qui hérite de la classe **java.lang.Throwable** par exemple : **IllegalArgumentException** (voir la documentation de l'API pour les différentes exceptions existantes)

Par défaut, toutes les exceptions sont capturées par la machine virtuelle java qui affiche un message (appel à la méthode **printStackTrace**()) et stoppe l'exécution.

Table des matières

1 Première expérimentation

Ce travail utilise les tests des classes réalisant l'abstraction jauge.

Objectifs

— tester la levée d'une exception dans les cas d'erreur découverts dans le code,

Méthode de travail

— Approche TDD : L'objectif des tests du développeur est de montrer l'adéquation entre le code écrit par le développeur et la spécification fournie par le client. Le principe est d'écrire et d'exécuter ces tests en parallèle avec l'écriture du code.

— Répartir le travail : Chaque tandem recopie l'ensemble des classes jauge dans son répertoire de travail. Les tandems vont modifier la classe de test **JaugeNaturelTest** et la classes **JaugeNaturel** "implémentant" l'interface **IJauge**.

1.1 Lever une exception

D'après les tests effectués dans la classe **JaugeNaturelTest**, il y deux cas d'instanciation qui aboutissent à un état incohérent.

Dans la classe **JaugeNaturelTest**, retirer les deux tests **testLimiteVigieMaxInferieurVigieMin**() et **testMaxEgaleMin**() et ajouter le test suivant :

```
private void testCreationNonValide() {
  IJauge inverse = creerJauge(78, 13, 0);

  IJauge egale = creerJauge(-45, -45, -45);
}
```

Dans le cas d'une instanciation avec des arguments invalides, il est obligatoire d'indiquer une erreur plutôt que de ne rien faire ou d'essayer de mettre l'objet dans un état cohérent. L'instanciation doit échouer. Ici seul le mécanisme des exceptions est utilisable.

Lever l'exception **IllegalArgumentException** dans le constructeur de votre classe.

Avec le test ci-dessus, vérifier si les exceptions sont bien levées.

⇒ *Expliquer pourquoi une seule exception est levée et non les deux.*

1.2 Capturer une exception

Au lieu d'effectuer une vérification visuelle, nous allons écrire le test pour vérifier si les exceptons sont bien levée dans le code du test. Pour cela, il faut capturer l'exception dans les tests.

Le traitement à réaliser est le suivant :

- Si l'exception est capturée, c'est à dire que l'exception a été levée. Le test ne doit rien afficher puisque c'est le comportement voulu.
- Si l'exception n'est pas capturée, c'est à dire que l'exception n'a pas été levée. Le test doit afficher un message qui montre qu'une assertion a échoué.

⇒ *Pourquoi faut-il deux blocs **try/catch** pour s'assurer que l'exception est bien levée dans tous les cas d'instanciation invalide.*

Vérifier le comportement du test si aucune exception n'est levée.

⇒ *Quelle est la valeur des variables* `inverse`, `egale` *dans la partie **catch** ? Vérifiez ces valeurs grâce à une assertion.*

⇒ *Comment déclarer les variables pour les utiliser à la fois dans la clause* `try` *et la clause* `catch` *?*

Remarque : Le code écrit dans ce test ne correspond pas à une gestion d'erreur.

1.3 Exception contrôlée

Ajoutons la méthode suivante à la classe **JaugeNaturelTest** :

```
private void testExceptionControlee() {
      throw new NullPointerException("Attention");
}
```

⇒ *Donner la classe de base de cette exception ?*

Compiler et exécuter. Expliquer le résultat.

Remplaçons l'exception dans le code précedent par une instance de la classe **ClassNotFoundException**. La compilation provoque une erreur.

⇒ *Pourquoi cette exception est-elle contrôlée et pas la précédente ?*

Pour les exceptions contrôlées, le compilateur oblige à renseigner la clause **throws** qui spécifie quelles exceptions risquent d'être propagées par une méthode.

Ajouter la clause **throws ClassNotFoundException** dans le prototype de la méthode **testExceptionControllee**(). Compiler.

⇒ *Comment corriger les autres erreurs sans capturer l'exception ?*

⇒ Exécuter les tests. *Expliquer le résultat.*

2 Quatrième itération du paquetage tec : Gestion des erreurs (Exception) et remaniement du code

Objectifs

- tester la levée d'une exception dans les cas d'erreur découverts dans le code,
- étudier le masquage d'information du paquetage,
- remanier un peu le code (Ajouter des constructeurs).

Méthode de travail

- Approche TDD : L'objectif des tests du développeur est de montrer l'adéquation entre le code écrit par le développeur et la spécification fournie par le client. Le principe est d'écrire et d'exécuter ces tests en parallèle avec l'écriture du code.
- Répartir le travail : garder la même répartition du travail que l'itération précédente.

2.1 Indiquer et tester les erreurs

Hélas, la spécification est incompléte. Elle ne fournit pas les cas d'erreur à considérer.

A partir du code développé dans les itérations précédente, précisez et compléter les cas d'erreur. Par exemple :

- paramètres d'instanciation invalides,
- état du passager incohérent,
- destination < numéro arrêt
- passager stocké deux fois dans un transport,
- problème de conversion de type.

Ces cas d'erreur doivent être détectés et lever une exception capturée par le programme du client.

Sauf pour les deux méthodes **monterDans**() et **allerArretSuivant**(), il n'y a pas de précision sur la catégorie de l'exception à utiliser.

Voici la réalisation attendue :

- Les instanciations et toutes les méthodes définies par les deux interfaces privées **Bus** et **Passager** lévent forcément une exception non controllée. Vous pouvez vous servir de la classe **IllegalArgumentException** ou **IllegalStateException**
- La spécification des deux méthodes **monterDans**() et **allerArretSuivant**() indique l'exception controllée propagée. Le code de ces méthodes va donc capturer l'exception non contrôlée et léver à la place l'exception **UsagerInvalideException**.

Pour tester la levée de ces exceptions, il est peut-être nécessaire de définir d'autres classes faussaires.

Pour garder vos tests lisibles, vous pouvez écrire les tests des exceptions dans une autre classe de test `PassagerStandardExceptionTest` par exemple.

2.2 Masquage d'information du paquetage

D'après la spécification, certaines classes/interfaces/méthodes ne doivent pas être visibles par le client.

Vérifier ce qui accessible par le client c'est à dire les portées publiques.

Boostez vos neurones ;-)

Vous devriez rencontrer un problème de portée provenant de la construction des interfaces java.

⇒ *Proposer une solution pour éviter ce problème de portée ?*

Désavantage de cette solution ?

Remanier votre code pour mettre en place cette solution.

2.3 Un peu de remaniement de code

1. Ajouter un deuxième constructeur mais sans dupliquer le code d'initialisation :
   - le constructeur **Autobus**(**int nbPlace**) où le nombre de places assises et le nombre de places debout sont égaux au paramètre "nbPlace".
   - le constructeur **PassagerStandard**(**int destination**) où le nom du passager est la concaténation du nom de la classe avec la valeur du parmètre "destination".

   Expliquer l'instanciation avec ce deuxième constructeur (tracer les appels).

⇒ *Dans quel cas faudrait-il tester ce nouveau code ?*

2.4 Fin de l'itération

A la fin de cette étape, il reste au moins deux problèmes qu'il faudrat veiller à prendre en compte dans la prochaine version.

⇒ *Expliquer le problème de duplication de code qui reste dans le code*

⇒ *Expliquer le problème de duplication d'instance qui reste dans le code*

⇒ *Avez-vous noté d'autres problèmes ?*

3 Nouvelle architecture

Présenter sur un diagramme de classe la nouvelle architecture obtenue.