EtatPassager et JaugeNaturel

Nous vous fournissons deux classes :

— JaugeNaturel (déjà étudiée). Elle représente une réalisation particulière de l'abstraction `jauge`, conforme à la spécification donnée dans `JaugeNaturel.html`.

— EtatPassager . Elle représente une réalisation particulière de l'abstraction de `l'Etat d'un passager`, conforme à la spécification donnée dans `EtatPassager.html`.

Suivant le standard Java, le code de ces classes se trouvent dans un fichier portant leur nom : `JaugeNaturel.java` et `EtatPassager.java`.

Ces sources se trouvent dans les resources Moodle dédiées à cet enseignement.

Table des matières

1 TDD : Deuxième exemple

On propose d'écrire le code de la classe *EtatPassager* selon la méthode TDD utilisée dans le précédent TP (JaugeNaturel).

Les tests du développeur pour la classe *EtatPassager* sont décrits par la documentation de la classe *EtatPassagerTest.html*.

Le travail consiste à écrire ces tests et à les faire passer au code fourni. Pour s'assurer que le code (syntaxiquement juste) correspond bien à la spécification (aux fonctionnalités) ;-).

1.1 Ecriture du premier test.

Les premiers tests à écrire concernent toujours l'état après instanciation.

D'après la documentation, le premier test à écrire est : *testExterieur()* ; test sur une instance avec un état dehors pour la classe *EtatPassager*. Pour le test, la valeur est donnée par *EtatPassager.Etat.DEHORS*. L'énumération *Etat* fixe les valeurs possibles de l'état d'un passager.

Sur l'état de cette instance, nous avons les trois méthodes suivantes :

— *estAssis()* est faux,
— *estDebout()* est faux.
— *estExterieur* est vraie,

Mise en place de la classe de tests

1. définir la classe de tests dans un fichier source,
2. définir la première méthode de test de cette classe (elle est possible de générer une partie),
3. écrire le code de votre test dans cette méthode en utilisant le mécanisme des assertions inclus dans java.

1.2 Réaliser la suite des tests.

Réaliser itérativement le reste des tests dans l'ordre indiqué par la docucumentation. Suivez la même procédure :

— Ecrire un test, vérifier l'echec du test.
— Modifier le code pour faire passer ce test.
— Refactoriser : Simplifier les expressions booléennes dans le code de la classe *EtatPassager*.

2 Ajouter de nouvelles réalisations à une abstraction

Dans ce qui suit, on propose de répartir le travail. Un tandem prend en charge les réalisation autours de `jauge` et l'autre les réalisations autours de `état de passager`.

Nous vous demandons d'écrire d'autres réalisations de l'abstraction `jauge` et de l'abstraction d'un `état de passager`. Evidemment ces réalisations doivent passer les tests écrits dans l'exercice précédent. Les tests du développeur sont des tests de non régression.

Voici les classes "un peu bizarre" :

EtatPassagerChaine Les états d'un passager sont représentés par les trois chaînes de caractères : "debout", "dehors", "assis".

EtatPassagerMonter On ne s'interesse qu'aux passagers qui sont déjà monté.

JaugeReel Les variables *min, max, valeur* sont de type réel ("float") divisées par 1000.

JaugeNegatif Les variables *min, max, valeur* sont de type entier ("long") mais les valeurs stockées ont un signe inversé par rapport aux valeurs passées en paramètres.

JaugeDistance L'objet stocke seulement deux variables : la distance entre la position courante et les deux vigies.

Mise en place des nouvelles classes par copier/coller :-(.

Evidemment, pour créer ces nouvelles classes, vous allez vous servir des fichiers existants.

Pour les classes EtatPassagerChaine,EtatPassagerMonter, JaugeNegatif, JaugeReel, JaugeDistance :

1. Recopier le fichier `JaugeNaturel.java` ou `EtatPassager.java`.
2. Changer son nom pour qu'il correspond au nom de la nouvelle classe.
3. Dans ce nouveau fichier changer le nom de la classe et du constructeur. **Ne modifier pas encore le reste du code.**
4. Inspecter le code et corriger les erreurs.

Même manière pour les classes de tests :

1. Recopier le fichier `JaugeNaturelTest.java` ou `EtatPassagerTest.java`.
2. Changer son nom.
3. Dans ce nouveau fichier remplacez toutes les occurences JaugeNaturel (ou EtatPassager) par le nom de votre classe à tester.
4. Compiler et exécuter.

Après ces modifications tous les tests doivent être valide car en fin de compte seul le nom des classes a changé.

Le développement des classes

Il reste à développer le code (variables, constructeurs, méthodes) des classes EtatPassagerChaine, EtatPassagerMonter, JaugeNegatif, JaugeReel, JaugeDistance.

Pour cela, vous avez la manière brutale "modifier entièrement le code et faire passer tous les tests". En gros ça marche ou ça casse suivant la complexité des modifications.

Pour éviter les surprise, vous pouvez adopter une démarche itérative test par test : prendre un test, modifier le code (pour faire passer le test), compiler, exécuter, prendre le test suivant.

Il suffit de mettre en commentaire les méthodes de test et de les décommenter au fur et à mesure.

Remarque : La manière de définir les nouveaux constructeurs va avoir une influence sur l'instanciation de la classe dans les tests. Essayer de minimiser ces modifications.

3 Une seule classe de tests

Il n'est pas très intelligent d'avoir dupliquer le code des tests même si c'était une solution rapide et simple (en plus c'était demandé :-))

Donner deux problèmes posés par ce copier/coller ?

Il faudrait pouvoir appliquer la même classe de tests aux instances des classes réalisant la même abstraction sans avoir à modifier le code de la classe de tests (le code existant).

En terme d'objet, il faut pouvoir substituer, dans le code de la classe de test, les instances des différentes classe à tester.

Remarque : Le code d'une classe de test dépend de la classe à tester de deux manières : type des variables (nom de la classe) et instanciation (nom du constructeur et parfois la liste des paramètre d'instanciation. Voyons comment remanier le code pour le rendre moins dépendant.

3.1 Relation de Type/Sous-Type

Dans un langage typé comme Java, une classe correspond à un type (et à l'implémentation de ce type). La définition d'une nouvelle classe entraine la définition d'un nouveau type.

Pour permettre la substitution d'objets, il faut que le type d'une variable accepte des instances d'une autre classe. C'est le polymorphisme.

Pour cela, il faut créer une relation de type/sous-type entre les classes. Si la classe B est un sous-type de la classe A, la classe B inclut forcément les opérations définies dans A. Donc, une variable de type A va pouvoir accepter des instances de la classe B.

Pour construire cette relation de type/sous-type, nous allons utiliser une interface java.

Il faut construire une interface nommée IJauge pour l'abstraction jauge et une interface nommée IEtatPassager pour l'abstraction état d'un passager.

L'interface déclare les opérations communes à toutes les instances des classes réalisant cette abstraction. Elle correspond à un type abstrait de donnée.

1. Définir l'interface IJauge dans un fichier `IJauge.java` et l'interface IEtatPassager) dans un fichier `IEtatPassager.java`. inspecter le code de l'interface, corriger les érreurs.
2. Donner les méthodes déclarées dans l'interface IJauge et celles déclarées dans l'interface IEtatPassager.
3. Modifier le code des classes JaugeNaturel, JaugeNegatif, JaugeReel pour indiquer qu'elles sont sous-type de Jauge.

   Modifier le code classes EtatPassagerChaine et EtatPassager pour qu'elles sont sous-type de IEtatPassager.

4. Recopier le fichier d'une des classes de test, renommer le fichier et la classe en `IJaugeTest` ou `IEtatPassagerTest`. Que peut-on modifier dans cette classe de test ? Compiler et exécuter les tests.

Après ce remaniement, il reste encore des lignes à modifier pour effectuer la substitution d'objets.

3.2 Factoriser l'instanciation

L'instanciation lie un objet à une classe. Un objet est instance d'une classe et d'une seule (impossible à un objet de changer de classe). Pour cette raison, il est difficile d'avoir une instanciation indépendante du nom de la classe et de la liste de paramètre d'initialisation.

Mais, il est possible de limiter le nombre de lignes à changer en factorisation l'instanciation.

Dans la classe IJaugeTest, ajoutons une méthode creerJauge(). Son objectif est de factoriser le code d'instanciation. Elle doit donc retourner une instance.

Voici les questions à se poser :

— Quel code va-t-elle contenir (factoriser) ?

— Est-elle privée ou publique ?

— Quel est son type de retour ?

— Quel est sa liste de paramètre ?

La liste de paramètre d'instanciation pose parfois un problème. La manière d'initialiser une instance peut dépendre de la réalisation de la classe. C'est le cas dans la classe IEtatPassagerTest suivant la réalisation le type du paramètre est différent. Heureusement, il n'y a que trois manière d'instancier cette classe. Nous allons abstraite la manière d'instancier en définissant trois méthodes creerAssis(), creerDehors(), creerDebout().

Voici les questions à se poser pour chaque méthode :

— Quel code va-t-elle contenir (factoriser) ?

— Est-elle privées ou publiques ?

— Quel est son type de retour de ?

— Quel est sa liste de paramètre ?

Modifier le code des classes de tests. Compiler et exécuter.

Hélas, nous n'avons pas atteint notre objectif. Il reste encore du code à modfier dans la classe de tests pour tester toutes les instances.

Remarque : Un peu plus tard, nous verrons une technique utilisant l'héritage.