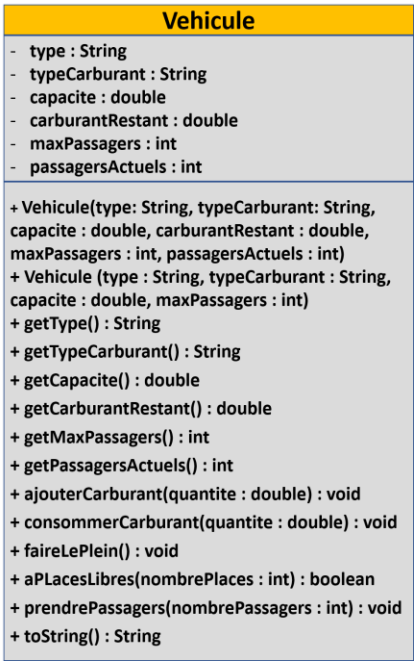
TD 9 Les tests unitaires

Jusqu'au présent, notre seule façon de vérifier qu'une méthode a été bien écrite était de faire des affichages intermédiaires ou d'appeler la méthode en question depuis d'une méthode `public static void main`, dans une classe de type application. Aujourd'hui notre but sera de revoir des éléments du TD4 et d'écrire des tests unitaires pour quelques-unes des méthodes qu'on considère.

Dans ce TP nous avions une classe Vehicule, dont le diagramme de classe est donné ci-dessous :



Dans ce diagramme de classe, pour la classe Vehicule on a spécifié une méthode `ajouterCarburant(double quantite)` et une méthode `consommerCarburant(double quantite)`. Nos premiers tests vont se concentrer sur ces deux méthodes là. Nous allons utiliser la syntaxe de JUnit4.

Attention : le but de cet exercice n'est pas de trouver et d'implémenter de façon exhaustive tout test qui serait pertinent à la classe `Vehicule`. Le but est plutôt de choisir quelques tests spécifiques pour s'entrainer à leur écriture.

Exercice 1

Nous allons premièrement écrire quelques tests pour la classe `Vehicule`. Ces tests seront écrits dans leurs propre classe, qu'on appelle `TestsVehicule`.

- Rappel : est-ce qu'une classe de tests peut avoir des attributs ?

- Rappel : qu'est-ce qu'une méthode `setUp` ? Comment doivent-on écrire la méthode de `setUp` ?
- Rappel : et la méthode `tearDown` ?
- Rappel : supposons que dans une classe de test, on a plusieurs méthodes de test, ainsi qu'une méthode de `setUp` et une de `tearDown`. Quand est-ce qu'on exécute les méthodes de `setUp` et de `tearDown` durant l'exécution des tests compris dans une classe de test ?
- La classe `TestsVehicule` aura deux attributs de type `Vehicule`, `voitureVide` et `voiturePleine`. Dans la méthode de setUp, les deux attributs seront instanciés à :
    - `voitureVide` : un véhicule de type "`Voiture`", utilisant du "`Diesel`", avec une capacité totale de 55l mais un reservoir vide (0l). La voiture aura une capacité de 5 passagers, mais il n'y aura que 2 actuellement dedans.
    - `voiturePleine` : un véhicule de type "`Voiture`", utilisant du "`Diesel`", avec une capacité totale de 55l et un reservoir plein. La voiture aura une capacité de 5 passagers, mais il n'y aura que 2 actuellement dedans.

  Ecrivez la méthode `setUp`.

- La méthode `tearDown` remettra les deux attributs à `null`. Ecrivez cette méthode.

Exercice 2

La méthode `ajouterCarburant(double quantite)` de la classe `Vehicule` prend en entrée une valeur `double`, qui est ajoutée à la valeur actuelle de la variable `carburantRestant`; toutefois la valeur totale du carburant restant ne doit jamais dépasser la capacité maximale du réservoir de la voiture, donnée par l'attribut `capacite`.

La méthode `consommerCarburant(double quantite)` de la classe `Vehicule` sert à diminuer la quantité de carburant stockée par `carburantRestant` par la valeur donnée en entrée de la méthode. Cependant, la quantité de carburant restant ne peut jamais baisser sous 0.

Pour les questions 1 et 2 le but ne sera pas de modifier le fonctionnement de ces méthodes sinon de tester leur fonctionnement.

1. Nous voulons tester la méthode `ajouterCarburant(double quantite)` en utilisant l'attribut `vehiculeVide` de la classe `TestsVehicule`. Pour chaque test ci-dessous, mentionnez : (a) le nom de la méthode de test que vous allez utiliser ; (b) quel est l'effet attendu de l'action que vous allez tester `vehiculeVide` ? ; (c) comment allez-vous tester que l'action a eu l'effet que vous avez prévu ? Finalement, écrivez chaque test.

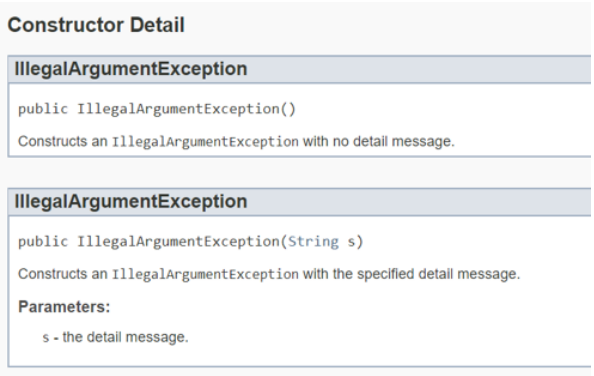
    - Test 1 : ajouter une valeur normale de carburant (disons 20l)
    - Test 2 : ajouter le maximum possible de carburant dans le vehicule (55l)
    - Test 3 : ajouter plus que le maximum de carburant (100l)

2. Nous voulons ensuite tester la méthode `consommerCarburant(double quantite)` sur l'attribut `vehiculePlein`. Utilisez la même méthodologie que dans l'exercice précédent.

- Test 1 : consommer une valeur normale de carburant (disons 20l)
- Test 2 : consommer tout le carburant dans la voiture (55l)
- Test 3 : consommer une quantité trop importante de carburant (100l)

3. Pour l'instant le fonctionnement des deux méthodes `ajouterCarburant` et `consommerCarburant` n'utilise pas d'exceptions lorsque la valeur de l'attribut `carburantRestant` monte ou baisse trop. Dans cet exercice nous allons modifier le fonctionnement de la méthode `ajouterCarburant`. Si maintenant le paramètre donné en entrée de cette méthode est négatif, alors on lève une nouvelle exception de type IllegalArgumentException (qui est un sous-type de `RuntimeException` et donc une exception non-contrôlée). Si une telle exception est levée, on ne fait aucune modification à la quantité de carburant restant.

   La documentation suivante concerne deux constructeurs de la classe `IllegalArgumentException`. Ecrivez la nouvelle méthode `ajouterCarburant`.



Constructor Detail

IllegalArgumentException

`public IllegalArgumentException()`

Constructs an `IllegalArgumentException` with no detail message.

IllegalArgumentException

`public IllegalArgumentException(String s)`

Constructs an `IllegalArgumentException` with the specified detail message.

Parameters:

`s` - the detail message.

4. Nous voulons ensuite tester si la méthode `ajouterCarburant` lève bien l'exception demandée. Ecrivez un test qui fait cela dans la classe `TestsVehicule`.

Exercice 3

Nous allons prochainement nous concentrer sur la classe `Voiture`, qui hérite de `Vehicule`. Dans la spécification mise dans le `TD4`, une voiture avait la place pour cinq personnes (y compris le chauffeur). Dans ce TD nous allons modifier ce fonctionnement. La classe Voiture aura désormais un attribut taille de type char qui stocke la taille de la voiture ('`S`', '`M`' ou '`L`'). Cet attribut n'affectera que le nombre maximal de passagers qu'elle peut accueillir : une voiture petite accueille un maximum de 2 passagers (y compris le chauffeur), une voiture de taille moyenne en accueille jusqu'à 5 et une voiture large accueille jusqu'à 8 passagers (y compris le chauffeur).

La méthode `prendrePassagers(int nombrePassagers)` de la classe `Vehicule` a la prochaine spécification : pour un nombre de passagers donné en paramètre la méthode vérifie s'il restent

nombrePassagers places dans la voiture. S'il y a assez de place, les passagers sont pris ; sinon, aucun passager n'est pris.

1. Nous aurons un premier constructeur de la classe Voiture dont la signature sera : `Voiture(char taille, String typeCarburant, double capacite, double carburantRestant, int passagersActuels)`. Celui-ci instancie les attributs d'une voiture aux valeurs en entrée à l'exception de deux valeurs : le type sera mis par défaut à « `Voiture` » tandis que le nombre maximal de passagers sera mis en fonction de la taille en entrée aux valeurs indiquées ci-dessus.

   Ecrivez ce constructeur.

2. Nous voulons écrire une classe de tests pour la classe Voiture, qui va tester le fonctionnement de la méthode héritée `prendrePassagers`. La nouvelle classe de tests s'appelle `TestsVoiture`. Cette classe aura trois voitures déclarées en tant qu'attributs : `voiturePetite`, qui sera de taille '`S`', `voitureMoyenne` de taille '`M`' et `voitureLarge` de taille '`L`'. Les trois voitures auront le type de carburant mis à «Essence ». Elles auront une capacité de 55l, avec un carburant restant de 55l et un seul passager à bord (le chauffeur).

   Ecrivez des méthodes de `setUp` et de `tearDown` dans la classe `TestsVoiture`. La méthode de `setUp` devra instancier les trois attributs de la classe. La méthode `tearDown` les remettra à `null`.

3. Nous allons premièrement écrire des tests pour une voiture de taille petite qui utilise la méthode `prendrePassagers`. Dans chaque méthode de test nous allons utiliser la méthode `prendrePassagers` pour ajouter plus de passagers à la voiture en question, puis nous allons vérifier le nombre de passagers dans la voiture. Il faut tout d'abord savoir comment nous pouvons vérifier ce nombre. Quelle méthode peut-on utiliser ?

4. Pour chacun des tests suivants, suivez la méthode de l'exercice précédent pour écrire les deux tests unitaires suivants :
   - Un test qui utilise la méthode `prendrePassagers(int nombrePassagers)` pour prendre 1 passager
   - Un test qui utilise la même méthode pour prendre 9 passagers.

5. Pour une voiture de taille moyenne nous devons écrire un troisième type de test. Lequel ? Ecrivez-le.

6. Décrivez (sans écrire), quels seraient les tests équivalents pour une voiture de taille large.

Exercice 4

La méthode `aPlacesLibres(int nombrePlaces)` retourne une valeur de type `boolean` : `true` si on a encore `nombrePlaces` places dans la voiture, false sinon.

- En suivant la méthodologie des exercices précédents écrivez les tests suivants :
    - Un test pour une voiture petite dans lequel nous vérifions qu'elle a encore de la place pour prendre 1 passager.
    - Un test pour une voiture de taille moyenne qui va vérifier si elle a encore 5 places disponibles.

- Finalement nous allons vérifier comment marche l'interaction entre les méthodes `prendrePassagers` et `aPlacesLibres`. Notamment nous allons commencer avec une voiture qui a encore de la place, on va ajouter un nombre de passagers tel que la voiture se remplisse et puis nous allons vérifier si elle a encore de la place. Nous allons faire cela avec l'attribut `voitureLarge` de la classe `TestsVoiture`. Ecrivez les tests suivants :
    a. On ajoute 5 passagers aux passagers actuels dans la voiture et on vérifie si on a encore de la place pour un passager de plus
    b. On ajoute 7 passagers et on vérifie si on a encore de la place pour un passager.
    c. On ajoute 10 passagers et puis on teste si on a encore de la place.