

Location de voitures

Classe de test

```
package be.ethnic.cars;

import static java.util.stream.Collectors.toSet;
import static org.junit.Assert.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.lang.reflect.Modifier;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Optional;
import java.util.Set;
import java.util.stream.IntStream;
import java.util.stream.Stream;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;

@SuppressWarnings("static-method")
public class RentalTest {
    @Test
    public void shouldCreateCarWhithModelAndYear() {
        Car car = new Car("ford mustang", 2014);
        assertEquals("ford mustang", car.getModel());
    }
}
```

```

@Test
public void shouldGetErrorWhenCreatingCarWhithoutModel() {
    assertThrows(NullPointerException.class, new Executable() {
        public void execute() throws Throwable {
            new Car(null, 2014);
        }
    });
}

```

```

@Test
public void shouldGetErrorWhenRemovingCarOnEmptyRental() {
    final CarRental rental = new CarRental();
    assertThrows(IllegalStateException.class, new Executable() {
        public void execute() throws Throwable {
            rental.remove(new Car("ford mustang", 2013));
        }
    });
}

```

```

@Test
public void shouldGetErrorWhenAddingNonExistentCarToRental() {
    final CarRental rental = new CarRental();
    assertThrows(NullPointerException.class, new Executable() {
        public void execute() throws Throwable {
            rental.add((Car) null);
        }
    });
}

```

```

@Test
public void shouldAddLotsOfNewCarsToRental() {
    CarRental rental = new CarRental();
    for (int i=0; i<10000; i++)
    {
        rental.add(new Car("foo car", i));
    }
}

```

```

@Test
public void shouldRemoveCarOfRental() {

```

```

    CarRental rental = new CarRental();
    rental.add(new Car("ford mustang", 2013));
    rental.remove(new Car("ford mustang", 2013));
    assertEquals("", rental.toString());
}

@Test
public void shouldConvertRentalToText() {
    CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2001));
    rental.add(new Car("ford mustang", 2006));
    assertEquals("audi tt 2001\nford mustang 2006", rental.toString());
}

@Test
public void shouldFindCarByYearInRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2012));
    rental.add(new Car("ford mustang", 2014));
    List<Car> all = rental.findAllByYear(2014);
    assertTrue(all.contains(new Car("ford mustang", 2014)));
}

@Test
public void shouldNotFindAnyCarWhenSearchingNonExistantYear() {
    CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2015));
    rental.add(new Car("ford mustang", 2013));
    List<Car> toSell = rental.findAllByYear(2014);
    assertTrue(toSell.isEmpty());
}

@Test
public void shouldVerifyEqualityOfIdenticalCamels() {
    Camel camel = new Camel(2014);
    assertEquals(camel, new Camel(2014));
}

@Test
public void shouldConvertCamelToText() {

```

```

    Camel camel = new Camel(2014);
    assertEquals("camel 2014", camel.toString());
}

@Test
public void shouldAddAndRemoveCarsOrCamelsOfRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("ford mustang", 2014));
    rental.add(new Camel(2010));
    rental.remove(new Camel(2010));
    rental.remove(new Car("ford mustang", 2014));
}

@Test
public void shouldGetErrorWhenRemovingCamelOnEmptyRental() {
    final CarRental rental = new CarRental();
    assertThrows(IllegalStateException.class, new Executable() {
        public void execute() throws Throwable {
            rental.remove(new Camel(2010));
        }
    });
}

@Test
public void shouldFindCarsAndCamelsByYearInRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("ford mustang", 2010));
    rental.add(new Camel(2010));
    final List<Car> list = rental.findAllByYear(2010);
    assertAll(
        new Executable() {
            public void execute() throws Throwable {
                assertTrue(list.contains(new Car("ford mustang", 2010)));
            },
        new Executable() {
            public void execute() throws Throwable {
                assertTrue(list.contains(new Camel(2010)));
            }
        }
    );
}

```

```

    );
}

@Test
public void shouldGetErrorWhenSearchingNonExistentCarInRental() {
    final CarRental rental = new CarRental();
    assertThrows(NullPointerException.class, new Executable() {
        public void execute() throws Throwable {
            rental.findACarByModel(null);
        }
    });
}

@Test
public void shouldComputeInsuranceCostOfRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2001));
    rental.add(new Car("ford mustang", 2009));
    rental.add(new Camel(2013));
    rental.add(new Camel(2010));
    assertEquals(rental.insuranceCostAt(2017), 1800);
}

@Test
public void shouldGetErrorIfWhenAskingAnInsuranceCostWithADateOlderThanTheCarCreation() {
    final CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2001));
    assertThrows(IllegalArgumentException.class, new Executable() {
        public void execute() throws Throwable {
            rental.insuranceCostAt(2000);
        }
    });
}

@Test
public void shouldGetErrorIfWhenAskingAnInsuranceCostWithADateOlderThanTheCamelBirth() {
    final CarRental rental = new CarRental();

```

```

        rental.add(new Camel(2013));
        assertThrows(IllegalArgumentException.class, new Executable() {
            public void execute() throws Throwable {
                rental.insuranceCostAt(2012);
            }
        });
    }

    @Test
    public void shouldNotFindACarByNonExistantModelInRental() {
        CarRental rental = new CarRental();
        rental.add(new Car("renault alpine", 1992));
        rental.add(new Camel(1992));
        assertNotNull(rental.findACarByModel("ford mustang"));
    }
}

```

Enoncé

Le but de cet exercice est de créer un ensemble de classes permettant de gérer une agence de location de voitures.

Les tests JUnit 5 de cet exercice sont [RentalTest.java](#).

1. Écrire une classe `Car` dans le package `fr.uml.v.rental`, correspondant à un véhicule qui pourra être loué. Un véhicule est décrit par un modèle (une chaîne de caractères) ainsi qu'une année de fabrication.

Par exemple, une Ford Mustang sera créée de cette façon:

```
Car mustang = new Car("ford mustang", 2014)
```

2. Modifier la classe `Car` pour que le code suivant affiche le texte "ford mustang 2014".

```
System.out.println(mustang);
```

3. Créer une classe `CarRental` (toujours dans le package `fr.uml.v.rental`) qui stocke l'ensemble des véhicules qui peuvent être loués dans une liste. La classe `CarRental` doit posséder une méthode `add` qui permet d'ajouter un véhicule dans la liste.

Faire en sorte que la liste ne puisse pas contenir `null` en empêchant d'ajouter des voitures `null`.

Pour tester si une valeur est `null`, vous utiliserez la méthode `Objects.requireNonNull()`.

4. Écrire une méthode `remove` qui permet de retirer un véhicule de la liste.
Que faire si le véhicule n'a pas été préalablement ajouté ?
Vérifier que le test `carRentalAddRemove` est valide. Sinon, expliquez quel est le problème et corrigez-le.
5. Pour visualiser une instance de la classe `CarRental`, on devra afficher l'ensemble des véhicules de la liste, séparés par des retours à la ligne (mais sans retour à la ligne final !).
Écrire le code correspondant en utilisant la classe `StringBuilder`.
6. Rappeler à quoi sert l'interface `Stream` en Java, comment obtenir un stream à partir d'une liste, comment marchent les méthodes `filter`, `map` et `collect` et enfin comment peut-on utiliser le collecteur `Collectors.joining()` pour simplifier l'implantation de la méthode d'affichage que vous venez d'écrire.
7. On cherche à connaître toutes les voitures enregistrées dans le `CarRental` ayant la même année de fabrication.
Écrire une méthode `findAllByYear(int year)` qui prend en paramètre une année et renvoie une liste des voitures ayant l'année de fabrication demandée.
Que doit-on faire si il n'y a pas de voiture correspondant à l'année demandée.
8. L'application que vous développez doit aussi être vendue en Egypte où malheureusement, il n'est pas rare de manquer d'essence. Pour éviter de mettre la clé sous la porte, les loueurs de voitures ont trouvé une solution de secours en louant aussi des chameaux.
Modifier le code de votre application pour permettre de louer non plus uniquement des véhicules mais aussi des chameaux, sachant qu'un chameau possède juste une date de naissance et que son affichage est "camel" suivi d'un espace et de sa date de naissance.
Par exemple, le code suivant devra fonctionner

```
var rental = new CarRental();
rental.add(new Car("ford mustang", 2014));
rental.add(new Camel(2010));
```

La méthode `findAllByYear` devra renvoyer une liste pouvant être constituée de véhicules et de chameaux.

En terme de design, faire en sorte que si l'on doit ajouter plus tard une classe `SpaceShuttle` pour gérer les navettes spatiales, alors on n'aura pas à modifier la classe `CarRental`.

9. Comment faire pour que la date de fabrication d'un véhicule et de naissance d'un chameau correspondent à un seul et même champ partagé par les classes `Car` et `Camel`?
10. Finalement, est-il vraiment nécessaire d'utiliser une interface?
11. Les véhicules à louer doivent être assurés. Une voiture de moins de 10 ans coûte 200 euros à assurer et sinon, l'assurance est de 500 euros. Pour un chameau, le prix de l'assurance est proportionnel à son âge, qu'il faut multiplier par 100 euros..
Écrire dans la classe `CarRental`, une méthode `insuranceCostAt` qui permet de calculer le coût total pour assurer tous les véhicules pour une année donnée (passée en paramètre).
Attention, l'hypothétique introduction de la classe `SpaceShuttle` dont le prix d'assurance

sera calculé en fonction du nombre de voyages effectués devra aussi se faire sans modifier la classe `CarRental`.

Note: pensez à gérer le cas où la date est plus ancienne l'année de création du véhicule ou de naissance des chameaux.

12. Enfin, écrire dans la classe `CarRental`, une méthode `findACarByModel` qui permet de trouver une voiture à partir de son modèle passé en paramètre.

Expliquer de plus pourquoi cette méthode doit retourner un objet de type [Optional](#).

Revision #1

Created 29 October 2019 21:17:50 by Admin

Updated 29 October 2019 21:19:19 by Admin