TP - Agence de location

Récupérez l'archive sur le portail pour en utiliser les codes source et les tests fournis.

On s'inspire du sujet du TD sur les agences de location de voiture (toujours disponible sur le portail) en y apportant les modifications et extensions suivantes :

- on remplace le type `Car` par la classe `Vehicle` dont voici le diagramme UML

rental::Vehicle
- brand : String - model : String - productionYear : int - dailyRentalPrice : float
+Vehicle(brand : String, model : String, productionYear : int, dailyrentalPrice : float) + getBrand() : String + getModel(): String + getProductionYear() : int + getDailyRentalprice() : float + equals(o : Object) : boolean + toString() : String

- l'interface `CarFilter` devient `VehicleFilter` et est adaptée pour gérer des objets `Vehicle` et non plus des objets `Car`.

- le filtre intersection de la question 10 du TD peut maintenant regrouper un **nombre quelconque** de filtres (et plus seulement deux) qui sont ajoutés via la méthode `addFilter`. Son diagramme UML est le suivant :

rental::AndFilter
- theFilters : List<VehicleFilter>
+AndFilter() +addFilter(f : Filter) +accept(v : Vehicle)

- la méthode `select` de la classe `RentalAgency` devient :

  ```
  public List<Vehicle> select(VehicleFilter filter)
  ```

  dont le résultat est la liste des véhicules qui sont acceptés par le filtre `filter` passé en paramètre.

- on ajoute à la classe `RentalAgency` une méthode `displaySelection` qui prend en paramètre un filtre et affiche les véhicules acceptés par ce filtre. Vous réutiliserez bien sûr la méthode `select`.

- on ajoute à la classe `RentalAgency` la gestion des locations des véhicules. Un client ne peut louer qu'un véhicule à la fois.

  On pourra utiliser et si nécessaire compléter la classe `Client` fournie où les clients sont modélisés par un attribut représentant l'âge et un autre correspondant à leur nom qui sera une chaîne de caractères. On supposera que les noms sont uniques, il n'y a pas d'homonyme.

  On décide de gérer ces locations par une table (`java.util.Map`) qui associe les clients (clés) avec le véhicule (valeur) qu'ils ont loué. Un client n'est présent dans cette table que si il est en train de louer un véhicule. Il en donc « supprimé » donc dès qu'il rend un véhicule.

  On complète la classe `RentalAgency` avec les méthodes suivantes :

  - `public void addVehicle(Vehicle v)` permet d'ajouter une véhicule à l'agence
  - `public float rentVehicle(Client client, Vehicle v)`
    `throws UnknownVehicleException, IllegalStateException`

    permet au client `client` de louer le véhicule `v`. Le résultat est le prix de location.

    L'exception `UnknownVehicleException` est levée si le véhicule n'existe pas dans l'agence et `IllegalStateException` est levée s'il est déjà loué ou que le client loue déjà un autre véhicule.
  - `public boolean hasRentedAVehicle(Client client)` renvoie `true` si et seulement si `client` est un client qui loue actuellement un véhicule et donc `false` sinon.

- `public boolean isRented(Vehicle v)` renvoie `true` si et seulement si le véhicule est actuellement loué, `false` sinon.
- `public void returnVehicle(Client client)` : le client `client` rend le véhicule qu'il a loué. Il ne se passe rien si il n'avait pas loué de véhicule.
- `public Collection<Vehicle> allRentedVehicles()` renvoie la collection des véhicules de l'agence qui sont actuellement loués.

Q 1 . Créez la classe `UnknownVehicleException` puis complétez le code des classes `RentalAgency` et `AndFilter` fournies en tenant compte des compléments au cahier des charges mentionnés ci-dessus.

N'oubliez pas les tests. Lorsque vous complèterez la class `RentalAgencyTest` fournie, vous utiliserez `@Before` dont vous trouverez un exemple d'utilisation dans la classe `VehicleTest` fournie.

`@Before` est présentée dans le document sur les tests présent dans la zone Documents du portail.

Q 2 . Comme dans le TD, créer un `main` grâce auquel vous effectuerez quelques expérimentations en créant quelques objets véhicules que vous ajouterez à une agence et en affichant les résultats de sélections par des filtres.

Créez aussi des clients et faites leur louer et rendre des véhicules et écrivez du code qui utilise les différentes méthodes de `RentalAgency` que vous avez définies.

Q 3 . Le code de la classe `RentalAgencyTest` fournie dispose de la méthode

Réalisation

```
@Test
public void twoClientObjectsWithSameNameCorrespondsToSameClient()
                                throws IllegalStateException, UnknownVehicleException {
    RentalAgency agency = new RentalAgency();
    Vehicle v = new Vehicle("Vroum", "Vroum", 2000, 100);
    agency.addVehicle(v);
    Client client1 = new Client("Tim Oleon", 25);
    assertEquals(100, agency.rentVehicle(client1, v),0.0001);
    assertTrue(agency.hasRentedAVehicle(client1));
    // client2 corresponds to same client than client1 since names are equals
    Client client2 = new Client("Tim Oleon", 25);
    assertTrue(agency.hasRentedAVehicle(client2));
}
```

1. Votre code passe-t-il ce test avec succès ?
   Ce devrait être le cas. Si ce n'est pas le cas, quelle en est la raison ?
2. Faites les corrections nécessaires pour obtenir le résultat attendu. A priori, ces corrections ne doivent pas consister à modifier les méthodes `hasRentedVehicle`, ni `rentVehicle` (si vous ne trouvez pas pourquoi demandez à votre enseignant !).

Héritage

Q 4 . Créez une classe `Car` qui hérite de `Vehicle`. Une voiture a comme propriété additionnelle le nombre de passagers qu'elle peut accueillir. La classe `Car` dispose de l'accesseur associé. La méthode `toString` de cette classe reprend les mêmes informations que `Vehicle` complétées du nombre de passagers.

Q 5 . Créez une classe `Motorbike` qui hérite de `Vehicle`. Une moto a comme propriété additionnelle la cylindrée (exprimée en cm^3). La classe `Motorbike` dispose de l'accesseur associé et sa méthode `toString` reprend les mêmes informations de `Vehicle` complétées par cette cylindrée.

Q 6 . Dans un `main` créez une agence à laquelle vous ajouterez à la fois des objets `Vehicle`, `Car` et `Motorbike` et faites une sélection sur un prix dont vous afficherez le résultat.

Q 7 . Créez une classe `SuspiciousRentalAgency` qui hérite de `Agency` et qui applique un surcoût de 10% sur le prix de location pour les conducteurs dont l'âge est inférieur à 25.

Comment gérer au mieux ce surcoût dans le code ?

Q 8 . Complétez le `main` précédent pour expérimenter le fonctionnement d'un telle agence.