Past-Year PE1 Question: Snatch A Ride

Adapted from PE1 of 19/20 Semester 1

Instructions to Past-Year PE1 Question:

1. Accept the repo on GitHub Classroom here
2. Log into the PE nodes and run `~cs2030s/get py1` to get the skeleton for all available past year PE1 questions.
3. The skeleton for this question can be found under `1920-s1-q1`. You should see the following files:
   - The files `Test1.java`, `Test2.java`, `Test3.java`, and `CS2030STest.java` for testing your solution.
   - No skeleton files are provided for this question.

Background

Snatch Pte Ltd is a transport service provider trying to vie for a place in the public transport arena. Snatch provides three types of ride services:

`JustRide`: `JustRide` charges a fare based on the distance traveled, at 22 cents per km, and the fare is the same regardless of the number of passengers. There is a surcharge of 500 cents if a ride request is issued between 0600 hours and 0900 hours, both inclusive.

`TakeACab`: `TakeACab` charges its fare based on distance traveled, at 33 cents per km, but there is a booking fee of 200 cents. The fair is the same regardless of the number of passengers. There is no peak hour surcharge.

`ShareARide`: The fare depends on the number of passengers and is calculated as follows: the base fare is 50 cents per km, but the passengers pay less if they share the ride. The paid fare is the base fare divided by the number of passengers with any fractional part of the fare (after division) absorbed by the driver. There is a surcharge of 500 cents if a ride request is issued between 0600 hours and 0900 hours, both inclusive.

In addition, there are two types of cars under Snatch. A `Cab` can provide only `JustRide` and `TakeACab` services. A `PrivateCar` can provide only `JustRide` and `ShareARide`

services.

A customer can issue a Snatch ride request, specified by the distance of the ride, the number of passengers, and the time of the request. A booking is made when a request is matched with a car under a particular ride service.

To get full marks, your code not only needs to be correct (including passing all the test cases) but its design must be extensible. In case, Snatch decides to provide additional types of ride services, support additional types of cars, or change the fare structure, your code should require minimal changes to support the new requirements.

Task

Request

Implement a `Request` class that encapsulates a request for a ride. The constructor for `Request` takes in three `int` parameters, the distance of the ride, the number of passengers, and the time of the request.

Services

Implement the three classes `JustRide`, `TakeACab`, and `ShareARide`. These classes should implement a `computeFare` method that takes in a `Request` instance as a parameter and returns the fare in cents.

```
jshell> new JustRide().computeFare(new Request(20, 3, 1000))
$.. ==> 440
jshell> new JustRide().computeFare(new Request(10, 1, 900))
$.. ==> 720
jshell> new TakeACab().computeFare(new Request(20, 3, 1000))
$.. ==> 860
jshell> new TakeACab().computeFare(new Request(10, 1, 900))
$.. ==> 530
jshell> new ShareARide().computeFare(new Request(20, 3, 1000))
$.. ==> 333
jshell> new ShareARide().computeFare(new Request(10, 1, 900))
$.. ==> 1000
```

In addition, each class should override `toString` to return the name of the service.

```
jshell> new JustRide().toString()
$.. ==> "JustRide"
jshell> new TakeACab().toString()
$.. ==> "TakeACab"
jshell> new ShareARide().toString()
$.. ==> "ShareARide"
```

You can test your code by running the `Test1.java` provided. Make sure your code follows the CS2030S Java style.

```
$ javac Test1.java
$ java Test1
$ java -jar ~cs2030s/bin/checkstyle.jar -c ~cs2030s/bin/cs2030_checks.xml
*.java
```

Service.java

```
abstract class Service {
  public abstract int computeFare(Request request);
}
```

JustRide.java

```
class JustRide extends Service {
    private static final int RATE = 22;
    private static final int SURCHARGE = 500;

    @Override
    public int computeFare(Request request) {
        return request.getDistance() * RATE +
            (request.getTime() >= 600 && request.getTime() <= 900 ?
SURCHARGE : 0);
    }

    @Override
    public String toString() {
        return "JustRide";
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof JustRide);
    }
}
```

TakeACab.java

```java
class TakeACab extends Service {
    private static final int RATE = 33;
    private static final int MINFARE = 200;

    @Override
    public int computeFare(Request request) {
        return MINFARE + request.getDistance() * RATE;
    }

    @Override
    public String toString() {
        return "TakeACab";
    }

    @Override
    public boolean equals(Object o) {
        return o instanceof TakeACab;
    }
}
```

ShareARide.java

```java
class ShareARide extends Service {
    private static final int RATE = 50;
    private static final int SURCHARGE = 500;

    @Override
    public int computeFare(Request request) {
        return (request.getDistance() * RATE +
                (request.getTime() >= 600 && request.getTime() <= 900 ?
SURCHARGE : 0)) /
            request.getNumOfPassengers();
    }

    @Override
    public String toString() {
        return "ShareARide";
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof ShareARide);
    }
}
```

Cars

Implement two classes `Cab` and `PrivateCar`. Their constructors should take in a `String` instance that corresponds to the license plate and the time (in minutes) until the driver is

available. In addition, each class should override `toString` to return the type of car, the license plate, and the time until the driver is available. The string should be formatted as shown in the examples below.

```
jshell> new Cab("SHA1234", 5).toString()
$.. ==> "Cab SHA1234 (5 mins away)"
jshell> new Cab("SHA1234", 1).toString()
$.. ==> "Cab SHA1234 (1 min away)"
jshell> new PrivateCar("SU4032", 4).toString()
$.. ==> "PrivateCar SU4032 (4 mins away)"
jshell> new PrivateCar("SU4032", 1).toString()
$.. ==> "PrivateCar SU4032 (1 min away)"
```

You can test your code by running the `Test2.java` provided. Make sure your code follows the CS2030S Java style.

```
$ javac Test2.java
$ java Test2
$ java -jar ~cs2030s/bin/checkstyle.jar -c ~cs2030s/bin/cs2030_checks.xml
*.java
```