Exercise: Snatch A Ride

Adapted from PE1 of 19/20 Semester 1

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) is 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
```

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
```

Bookings

Implement a class `Booking` that encapsulates a car, a service, and a request. A booking should implement the `Comparable<Booking>` interface. A booking is compared to another booking based on the fare, breaking ties by the waiting time. If two bookings have the same fare and waiting time, you can break ties arbitrarily.

```
jshell> Comparable<Booking> b = new Booking(new Cab("SHA1234", 5), new
JustRide(), new Request(20, 3, 1000));
jshell> Booking b1 = new Booking(new Cab("SHA1234", 3), new JustRide(),
new Request(20, 3, 1000));
jshell> Booking b2 = new Booking(new Cab("SBC8888", 5), new JustRide(),
new Request(20, 3, 1000));
jshell> Booking b3 = new Booking(new PrivateCar("SU4032", 5), new
ShareARide(), new Request(20, 3, 1000));
jshell> b3.compareTo(b2) < 0
$33 ==> true
jshell> b1.compareTo(b3) < 0
$34 ==> false
jshell> b1.compareTo(b2) < 0
$35 ==> true
```

If a booking is created with a car and a service that is not compatible (i.e., the type of car does not provide the given service), throw an `IllegalArgumentException`. Construct an `IllegalArgumentException` instance by passing in a message (of type `String`) into its constructor. This message can be retrieved by the `getMessage()` method.

```
jshell> try {
   ...>   new Booking(new Cab("SHA1234", 5), new ShareARide(), new
Request(20, 3, 1000));
   ...> } catch (IllegalArgumentException e) {
   ...>   System.out.println(e.getMessage());
```

```
6       ...> }
    Cab SHA1234 (5 mins away) does not provide the ShareARide service.
```

You can test your code by running the `Test3.java` provided. Make sure your code follows the CS2030S Java style.

```
1   $ javac Test3.java
2   $ java Test3
3   $ java -jar ~cs2030s/bin/checkstyle.jar -c ~cs2030s/bin/cs2030_checks.xml
    *.java
```