Snatch!

Topic Coverage

- Abstract Classes / Interfaces
- Inheritance vs Composition
- Design Principles
- [CS2030 Java Style Guide](#)

Problem Description

Snatch is yet another transport service provider trying to vie for a place in the public transport arena.

Snatch provides three types of ride services:

- `JustRide`
  - Fare is based on the distance @ `22` cents per km
  - Fare is the same regardless of the number of passengers (pax)
  - There is no booking fee.
  - A surcharge of 500 cents if a ride request is issued between the peak hour of `600` hrs to `900` hrs, both inclusive
- `TakeACab`
  - Fare is based on the distance @ `33` cents per km, but there is a booking fee of 200 cents
  - Fare is the same regardless of the number of passengers (pax)
  - No peak hour surcharge
- `ShareARide`
  - Fare is based on the distance @ `50` cents per km
  - Fare is divided equally among the number of passengers
  - There is no booking fee.
  - Any fractional part of the fare is absorbed by your friendly driver
  - A surcharge of 500 cents if a ride request is issued between `600` hrs to `900` hrs, both inclusive

In addition, there are two types of drivers under *Snatch*. Each can provide a subset of the services above.

- `NormalCab` drivers provide `JustRide` and `TakeACab` services.
- `PrivateCar` drivers provide `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.

Task

You shall be given a request, followed by a list of `NormalCab` or `PrivateCar` drivers together with their corresponding waiting times. A booking is a pairing of a driver with the request. The task is to find the best booking with the lowest fare by matching the available drivers based on the services they provide to the given request. Break ties among the same lowest fares by selecting the booking with the smaller waiting time.

You may assume that no two bookings have the same fare and the same waiting time.

This task is divided into several levels. Read through all the levels to see how the different levels are related.

Remember to:

- always compile your program files first before using `jshell` to test your program
- use `checkstyle` and `javadoc` comments to enhance code readability and facilitating code review

Level 1

Define a `Request` class to handle a request comprising the distance, number of passengers, and time.

```
$ javac your_java_files
$ jshell -q your_java_files_in_bottom-up_dependency_order < test1.jsh
jshell> new Request(20, 3, 1000)
$.. ==> 20km for 3pax @ 1000hrs
jshell> new Request(10, 1, 900)
$.. ==> 10km for 1pax @ 900hrs
jshell> /exit
```

Level 2

Now include the `JustRide` and `TakeACab` services. Note that every service needs to implement the `computeFare` method that returns the fare in cents.

```
$ javac your_java_files
$ jshell -q your_java_files_in_bottom-up_dependency_order < test2.jsh
jshell> new JustRide()
$.. ==> JustRide
jshell> new JustRide().computeFare(new Request(20, 3, 1000))
$.. ==> 440
jshell> new JustRide().computeFare(new Request(10, 1, 900))
$.. ==> 720
jshell> new TakeACab()
$.. ==> TakeACab
jshell> new TakeACab().computeFare(new Request(20, 3, 1000))
$.. ==> 860
jshell> new TakeACab().computeFare(new Request(10, 1, 900))
$.. ==> 530
jshell> /exit
```

Level 3

Now, include a `NormalCab` driver who is identified by its license plate number (a string) and the passenger waiting time in minutes.

Then, add a `Booking` class that takes in a driver and a request. From the services that a driver provides, the best service with the lowest fare is selected.

Two bookings may be compared using their computed fares; if both fares are the same, prefer the one with the shorter waiting time.

```
$ javac your_java_files
$ jshell -q your_java_files_in_bottom-up_dependency_order < test3.jsh
jshell> new NormalCab("SHA1234", 5)
$.. ==> SHA1234 (5 mins away) NormalCab
jshell> new Booking(new NormalCab("SHA1234", 5), new Request(20, 3, 1000))
$.. ==> $4.40 using SHA1234 (5 mins away) NormalCab (JustRide)
jshell> new NormalCab("SHA2345", 10)
$.. ==> SHA2345 (10 mins away) NormalCab
jshell> new Booking(new NormalCab("SHA2345", 10), new Request(10, 1, 900))
$.. ==> $5.30 using SHA2345 (10 mins away) NormalCab (TakeACab)
jshell> Booking b1 = new Booking(new NormalCab("SHA2345", 10), new Request(10, 1, 900))
jshell> Booking b2 = new Booking(new NormalCab("SHA2345", 10), new Request(10, 1, 900))
jshell> b1.compareTo(b2) == 0
$.. ==> true
jshell> Booking b3 = new Booking(new NormalCab("SHA1234", 5), new Request(10, 1, 900))
jshell> Booking b4 = new Booking(new NormalCab("SHA2345", 10), new Request(10, 1, 900))
jshell> b3.compareTo(b4) < 0
$.. ==> true
jshell> /exit
```

Level 4

Now include the `ShareARide` service and `PrivateCar` driver.

You should aim to make the `Booking` class general such that it does not need to check for any invalid pairing, say between `PrivateCar` driver and `TakeACab` service. If you have designed your program appropriately, then extending your program with additional drivers and services would not require any modification to existing classes.

```
$ javac your_java_files
$ jshell -q your_java_files_in_bottom-up_dependency_order < test4.jsh
jshell> new ShareARide()
$.. ==> ShareARide
jshell> new PrivateCar("SMA7890", 5)
$.. ==> SMA7890 (5 mins away) PrivateCar
jshell> new Booking(new PrivateCar("SMA7890", 5), new Request(20, 3, 1000))
$.. ==> $3.33 using SMA7890 (5 mins away) PrivateCar (ShareARide)
jshell> new PrivateCar("SLA5678", 10)
$.. ==> SLA5678 (10 mins away) PrivateCar
jshell> new Booking(new PrivateCar("SLA5678", 10), new Request(10, 1, 900))
$.. ==> $7.20 using SLA5678 (10 mins away) PrivateCar (JustRide)
jshell> Booking b1 = new Booking(new PrivateCar("SMA7890", 5), new Request(10, 1, 900))
jshell> Booking b2 = new Booking(new PrivateCar("SLA5678", 10), new Request(10, 1, 900))
jshell> b1.compareTo(b2) < 0
$.. ==> true
jshell> /exit
```

Level 5

Now complete the task by defining the `findBestBooking` method to return the best booking given a request, and an array of drivers with their waiting times. Save the method in the file `level5.jsh`.

Assume that no two bookings have the same fare and the same waiting time.

```
$ javac your_java_files
$ jshell -q your_java_files_in_bottom-up_dependency_order level5.jsh < test5.jsh
jshell> findBestBooking(new Request(20, 3, 1000),
   ...> new Driver[]{new NormalCab("SHA1234", 5), new PrivateCar("SMA7890", 10)})
$.. ==> $3.33 using SMA7890 (10 mins away) PrivateCar (ShareARide)
jshell> findBestBooking(new Request(10, 1, 900),
   ...> new Driver[]{new NormalCab("SHA1234", 5), new PrivateCar("SMA7890", 10)})
$.. ==> $5.30 using SHA1234 (5 mins away) NormalCab (TakeACab)
jshell> /exit
```