CodeCrunch

CS2030 (2310) Mock PA #1

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

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`, `TakeACab` and `ShareARide`.

There are also different types of drivers under *Snatch*. Each can provide a subset of the services above.

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.

The task is to provide a list of bookings (a pairing of a driver with the request) in order of increasing fares 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.

The `ImList` class has also been provided to you.

Take note!

There are altogether five levels. You are required to complete ALL levels.

You should keep to the constructs and programming discipline instilled throughout the module.

- Write each class/interface in its own file.
- Ensure that ALL object properties and class constants are declared private final.
- Ensure that that your classes are NOT cyclic dependent. You are NOT allowed to use java libraries, other than those from `java.lang`, `java.util.List` and `java.util.Comparator`.
- You are NOT allowed to use java keywords/literals: `null`, instanceof (except within the typical `equals` method)
- You are NOT allowed to use Java reflection, i.e. `Object::getClasses` and other methods from `java.lang.Class`

- You are NOT allowed to use * wildcard imports.
- You are NOT allowed to define array constructs, e.g. `String[]` or using ellipsis, e.g. `String...`

You may assume that all tests provide valid arguments to methods; hence there is no need to validate method arguments.

Level 1

The fare of a service is based upon the distance travelled, the number of passengers, and the time of the service.

JustRide is one such service:

- 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

Create the `JustRide` service with `computeFare` method the takes in three integer arguments comprising the distance, number of passengers, and time of service. The method returns the fare in cents.

```
$ javac your_java_files
$ jshell your_java_files_in_bottom-up_dependency_order
jshell> new JustRide()
$.. ==> JustRide

jshell> new JustRide().computeFare(20, 3, 1000)
$.. ==> 440

jshell> new JustRide().computeFare(10, 1, 900)
$.. ==> 720
```

Level 2

Define a `Request` class that encapsulates the distance, number of passengers, and time. Write a method `computeFare` that takes in a service and returns the fare accordingly.

```
$ javac your_java_files
$ jshell your_java_files_in_bottom-up_dependency_order
jshell> new Request(20, 3, 1000)
$.. ==> 20km for 3pax @ 1000hrs

jshell> new Request(20, 3, 1000).computeFare(new JustRide())
$.. ==> 440

jshell> new Request(10, 1, 900)
request ==> 10km for 1pax @ 900hrs

jshell> new Request(10, 1, 900).computeFare(new JustRide())
$.. ==> 720
```

In addition, include the `TakeACab` service:

- 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

```
jshell> new TakeACab()
$.. ==> TakeACab

jshell> new Request(20, 3, 1000).computeFare(new TakeACab())
$.. ==> 860

jshell> new Request(10, 1, 900).computeFare(new TakeACab())
$.. ==> 530
```

Level 3

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

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.

To compare two bookings using their computed fare, we let the `Booking` class implement the `Comparable` interface that specifies a `compareTo` method to be implemented. (note that this is different from the `Comparator` interface).

```
class Booking implements Comparable<Booking> {
   ...
   @Override
   public int compareTo(Booking other) {
      ...
   }
}
```

Note that if both fares are the same, we prefer the one with the shorter waiting time.

```
$ javac your_java_files
$ jshell your_java_files_in_bottom-up_dependency_order
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))
b1 ==> $5.30 using SHA2345 (10 mins away) NormalCab (TakeACab)

jshell> Booking b2 = new Booking(new NormalCab("SHA2345", 10), new Request(10, 1, 900))
b2 ==> $5.30 using SHA2345 (10 mins away) NormalCab (TakeACab)

jshell> Comparable<Booking> cmp = b1
cmp ==> $5.30 using SHA2345 (10 mins away) NormalCab (TakeACab)

jshell> b1.compareTo(b2) == 0
$.. ==> true

jshell> Booking b3 = new Booking(new NormalCab("SHA1234", 5), new Request(10, 1, 900))
b3 ==> $5.30 using SHA1234 (5 mins away) NormalCab (TakeACab)

jshell> Booking b4 = new Booking(new NormalCab("SHA2345", 10), new Request(10, 1, 900))
b4 ==> $5.30 using SHA2345 (10 mins away) NormalCab (TakeACab)

jshell> b3.compareTo(b4) < 0
$.. ==> true
```

Level 4

Now include the `ShareARide` service:

- Fare is based on the distance `@ 50` cents per km
- A surcharge of 500 cents if a ride request is issued between `600` hrs to `900` hrs, both inclusive
- There is no booking fee
- The total fare is divided equally among the number of passengers
- Any fractional part of the fare is absorbed by your friendly driver

Moreover, include

- `PrivateCar` drivers that provide `JustRide` and `ShareARide` services.

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 your_java_files_in_bottom-up_dependency_order
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))
b1 ==> $7.20 using SMA7890 (5 mins away) PrivateCar (JustRide)

jshell> Booking b2 = new Booking(new PrivateCar("SLA5678", 10), new Request(10, 1, 900))
b2 ==> $7.20 using SLA5678 (10 mins away) PrivateCar (JustRide)

jshell> b1.compareTo(b2) < 0
```

Level 5

Now complete the task by defining the `findBestBooking` method in `level5.jsh` that returns the list of booking matches sorted by increasing fare (breaking ties by waiting time).

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

```
jshell> /open level5.jsh

jshell> ImList<Booking> bookings = findBestBooking(new Request(20, 3, 1000),
   ...> List.of(new NormalCab("SHA1234", 5), new PrivateCar("SMA7890", 10)))
bookings ==> [$3.33 using SMA7890 (10 mins away) PrivateCar (S ... way) NormalCab (TakeACab)]

jshell> for (Booking b : bookings) {
   ...>     System.out.println(b);
   ...> }
$3.33 using SMA7890 (10 mins away) PrivateCar (ShareARide)
$4.40 using SHA1234 (5 mins away) NormalCab (JustRide)
$4.40 using SMA7890 (10 mins away) PrivateCar (JustRide)
$8.60 using SHA1234 (5 mins away) NormalCab (TakeACab)

jshell> bookings = findBestBooking(new Request(10, 1, 900),
   ...> List.of(new NormalCab("SHA1234", 5), new PrivateCar("SMA7890", 10)))
bookings ==> [$5.30 using SHA1234 (5 mins away) NormalCab (Tak ... ) PrivateCar (ShareARide)]

jshell> for (Booking b : bookings) {
   ...>     System.out.println(b);
   ...> }
$5.30 using SHA1234 (5 mins away) NormalCab (TakeACab)
$7.20 using SHA1234 (5 mins away) NormalCab (JustRide)
$7.20 using SMA7890 (10 mins away) PrivateCar (JustRide)
$10.00 using SMA7890 (10 mins away) PrivateCar (ShareARide)
```