Term 4 Assignment 4

Quicka Self-driving

You are the Lead Systems Engineer for the Quicka ridesharing platform and are in charge of developing all the necessary software for drivers, riders, and self-driving cars.

For this assignment, you are asked to write the Control Operation System (aka COS-101) software for the Quicka Autonomous Vehicle. The computers onboard the QAV are outfitted with a JVM capable of running Java code.

When designing and writing your systems, you may use any IDE of your choice, but you must ensure that your code is able to run on the JVM.

Mission 4.4: Passengers, Rides, and Drivers

For this mission, we have been asked to help with the planning and development of the core of the Quicka ridesharing platform. For now, we will focus on human drivers and passengers.

We have also been supplied with generated JavaDocs in the `docs.zip` file. To read them, we need only open the `docs/index.html` file in a compatible browser.

Task 1: Planning and Passengers

For this task we will be creating an overview of the classes in our system using the Unified Modeling Language (UML) Class Diagram.

IMPORTANT: When submitting this question, you *must* submit it as a Portable Network Graphics (PNG) file.

Below is a list of tools for drawing UML diagrams, but you are free to use any program of your choosing:

- Google Drawings (https://docs.google.com/drawings/)
- PlantUML (https://plantuml.com/)
- Inkscape (https://inkscape.org/)
- Visual Paradigm (https://online.visual-paradigm.com/)

You may use any tool you wish as long as your diagram follows the UML conventions and is in the correct format (PNG).

Instructions *IMPORTANT*: Unless otherwise stated, all attributes should be private or protected. You may decide for yourself whether an attribute should be private or protected.

Draw the UML class diagrams for the following classes:

1. An abstract Java class called `Person` that has the following attributes:
   - a String called `idNumber`
   - a Date called `dateOfBirth` representing the date of birth.
   - a String for `name`
2. A class called `RideRecord` with the following attributes:
   - a `Driver` attribute
   - an enum called `type` of `RideType`
   - a Date called `date`
   - a double called `fee`
3. A class called `Passenger` that extends the `Person` class and has the following attributes and behaviours:
   - an ArrayList that contains `RideRecord` objects called `rideHistory`
   - a public method called `takeRide` that takes a `RideRecord` with the following signature:

   ```
   public void takeRide(RideRecord rideRecord);
   ```

   - a public boolean method called `hasRideHistory` that returns whether or not the `Passenger` has taken a ride before.
4. A `Driver` class that extends the `Person` class and has the following attributes and behaviors:
   - a String called `licenseCode`
   - a public method called `giveRide` that takes as a parameter a `Passenger` object, a `RideType`, and a double `fee`.

Task 2: General Abstracts

For this task you will be asked to create the Java classes described in the UML diagram in Task 1.

Instructions

1. The class `Person` has the following implementations in addition to those specified above:
   - the `dateOfBirth` attribute is of type `java.util.Date`
   - public getter methods for the attributes
   - a public constructor `Person` with the following method signature:

   ```
   public Person(String idNumber, String name, Date dateOfBirth)
   ```

   - declare an abstract `compareTo` method in `Person` with the following signature:

`public int compareTo(Person p);`

2. The class called `RideRecord` has the following extra details:
   - the enum called `RideType` has the following possible values:

   `DUIKER,`
   `KUDU,`
   `TEMBO`
   - the public constructor for the `RideRecord` has the following signature:

   `public RideRecord(Driver driver, RideType type, double fee);`
   - the `date` attribute is of type `java.util.Date`
3. The class called `Passenger` that extends the `Person` class has the following additional attributes and behaviours:
   - the public method called `takeRide` takes a `rideRecord` and adds it to the `rideHistory` list
   - a constructor with the following method signature:

   `public Passenger(String idNumber, String name, Date dateOfBirth);`
4. The class called `Driver` has the following additional attributes and behaviours:
   - a public constructor with the following method signature:

   `public Driver(String idNumber, String name, Date dateOfBirth, String licenseCode);`
   - the `giveRide` method creates a new `RideRecord` using the `RideType` and `fee` and calls the `takeRide` method on the `Passenger`. The method signature is as follows:

   `public void giveRide(RideRecord.RideType type, double fee, Passenger passenger);`

Sample Input/Output

The following demo code produces the sample output:

```
import java.util.Date;
public class AssignmentDemo {
  public static void main(String[] args) {
    Driver goliath = new Driver("6405013145087", "John Goliath",
                                new Date(-178934400000L), "CAA202210");

    Passenger david =
        new Passenger("9906014269088", "David Chord", new Date(928195200000L));
    System.out.printf("Name: %s\nRSA ID:%s\nDOB:%s\nVaccinated?: %b\n",
                      david.getName(), david.getIdNumber(),
                      david.getDateOfBirth(), david.hasRideHistory());
    goliath.giveRide(RideRecord.RideType.KUDU, 56.00, david);
    System.out.println("----------------------------");
    System.out.printf("Name: %s\nRSA ID:%s\nDOB:%s\nHas ride history?: %b\n",
                      david.getName(), david.getIdNumber(),
                      david.getDateOfBirth(), david.hasRideHistory());
  }
}
```

```
Name: David Chord
RSA ID:9906014269088
DOB:Tue Jun 01 02:00:00 SAST 1999
Has ride history?: false
----------------------------
Name: Ash Ketchum
RSA ID:9906014269088
DOB:Tue Jun 01 02:00:00 SAST 1999
Has ride history?: true
```

Submission Instructions

Submit only your files `Person.java`, `RideRecord.java`, `Passenger.java`, `Driver.java` and `UMLClassDiagram.png` in a compressed `.zip` folder labeled with your student number an underscore and `assignment7`.

For example if your student number is `12345` submit a file `12345_assignment7.zip` that contains `Person.java`, `RideRecord.java`, `Passenger.java`, `Driver.java` and `UMLClassDiagram.png`.

Late submissions incur a 10% penalty per day maximum of 5 days late.

Marking

Section	Marks
Task 1: Planning and Passengers	15
Task 2: General Abstracts	25
Total	**40**