

TCSS 305 Programming Practicum
Assignment 3: Inventory Creation for Rentz
Value: 10% of the course grade
Due: Monday, 17 February 2019, 23:59:00

Program Description:

Assignment 0, Assignment 1, 2, 3, 4 aim to develop Rentz, a vehicle rental system. You will incrementally build your code for each assignment, i.e., your Assignment 2 will be built using the code that you submitted for Assignment 1. Make sure you finish every assignment on time, so that your future submissions remain unaffected. Assignment 3 focuses on creating the Vehicle Classes using inheritance for Rentz.

Rentz is operated by a rental agent. The rental agent should be a registered user. The rental agent can 1) Register New Users; 2) Login to the rental system; 3) Rent a vehicle for a registered user; 4) Drop-off a vehicle for a registered user; 5) Exit from the system. In Assignment 1-3, you have already created and tested the registration and login functionalities. Assignment 3-4 will focus on the remaining tasks.

Using Previous Code: Rename your project to “username-rentz-3” and share it to the repository following steps 1-3 under “Using Previous Code” given in “hw1.pdf”.

Implementation Guidelines:

Package `model.vehicles`: You need to create classes in the “`model.vehicles`” package following principles of inheritance appropriately. The classes need to be created based on the guidelines given below. *You can add class private/public fields, methods, apart from the ones mentioned in the guidelines if necessary.* Unnecessary class members will be penalized. Below just gives the guidelines, you can move around members between the parent-child for a better class design.

`Vehicle`

- You can pre-pend “Abstract” to the class name, if necessary
- Every `vehicle` should have the following fields (they can have other fields if necessary)

Class Fields	Description
<code>int myVehicleID</code>	Unique ID generated by rental manager during inventory creation. It starts from 1 and generated sequentially
<code>String myVIN</code>	Unique Vehicle Identification Number
<code>String myName</code>	The name of the vehicle

- `Vehicle` can be of 3 types: `BiCycle`, `MotorBike`, `Car`
- `Vehicle` can be available/unavailable for renting
- Every `Vehicle` has a `myRentalAmount` (Big Decimal value in \$) per day
- The `BASE_FARE` for renting any vehicle is \$10.00 per day

BiCycle:

- The `CYCLE_FARE` for renting bi-cycles is same as `BASE_FARE` per day
- BiCycles can be of 4 types: Road, Mountain, Cruiser, Hybrid
- The rental fare for “Road” BiCycles, is the same as the normal `CYCLE_FARE`
- The rental fare for “Mountain” BiCycles, is 1% more than the normal `CYCLE_FARE`
- The rental fare for “Cruiser” BiCycles, is 2% more than the normal `CYCLE_FARE`
- The rental fare for “Hybrid” BiCycles, is 4% more than the normal `CYCLE_FARE`
- Use the function `calculateRentalAmount()` to compute the total rental fare depending on the cycle type.
- The string representation of BiCycle should follow the below example:
`BiCycle (ID:6, Name:Roadies, VIN:C100, CanRent?:true, CycleType:Road)`
- The `equals` and `hashCode` methods should be based on all fields except `myRentalAmount`

MotorBike:

- The `BIKE_FARE` for renting motor-bikes is twice the `BASE_FARE` per day
- If the MotorBike is of “Touring” type, an additional \$5 is charged above the `BIKE_FARE`
- Use the function `calculateRentalAmount()` to compute the total rental fare depending on the above rules.
- The string representation of MotorBike should follow the below example:
`MotorBike (ID:5, Name:Bike2, VIN:B101, CanRent?:true, IsTouring?:true)`
- The `equals` and `hashCode` should be based on all fields except `myRentalAmount`

Car:

- The `CAR_FARE` for renting cars is thrice the `BASE_FARE` per day
- If the car is a “Luxury” car, an additional \$10.00 is charged above the `CAR_FARE`
- If the car has “Navigation”, an additional \$1.00 is charged above the `CAR_FARE`
- If the car is a “DrivingAssistance”, an additional \$2.00 is charged above the `CAR_FARE`
- Use the function `calculateRentalAmount()` to compute the total rental fare depending on the about rules.
- The string representation of Car should follow the below example:
`Car (ID:3, Name:BMW, VIN:V102, CanRent?:true, IsLuxury?:true, HasNavigation?:true, HasAssistance?:true)`
- The `equals` and `hashCode` should be based on all fields except `myRentalAmount`

Package model: You have already created the User, Registration class. You will need to add RentalManager class and make changes to the RentalMain class. *You can add class private/public fields, methods, apart from the ones mentioned in the guidelines if necessary.* Unnecessary class members will be penalized.

RentalManager:

Field	Description		
myVehicleList	HashMap which saves with Key= myVehicleID, Value=Vehicle Object		
myRegistration	A reference to the Registration object created for registration/login		
Method	Parameters	Return Type	Description
Parameterized Constructor	Registration Object		<ul style="list-style-type: none"> Assigns initial values for myRegistration myVehicleList is initialized with inventory by calling the function generateInventory() <p>The parameters should be checked for null values and should implicitly throw NullPointerException for null parameters</p>
generateInventory		Map <VehicleID, Vehicle Object>	Creates 8 vehicle objects by assigning VehicleID in sequence. The exact specs for the vehicles can be seen in the execution output shown below.
getMyVehicleList		Map <VehicleID, Vehicle Object>	Getter for myVehicleList
printOptions			From the inventory generated by generateInventory method, it selects the available vehicles and prints them as shown below
getMyRegistration		Registration object	Returns the registration object

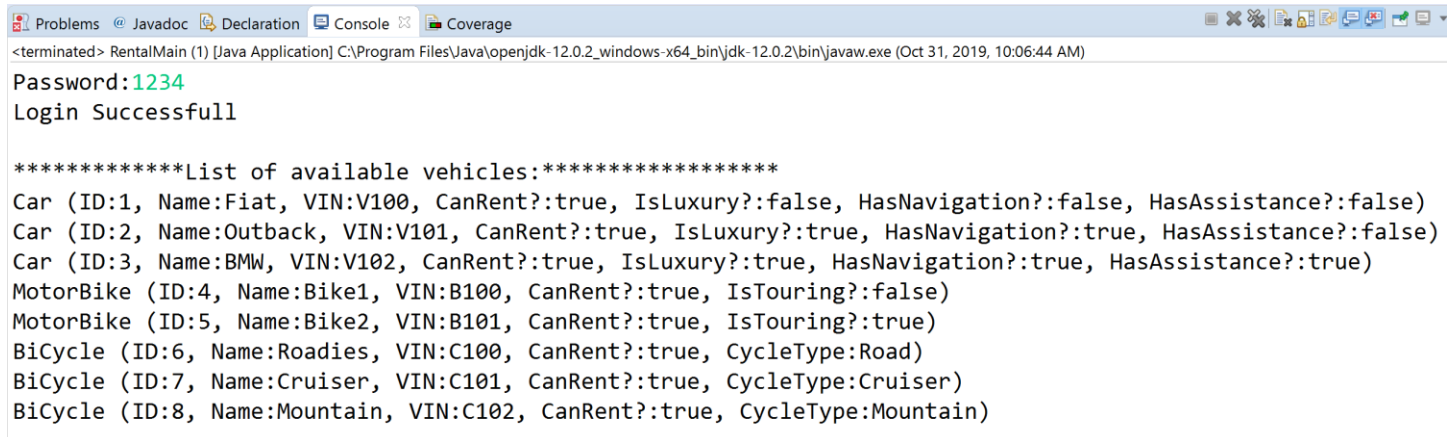
Registration: This class needs to be modified

Method	Parameters	Return Type	Description
printSignin		boolean	Only when Login option is selected and login is successful, printSignin should return true. It should return false in all other cases.

RentalMain: This class needs to be modified

When login is successful, i.e., `printSigin` returns true, create an Object of `RentalManager` and call the `printOptions` function from the `RentalManager` class.

Expected Output: When you run the `RentalMain` class this should be the expected output (Exact formatting should be followed)



```
Problems  @ Javadoc  Declaration  Console  Coverage
<terminated> RentalMain (1) [Java Application] C:\Program Files\Java\openjdk-12.0.2_windows-x64_bin\jdk-12.0.2\bin\javaw.exe (Oct 31, 2019, 10:06:44 AM)

Password:1234
Login Successfull

*****List of available vehicles:*****
Car (ID:1, Name:Fiat, VIN:V100, CanRent?:true, IsLuxury?:false, HasNavigation?:false, HasAssistance?:false)
Car (ID:2, Name:Outback, VIN:V101, CanRent?:true, IsLuxury?:true, HasNavigation?:true, HasAssistance?:false)
Car (ID:3, Name:BMW, VIN:V102, CanRent?:true, IsLuxury?:true, HasNavigation?:true, HasAssistance?:true)
MotorBike (ID:4, Name:Bike1, VIN:B100, CanRent?:true, IsTouring?:false)
MotorBike (ID:5, Name:Bike2, VIN:B101, CanRent?:true, IsTouring?:true)
BiCycle (ID:6, Name:Roadies, VIN:C100, CanRent?:true, CycleType:Road)
BiCycle (ID:7, Name:Cruiser, VIN:C101, CanRent?:true, CycleType:Cruiser)
BiCycle (ID:8, Name:Mountain, VIN:C102, CanRent?:true, CycleType:Mountain)
```

Extra Credit (Upto 5 points):

You must write JUNIT test cases for all the classes under `model.vehicles` package. The test cases should have 100% coverage.

Submitting your executive summary:

The executive summary template will be provided to you in canvas. Download the file. Rename the executive summary “`executivesummary-username-rentz-3`”, where `username` is your UWNNetID. Fill and upload the *executive summary* to canvas as the same WORD format. The grader will grade your submission based on the executive summary as well as the project you submit to the SVN repository. If you do not submit either of them, you will receive no grades.

Grading Rubric (Total 100 points (105 possible)):

Task	Max Score Possible
Executive Summary -Submission on canvas with correct version number	2
Source Code – Submission to SVN	2
model.vehicles (correct use of package)	2
Creation of correct class hierarchy	10
Vehicle Class – Appropriate Class Name	2
Vehicle Class – Appropriate fields created with correct modifiers	5
Vehicle Class – Appropriate methods created	10
Vehicle Class – Use of abstract methods when necessary	2
Car – correct hierarchy used	2
Car – correct fields created with proper modifiers	2
Car – implementation of CalculateRentalAmount()	3
Car – implementation of toString()	2
Car – implementation of equals() and hashCode()	5
BiCycle – correct hierarchy used	2
BiCycle – correct fields created with proper modifiers	2
BiCycle – implementation of CalculateRentalAmount()	3
BiCycle – implementation of toString()	2
BiCycle – implementation of equals() and hashCode()	5
MotorBike – correct hierarchy used	2
MotorBike – correct fields created with proper modifiers	2
MotorBike – implementation of CalculateRentalAmount()	3
MotorBike – implementation of toString()	2
MotorBike – implementation of equals() and hashCode()	5
RentalManager – correct fields with proper modifiers created	2
RentalManager – constructor implementation	2
RentalManager – generateInventory implementation	5
RentalManager – getMyVehicleList implementation	1
RentalManager – getMyRegistration implementation	1
RentalManager – printOptions implementation	3
Registration – printSignin method update	1
Rental Main – main method update	1
Correct Project Name and Executive summary name	1
No Errors and Warnings (except reasonable ones which should be clearly explained in executive summary)	3
Proper Java docs and Header comments	3
Extra Credit	5
Total (105 possible)	100