Introduction to Java

CS9053 Section I2

Wednesday 6:00 PM – 8:30 PM

Prof. Dean Christakos

Feb 7th, 2025

Due: Feb 14th, 2025 11:59 PM

Part I – Creating Objects

1. Dodecahedron: In the lecture you have seen the creation of a circle. Here you are going to create a dodecahedron. Dodecahedron is a shape with 12 pentagonal sides, with each pentagon having 5 equal sides of length **a**. This side **a** is called the **edge**.

A colorful octahedron with a black background

Description automatically generatedA hexagonal shape with a letter a

Description automatically generated

The volume of dodecahedron is given by

And the surface area is given by

The height is given by:

You will create a class Dodecahedron using the following UML:

A screenshot of a computer

Description automatically generated

**In standard UML parlance, “+” indicates that a field or method is public and “-“ indicates that a field or method is private. An underlined field or method indicates it is static.**

Every time you create a new Dodecahedron instance, it should have a new sequential id, based on the value of nextId, which should be incremented every time you create a new Dodecahedron instance.

1. Objects and Arrays of Objects

Your objective is to develop the **RentalCar** and **RentalAgency** class and to use their methods. The classes are described below to guide you.

RentalCar

The **RentalCar** class contains data related to a rental car.The **RentalCar** class is not fully immutable, but only one of its features can be changed, the **dailyCost**. Here is the UML:

A screenshot of a computer

Description automatically generated

I didn’t put any methods or constructors in here. Figure that out how to make methods and constructors for a **RentalCar** class while allowing us to read the data and change the **dailyCost**. The constructor should be able to fully describe the rental car and automatically generate an ID.

Car Rentals

The **RentalAgency** class contains and Array of **RentalCar** objects that are held by the **RentalAgency**. The **RentalCar** objects have an ID and the **RentalAgency** keeps track of which cars are rented out by id. When renting out a car, the rentCar() method takes a number of days the car will be rented for as an argument returns the **RentalCar** object and returns null if it isn’t available (eg, doesn’t exist or is already rented out). (you probably want to have the “empty” in the rented array be -1). returnCar(int, days) takes an id and an integer number of days as an argument returns a double representing the cost of the rental (dailyCost x rentalDays) if the car is part of the stock and has been rented out and -1 if it isn’t rented out or doesn’t exist. The **RentalAgency** has a pre-set and **constant** capacity of 10 cars and addRentalCar return false if a **RentalCar** is added to the **RentalAgency** beyond its capacity. A **RentalCar** can be removed from the inventory with removeRental unless it is rented out or doesn’t exist in the inventory, in which case it returns false. Note what can be changed and what can’t be changed based on the methods available in the UML.

Implement the class according to the following UML diagram:

A screenshot of a computer

Description automatically generated

Because we are using arrays rather than other data structures like Maps and Sets, the process of inserting and removing items from arrays is going to be inefficient. That’s ok. We aren’t testing you for a job. Do what you have to do, even if they run in n2

After implementing this, create a **RentalAgency** called “Enterprise” in Farmingdale, NY. Then create three **RentalCar** objects of your choice (daily costs of $50, $75, and $85), add them to the **RentalAgency**. Show the rental cars that are in the inventory. Rent out a car for and show which cars are remaining. Return a car after 5 days and show how much money is due.