CSE 102L Computer Programming Laboratory – Exercise1

Due Date: 23:59 Friday March 3rd

Information

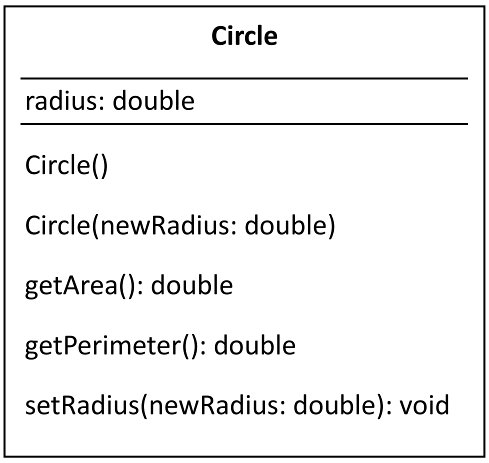
A class defines the properties and behaviors for objects. An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle. An object has a unique identity, state and behavior.

- The *state* of an object (also known as its properties or attributes) is represented by *data fields* with their current values. A circle object, for example, has a data field **radius**, which is the property that characterizes a circle. A rectangle object has the data fields **width** and **height**.
- The *behavior* of an object (also known as its actions) is defined by methods. To invoke a method on an object is to ask the object to perform an action. For example, you may define methods named **getArea()** and **getPerimeter()** for circle objects. You may also define a **setRadius(radius)** method to modify the data fields.

Objects of the same type are defined using a common class. A class is a template, blueprint or contract that defines what an object is. An object is an instance of class. You can create instances of a class. Creating an instance is referred to as *instantiation*. The terms object and instance are often interchangeable. The relationship between classes and objects is analogous to that between an apple-pie recipe and apple pies.

A java class uses variables to define data fields and methods to define actions. Additionally, a class provide methods of a special type, known as *constructors*, which are invoked to create a new object. A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects. The *new* operator is used to instantiate an object from a class

The illustration of class templates and objects can be standardized using *Unified Modelling Language (UML)* notation. This notation, as shown in below, is called *UML class diagram*, or simply a *class diagram.*



In the class diagram:

the data field is denoted as *dataFieldName: dataFieldtype*, the constructor is denoted as *ClassName(parameterName: parameterType),*

and finally methods are denoted as *methodName(parmeterName: parameterType): returnType*

Following code is written according to given above UML diagram.

```java
class Circle {
    /////// Data Fields ///////

    /** The radius of this circle */
    double radius = 1;

    /////// Constructors ///////
    /** Construct a circle object */
    Circle() { }

    /** Construct a circle object given its radius */
    Circle(double newRadius) {
        radius = newRadius;
    }

    /////// Methods ///////
    /** Return the area of this circle */
    double getArea() {
        return radius * radius * Math.PI;
    }

    /** Return the perimeter of this circle */
    double getPerimeter() {
        return 2 * radius * Math.PI;
    }

    /** Set new radius for this circle */
    void setRadius(double newRadius) {
        radius = newRadius;
    }
}
```

Disclosure

You will submit your file to an assignment that is given through MS teams. Your filename should be *Ex1_yourStudentNumber.java*. Your main method should be in the class with your student number (*Ex1_yourStudentNumber*). Submissions made after the deadline will not be accepted, be sure to submit your work before the due date. Your code will be automatically controlled, so be sure to have the same class, method, variable names. Failure to do so may result in you receiving 0 from this exercise. All classes should be written to a single Java file.

Your UML Class diagrams should be in pdf format.

Exercises

1. (The Stock Class): Write a class named **Stock** that contains:

- A **string** data field named **symbol** for the stock's symbol
- A **string** data field named **name** for the stock's name
- A **double** data field named **previousClosingPrice** that storess the stock price for the previous day
- A **double** data field named **currentPrice** that stores the stock price for the current time
- A constructor that creates a stock with the specified symbol and name
- A method named **getChangePercent()** that returns the percentage change from **previousClosingPrice** to **currentPrice()**.

Draw the UML diagram for the class. Write a test program (in a main method) that creates a **Stock** object with the stock symbol **ORCL**, the name **Oracle Corporation**, and the previous closing price of **34.5**. Set a new current price **34.35** and display the price-change percentage.

2. (The Fan Class) Design a class named **Fan** to represent a fan. The class contains:

- Three constants named **SLOW**, **MEDIUM**, and **FAST** with values **1**, **2**, and **3** to denote the fan speed
- A private **int** data field named **speed** that specifies the speed of the fan (the default is **SLOW**)
- A private **booelan** data field named **on** that specifies whether the fan is on (the default is **false**)
- A private **double** data field named **radius** that specifies the radius of the fan (the default is **5**)
- A **string** data field named **color** that specifies the color of the fan (the default is **blue**)
- The **accessor and mutator methods** for all four data fields
- A no-arg constructor that created a default fan.
- A constructor that takes radius and color as parameters and creates a fan.
- A method named **toString()** that returns a string description of the fan. If the fan is on, the method returns the fan speed, color and radius in one combined string. If the fan is not on, the method returns the fan color and radius along with the string "fan is off" in one combined string

Draw the UML diagram for the class and implement the class.

Challenge

Write a test program that creates user defined amount of **Fan** objects. For each even indexed fan should be a default fan, for each odd indexed fan should have their radius increase by one and their fan color set as **yellow**. Be sure to use suitable constructor. Write a method that takes an array of **Fans** and increases the speed of every 3rd fan to next level, if it is already in **FAST** level, then it should be set to **SLOW. Do these only if the fan is on.** (*hint: you can use setter method of speed data field to encapsulate this logic*)