CSC 110 2.0 Object Oriented Programming

Tutorial 04

Instructions:

- All questions must be attempted and answers submitted in a handwritten document, on or before, 10.00am on Tuesday, 9th August 2019, to the Instructor Room.
- You must indicate your Index Number and the Tutorial Class to which you belong to (LCS1/ LCS2/ NFC3.1) clearly on the front page of your submission.

A. Understanding Code Segments

For each of the code segments below, write the output of the main method in the respective Driver Class.

Encapsulation

1. Consider the following Animal class and the TestAnimal class which contain our main method.

```
class Animal {
	private String name;
	private double averageWeight;
	private int numberOfLegs;

	// Getter methods
	public String getName() {
		return name;
	}
	public double getAverageWeight() {
		return averageWeight;
	}
	public int getNumberOfLegs() {
		return numberOfLegs;
	}

	// Setter methods
	public void setName(String name) {
		this.name = name;
	}
	public void setAverageWeight(double averageWeight) {
		this.averageWeight = averageWeight;
	}
	public void setNumberOfLegs(int numberOfLegs) {
		this.numberOfLegs = numberOfLegs;
	}
}
```

```
public class TestAnimal {
	public static void main(String[] args) {
		Animal myAnimal = new Animal();
```

```
             myAnimal.setName("Eagle");
             myAnimal.setAverageWeight(1.5);
             myAnimal.setNumberOfLegs(2);

             System.out.println("Name: " + myAnimal.getName());
             System.out.println("Average weight: " + myAnimal.getAverageWeight()
+ "kg");
             System.out.println("Number of legs: " +
myAnimal.getNumberOfLegs());
      }
}
```

Inheritance

2. Consider the following Bird, Eagle and TestEagle classes.

```
class Bird {
      public String reproduction = "egg";
      public String outerCovering = "feather";

      public void flyUp() {
             System.out.println("Flying up...");
      }
      public void flyDown() {
             System.out.println("Flying down...");
      }
}

class Eagle extends Bird {
      public String name = "eagle";
      public int lifespan = 15;
}
```

```
class TestEagle {
      public static void main(String[] args) {
             Eagle myEagle = new Eagle();

             System.out.println("Name: " + myEagle.name);
System.out.println("Reproduction: " + myEagle.reproduction);
             System.out.println("Outer covering: " + myEagle.outerCovering);
             System.out.println("Lifespan: " + myEagle.lifespan);
             myEagle.flyUp();
             myEagle.flyDown();
      }
}
```

Polymorphism

3. Consider the following Bird and TestEagle classes.

```
class Bird {
        public void fly() {
                System.out.println("The bird is flying.");
        }
        public void fly(int height) {
                System.out.println("The bird is flying " + height + " feet high.");
        }
        public void fly(String name, int height) {
                System.out.println("The " + name + " is flying " + height + " feet
high.");
        }
}
```

```
class TestBird {
        public static void main(String[] args) {
                Bird myBird = new Bird();

                myBird.fly();
                myBird.fly(10000);
                myBird.fly("eagle", 10000);
        }
}
```

4. Consider the following Animal, Bird and TestBird classes.

```
class Animal {
        public void eat() {
                System.out.println("This animal eats insects.");
        }
}

class Bird extends Animal {

        public void eat() {
                System.out.println("This bird eats seeds.");
        }

}
```

```
class TestBird {
        public static void main(String[] args) {
                Animal myAnimal = new Animal();
                myAnimal.eat();

                Bird myBird = new Bird();
                myBird.eat();
        }
}
```

Constructors

5. Consider the following Employee class.

```java
public class Employee {

   int empId;
   String empName;

   //parameterized constructor with two parameters
   Employee(int id, String name){
       setEmpId(id);
       setEmpName(name);
   }
   void info(){
        System.out.println("Id: "+empId+" Name: "+empName);
   }

   void setEmpId(int id){
       this.empId = id;
   }

   void setEmpName(String name){
       this.empName = name;
   }

   public static void main(String args[]){
       Employee obj1 = new Employee(10245,"Chaitanya");
       Employee obj2 = new Employee(92232,"Negan");
       obj1.info();
       obj2.info();
   }
}
```

6. Consider the following Geek and TestGeek classes.

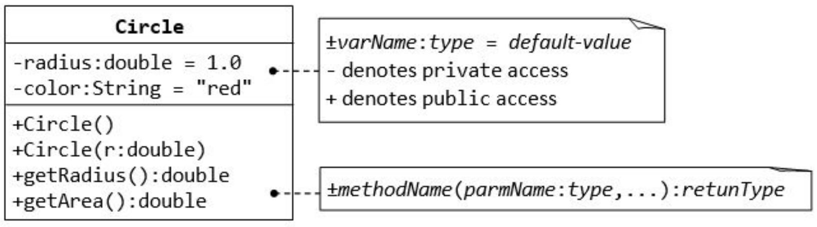
```java
import java.io.*;

class Geek
{
    // constructor with one argument
    Geek(String name)
    {
        System.out.println("Constructor with one " +
                      "argument - String : " + name);
    }

    // constructor with two arguments
    Geek(String name, int age)
    {

        System.out.println("Constructor with two arguments : " +
                " String and Integer : " + name + " "+ age);
    }

    // Constructor with one argument but with different
    // type than previous..
    Geek(long id)
    {
        System.out.println("Constructor with one argument : " +
                                            "Long : " + id);
    }
}
```

```
class TestGeek
{
    public static void main(String[] args)
    {
        // Creating the objects of the class named 'Geek'
        // by passing different arguments
        Geek geek2 = new Geek("Shikhar");
        Geek geek3 = new Geek("Dharmesh", 26);
        Geek geek4 = new Geek(325614567);
    }
}
```

B. Coding in Java

For each of the following class diagrams, write the code in Java following Object Oriented Programming concepts. An example is shown to you below to get a basic idea about the task.

Example:
A class called **circle** is designed as shown in the following class diagram.



It contains:

- Two private instance variables: radius (of the type double) and color (of the type String), with default value of 1.0 and "red", respectively.
- Two *overloaded* constructors - a *default* constructor with no argument, and a constructor which takes a double argument for radius.
- Two public methods: getRadius() and getArea(), which return the radius and area of this instance, respectively.

The source codes for Circle.java is as follows:

```java
public class Circle {
   // private instance variable, not accessible from outside this class
   private double radius;
   private String color;

   // The default constructor with no argument.
   // It sets the radius and color to their default value.
   public Circle() {
      radius = 1.0;
      color = "red";
   }

   // 2nd constructor with given radius, but color default
   public Circle(double r) {
      radius = r;
      color = "red";
   }

   // A public method for retrieving the radius
   public double getRadius() {
     return radius;
   }

   // A public method for computing the area of circle
   public double getArea() {
      return radius*radius*Math.PI;
   }
}
```

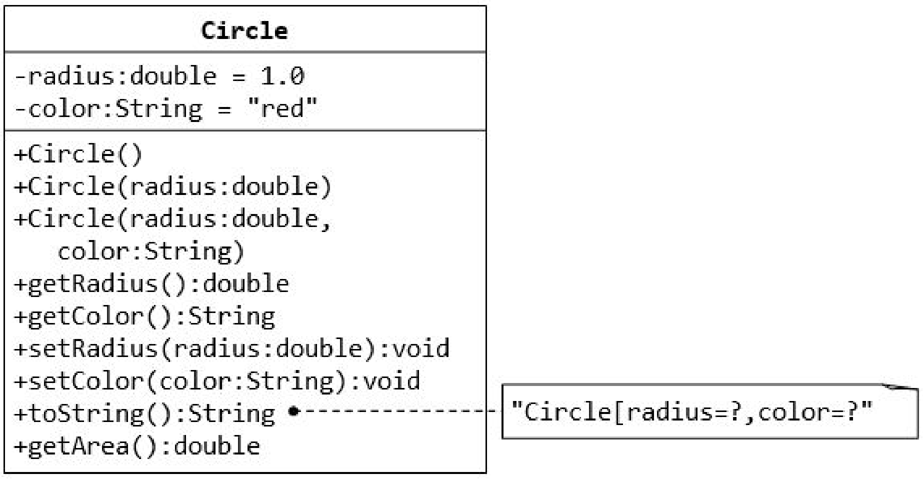
Compile "Circle.java". Can you run the Circle class? No. But why?

This Circle class does not have a main() method. Hence, it cannot be run directly. This Circle class is a “building block” and is meant to be used in another program.

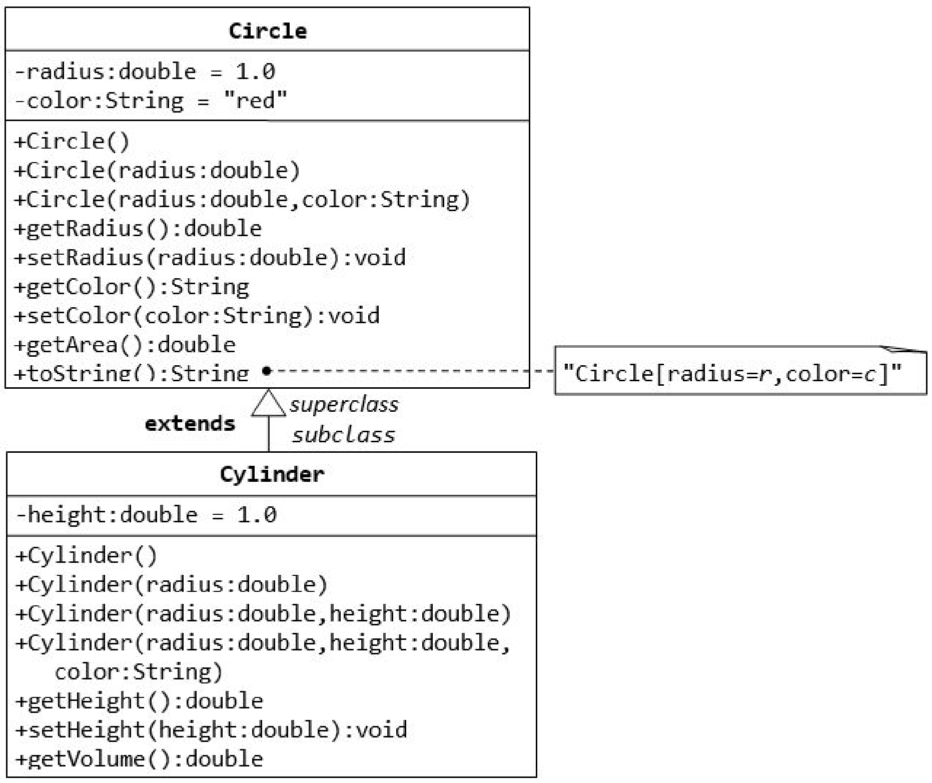
Let us write a *test program* called TestCircle (in another source file called TestCircle.java) which uses the Circle class, as follows:

```
public class TestCircle {
   public static void main(String[] args) {
      // Declare an instance of Circle class called c1.
      // Construct the instance c1 by invoking the "default" constructor
      // which sets its radius and color to their default value.
      Circle c1 = new Circle();
      // Invoke public methods on instance c1, via dot operator.
      System.out.println("The circle has radius of "
         + c1.getRadius() + " and area of " + c1.getArea());

      // Declare an instance of class circle called c2.
      // Construct the instance c2 by invoking the second constructor
      // with the given radius and default color.
      Circle c2 = new Circle(2.0);
      // Invoke public methods on instance c2, via dot operator.
      System.out.println("The circle has radius of "
         + c2.getRadius() + " and area of " + c2.getArea());
   }
}
```

1. Consider the following extended version of the Circle class.



2. Using the extended Circle class that you wrote above, consider the Cylinder class.



C. Solving Problems: the Object-Oriented way

Read the given passage and design a solution to the said problem using Object Oriented Programing. Both a class diagram and source code is necessary.

A class called MyTime, which models a time instance,as explained below.

It contains the following private instance variables:

- hour: between 0 to 23.
- minute: between 0 to 59.
- Second: between 0 to 59.

You are required to perform *input validation*.

It contains the following public methods:

- setTime(int hour, int minute, int second): It shall check if the given hour, minute and second are valid before setting the instance variables.
  (Advanced: Otherwise, it shall throw an IllegalArgumentException with the message "Invalid hour, minute, or second!".)
- Setters setHour(int hour), setMinute(int minute), setSecond(int second): It shall check if the parameters are valid, similar to the above.
- Getters getHour(), getMinute(), getSecond().
- toString(): returns "HH:MM:SS".

- nextSecond(): Update this instance to the next second and return this instance. Take note that the nextSecond() of 23:59:59 is 00:00:00.
- nextMinute(), nextHour(), previousSecond(), previousMinute(), previousHour(): similar to the above.

Write the code for the MyTime class. Also write a test driver (called TestMyTime) to test all the public methods defined in the MyTime class.

* * * * * * * *