CS 501B – Introduction to JAVA Programming
Fall 2023 Semester
Due: 10/20/2023 Friday at 11:59 PM

Instructions:

1. You are not allowed to use any package/library unless stated otherwise.
2. There is no skeleton given for this assignment.
3. You are required to follow questions and additional requirements.
4. We will be testing your code on hidden test cases.
5. Test your code with your own test cases.
6. Please comment your name and CWID in the first two lines of every **`.java`** file.
7. Add comments in your code about what you are doing.
8. You are required to zip only **`.java`** file as FirstName_LastName_Assignment#.zip (Ex: Roushan_Kumar_Assignment5.zip).
9. This assignment covers topics from week 7.
10. Students are not allowed to collaborate with classmates and any other people outside. All work must be done individually. Any work having evidence of showing academic dishonesty violation is subjected to zero for the assignment.

Penalty:

1. 10 marks will be deducted for invalid format of file / assignment submission.
2. If you submit after the due date then 10 marks will be deducted for every day after the due day.
3. If you submit an assignment after two weeks from the due date then you will get zero marks.
4. You will receive zero, if code doesn't compile / run.

Questions:
Each question carries **25** points and total points is **100.**

1. To assess your understanding of interfaces and abstract classes, encapsulation, and inheritance in Java through the design of a simple geometric shape system.

   Create Interface named **`Colorable`**:

   Include the following method:
   - **`String fillColor(String color);`**
     - Description: Assigns and returns the color for a shape.

Create an Abstract Java class named **`Shape`**:

Fields:

- **`String color`**: The color of the shape.

Include:

- Constructor with parameter: Initializes the **`color`** with the given value.

Implement the following methods:

- **`public abstract double area();`** Calculates and returns the area of the shape.

Create a Java class named **`Circle`** that extends **`Shape`** and implements **`Colorable`**:

Fields:

- **`double radius`**: The radius of the circle.

Include:

- No-argument constructor: Initializes the **`radius`** to 1.0 and **`color`** to "white".
- Constructor with parameters: Initializes the **`radius`** and **`color`** with given values.

Implement the following methods:

- **`public double area();`** Calculates and returns the area of the circle. Take π as 3.14.
- **`public String fillColor(String color);`** Assigns and returns the color of the circle.

Create a Java class named **`Rectangle`** that extends **`Shape`** and implements **`Colorable`**:

Fields:

- **`double length, width;`** The dimensions of the rectangle**.**

Include:

- No-argument constructor: Initializes the **`length`** and **`width`** to 1.0 and **`color`** to "white".
- Constructor with parameters: Initializes the **`length`**, **`width`**, and **`color`** with given values..

Implement the following methods:

- **`public double area();`** Calculates and returns the area of the rectangle.

- **`public String fillColor(String color);`** Assigns and returns the color of the rectangle.

2. Evaluate your proficiency in designing and implementing abstract classes and interfaces in Java, applying principles of polymorphism, encapsulation, and inheritance.

   Create Interface named **`Machine`**:

   Include the following method:
   - **`boolean start();`**
     - Description: Starts the machine operation. Returns **`true`** if the machine starts successfully, or **`false`** if it's already running..
   - **`boolean stop();`**
     - Description: Stops the machine operation. Returns **`true`** if the machine stops successfully, or **`false`** if it's already stopped.

   Create an Abstract Java class named **`ElectricMachine`** that implements **`Machine`**:

   Fields:
   - **`boolean powerStatus`**: Indicates whether the machine is powered on (true) or off (false)

   Include:
   - No-argument constructor: Initializes **`powerStatus`** to **`false`**.

   Implement the following methods:
   - **`public boolean start();`** Starts the machine operation. Returns **`true`** if the machine starts successfully, or **`false`** if it's already running.
   - **`public boolean stop();`** Stops the machine operation. Returns **`true`** if the machine stops successfully, or **`false`** if it's already stopped.

   Create a Java class named **`Computer`** that extends **`ElectricMachine`**:

   Fields:
   - **`String os`**: The operating system of the computer.

   Include:
   - No-argument constructor

- Constructor with parameters: Initializes the `**os**` with given values and sets `**powerStatus**` to `**false**`.

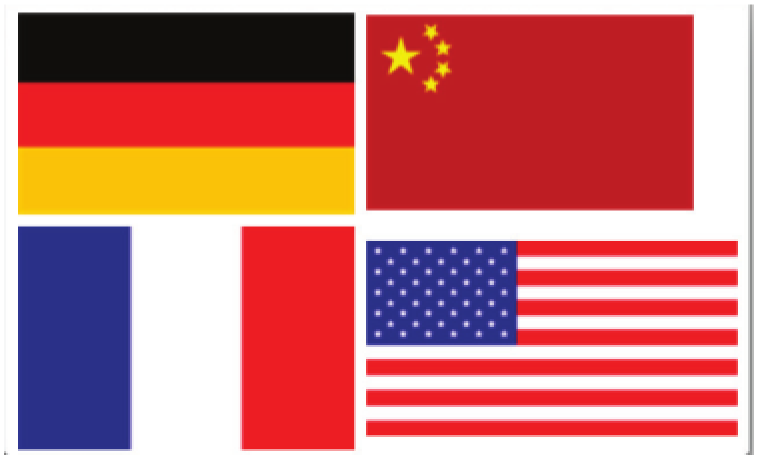
- Implement the `**start()**` and `**stop()**` methods, ensuring the `**powerStatus**` reflects the operational status of the computer.
    - Description: `**start()**` starts the computer if it's powered off and returns true, or returns `**false**` if it's already running. `**stop()**` stops the computer if it's running and returns `**true**`, or returns `**false**` if it's already stopped.

Create a Java class named `**Refrigerator**` that extends `**ElectricMachine**`:

Fields:
- `**double temperature**`: The current temperature of the refrigerator.

Include:
- No-argument constructor
- Constructor with parameters: Initializes the `**temperature**` with given values and sets `**powerStatus**` to `**false**`.
- Implement the `**start()**` and `**stop()**` methods, ensuring the `**powerStatus**` reflects the operational status of the refrigerator.
    - Description: `**start()**` starts the refrigerator if it's powered off and returns `**true**`, or returns `**false**` if it's already running. `**stop()**` stops the refrigerator if it's running and returns `**true**`, or returns `**false**` if it's already stopped.

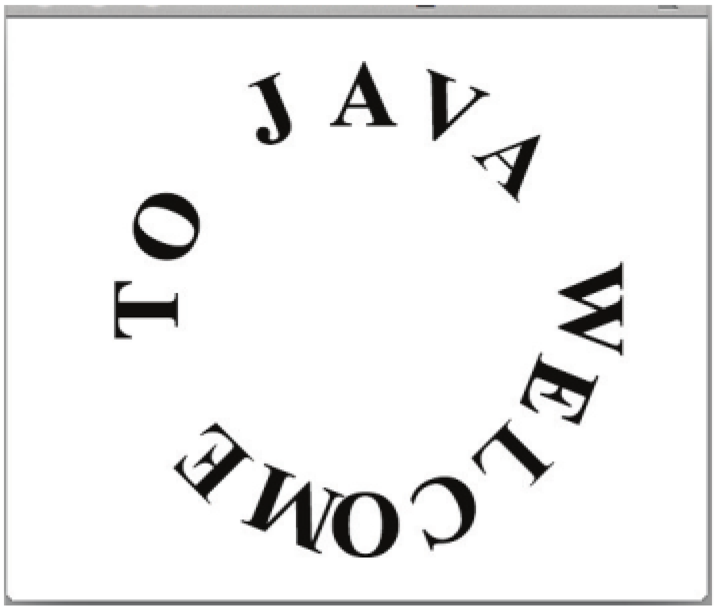
3. Write a program that displays four images in a grid pane, as shown in Figure below.



File name **:** `Grid.java`

```
// Do coding in below class
class Grid {
          }
```

4. Write a program that displays a string "Welcome to Java" around the circle, as shown in below Figure. (Hint: You need to display each character in the right location with appropriate rotation using a loop)



.

File name **:** `Welcome.java`

```
// Do coding in below class
class Welcome {
          }
```

Additional Requirement for All Questions:

- Draw the UML diagram for each class then implement the class. **Submit** a UML diagram in PDF format named **`class_name_UML.pdf`** (example for question 1: `**Rectangle_UML.pdf**`). `**[** This is only for question 1 and question 2 **]**`

- Create each class including interface and abstract of every question in a separate **`.java`** file. Name of the file should be **`class_name.java` / `interface_name.java` / `abstract_name.java`** and the name of the class/interface/abstract is already mentioned in each question.

  Ex: for question 1 of class **`Rectangle`** , file name will be **`Rectangle.java`** and class name will be **`Rectangle`.**
  Follow the same convention for all classes in every question.

- Write test cases for each method of each concrete class of every question in a separate **`.java`** test file. Print the input and output in each test case. Sample for question 1 of class `**Rectangle**` is given below for this.**`[** This is only for question 1 and question 2 **]`**

  File Name : `class_nameTest.java` -> `RectangleTest.java`
  ClassName : `class_nameTest` -> `RectangleTest`

  RectangleTest.java

  ```
  RectangleTest {
          public static void main(String[] args){
                          // Test Case
                  }
  }
  ```

  Follow the same convention for all other classes of every question for writing your own test case and **submit** it also.