CS 501B – Introduction to JAVA Programming
Fall 2023 Semester
Due: 10/13/2023 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_Assignment4.zip).
9. This assignment covers topics from week 6.
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. Create a Java class named **`Animal`** with the following specifications:

   Fields:
   - **`int age`**: The age of the animal.
   - **`double weight`**: The weight of the animal.

   Include:
   - No-argument constructor: Initializes the **`age`** to 0 and **`weight`** to 0.0.
   - Constructor with parameters: Initializes the **`age`** and **`weight`** with the given values.
   - Method **`public String eat()`**: Returns the string "The animal is eating".
   - Method **`public String move()`**: Returns the string "The animal is moving".

Next, create a subclass named `**Bird**` that inherits from `**Animal**` with the following additional method:

- No-argument constructor: Calls the no-argument constructor of the `**Animal**` class.
- Method `**public String fly()**`: Returns the string “The bird is flying”.
  Override the `**move()**` method in `**Bird**` to return “The bird is flying”.

2. Create a Java interface named `**Shape**` with the following method:
   - `**public double area()**`: Calculates and returns the area of the shape.

   Implement the `**Shape**` interface in two classes: `**Circle**` and `**Rectangle**`.

   For `**Circle**`:
   - Field: `**double radius**`.
   - No-argument constructor: Initializes the `**radius**` to 1.0.
   - Constructor with parameters: Initializes the `**radius**` with the given value.
   - Method `**public double area()**`: Calculates and returns the area of the circle.Take `**π**` as `**Math.*PI***`.

   For `**Rectangle**`:
   - Fields: `**double height**` and `**double width**`.
   - No-argument constructor: Initializes the `**height**` and `**width**` to 1.0.
   - Constructor with parameters: Initializes the `**height**` and `**width**` with the given values.
   - Method `**public double area()**`: Calculates and returns the area of the rectangle.

3. Create a Java class named `**ExceptionHandling**` with the following specifications:

   Fields:
   - `**int number`:** To store the user-inputted integer.

   Include:
   - No-argument constructor: Initializes `**number**` to 0.
   - Method `**public String validateNumber(String input)**`: This method should attempt to parse the input string as an integer using `**Integer.parseInt(input)**`. If the input string cannot be parsed as an integer (i.e., it’s not a valid integer representation), it should catch a `**NumberFormatException**` and return a string saying "Please enter a valid integer.". If the parsed integer is negative, it should throw and catch a custom exception named `**NegativeNumberException**` (which you also need to create) and return an appropriate error string like

"Negative numbers are not allowed.". For a valid non-negative integer, it should store the value in `**number**` and return a success message like "Number accepted: " followed by the entered number (Ex : "**Number accepted: 3**" for input 3).

Instructions:

- You need to create a `**NegativeNumberException**` class that extends `**Exception**` and initializes with a custom error message**.**
- Implement the `**validateNumber(String input)**` method in the `**ExceptionHandling**` class to include the described functionality.

4. Create a Java class named `**DataProcessor**` with the following specifications:

   Include:
   - **No-argument constructor.**

   Implement the following methods:

   - `**public String writeData()**`: This method should create a file named `**Java_Program.txt**` if it does not exist. Then, write 100 integers created randomly into the file using text I/O. Integers should be separated by spaces in the file. Return a string indicating whether the data was written successfully, such as "Data written to Java_Program.txt successfully.".

   - `**public int[] readAndSortData()**`: This method should read the data back from `**Java_Program.txt**`, sort the integers in increasing order, and then return the sorted integers as an array of `**int**`.

   Instructions:
   - Ensure to handle any `**IOException**` that might occur during the writing and reading processes, and use try-with-resources or finally blocks to close your files properly.
   - The random integers should be generated in a range that you define.

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: `**Animal_UML.pdf**`).

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

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

- Write test cases for each method of each 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 **`Animal`** is given below for this.

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

  AnimalTest.java

```
AnimalTest {
        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.