Assignment 2

Object Oriented Design and Development 2 (15%)

DUE DATE

Session 12

OBJECTIVES

- Implement class methods.
- Implement the visibility of declarations.
- Design and develop method overloading.
- Declare, initialize, and access individual members of arrays in a Java program.
- Implement the public and private modifiers to control access to a class's variables and methods.
- Implement inheritance to develop new classes based on existing classes.
- Design and implement polymorphism to make systems extensible and maintainable.
- Implement exception handling in a Java program using the try, throw, catch, and finally code blocks.

DESCRIPTION

For each of the following problems:

- Start by developing the program logic in form of Flowchart, or pseudocode algorithm.
- Based on the program logic, implement each solution in Java code.

INSTRUCTIONS

1. **Using the Enhanced for Statement**: Write an application that uses an enhanced for statement to sum the `double` values passed by the command-line arguments. [Hint: Use the `static` method `parseDouble` of class `Double` to convert a `String` to a `double` value.]

2. **Sieve of Eratosthenes:** A prime number is any integer greater than 1 that's evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:
   a) Create a primitive-type `boolean` array with all elements initialized to `true`. Array elements with prime

indices will remain `true`. All other array elements will eventually be set to `false`.

b) Starting with array index 2, determine whether a given element is `true`. If so, loop through the remainder of the array and set to `false` every element whose index is a multiple of the index for the element with value true. Then continue the process with the next element with value true. For array index 2, all elements beyond element 2 in the array that have indices which are multiples of 2 (indices 4, 6, 8, 10, etc.) will be set to `false`; for array index 3, all elements beyond element 3 in the array that have indices which are multiples of 3 (indices 6, 9, 12, 15, etc.) will be set to `false`; and so on.

When this process completes, the array elements that are still true indicate that the index is a prime number. These indices can be displayed. Write an application that uses an array of 1,000 elements to determine and display the prime numbers between 2 and 999. Ignore elements 0 and 1. Display the list of prime numbers in the console.

3. **Enhancing Class Date:** Modify class `Date` of Fig. 8.7 to perform error checking on the initializer values for variables `month`, `day`, and `year` (currently it validates only the month and day). Provide a method `nextDay` to increment the day by one. Write a program that tests method `nextDay` in a loop that prints the date during each iteration to illustrate that the method works correctly. Test the following cases:

   a) incrementing into the next month and

   b) incrementing into the next year.**(Enhancing Class Time2)** Modify class `Time2` of Fig. 8.5 to include a tick method that increments the time stored in a `Time2` object by one second. Provide method `incrementMinute` to increment the minute by one and method `incrementHour` to increment the hour by one. Write a program that tests the tick method, the `incrementMinute` method and the `incrementHour` method to ensure that they work correctly. Be sure to test the following cases:

   a) incrementing into the next minute,

   b) incrementing into the next hour and

   c) incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

4. **Date and Time Class:** Create class `DateAndTime` that combines the modified `Time2` class and the modified `Date` class of the previous question 3. Modify method `incrementHour` to call method `nextDay` if the time is incremented into the next day. Modify methods `toString` and `toUniversalString` to output the date in addition to the time. Write a program to test the new class `Date-AndTime`. Specifically, test incrementing the time to the next day.

5. **Date and Time Class Inheritance:** In question 4, you combined Time2 and Date to make the **DateAndTime** class. In this question, create a class called Time3 and Date3. Copy the contents in the modified Time2 class to Time3 class and copy the contents in the modified Date class to Date3 class.

   a) Time3 class inherits from the class Date3.

   b) Call superclass Date3's toString method in Time3's toString method to display both date and time.

c) Call Date3's constructor in the Time3 constructor to initialize the date and then initialize the time.

d) In the incrementHour method of Time3 class, every 24 hours, it should call the super class Date3 classe's nextDay method.

e) Initialize the tester class Clock.

- Call toString method of the Time3 class.
- Clock class will call incrementMinute class until it passes 24 hours.
- Call toString method of the Time3 class.

6. ***`Shape` Hierarchy:*** Implement the `Shape` hierarchy shown in **Fig. 9.3**.

   Each `TwoDimensionalShape` should contain method `getArea` to calculate the area of the two-dimensional shape. Each `ThreeDimensionalShape` should have methods `getArea` and `getVolume` to calculate the surface area and volume, respectively, of the three- dimensional shape. Create a program that uses an array of `Shape` references to objects of each concrete class in the hierarchy. The program should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a `TwoDimensionalShape` or a `ThreeDimensionalShape`. If it's a `TwoDimensionalShape`, display its area. If it's a `ThreeDimensionalShape`, display its area and volume.

7. ***`Payroll` System Modification:*** Modify the payroll system of **Figs. 10.4**–**10.9** to include an additional `Employee` subclass `PieceWorker` that represents an employee whose pay is based on the number of pieces of merchandise produced.

   Class `PieceWorker` should contain `private` instance variables `wage` (to store the employee's wage per piece) and `pieces` (to store the number of pieces produced). Provide a concrete implementation of method `earnings` in class `PieceWorker` that calculates the employee's earnings by multiplying the number of pieces produced by the wage per piece. Create an array of `Employee` variables to store references to objects of each concrete class in the new `Employee` hierarchy. For each `Employee`, display its `String` representation and earnings.

8. **Exception Handler (Chapter 11):** In question 2, the application was to determine a prime number up to 1000.

   a) Modify the application to separate the calculating class and the tester class. Name each Prime and `PrimeTester`.

   b) Features from question 2 will go into the `calculatePrime` method in the Prime class. `calculatePrime` method will take an int parameter to determine when the calculation will stop. For example, if the passed in value is 591, it will never check if 592 is prime or not.

   c) `calculatePrime` method will throw exceptions when the parameter is a negative integer or is over

10000.

d) `calculatePrime` will return its result as an array.

e) `PrimeTester` will `prompt` the user to input the parameter to pass into `calculatePrime` method.

f) `PrimeTester` has to have a try-catch to catch the exceptions from the `calculatePrime` method.

g) Display the list of prime numbers in the console.

SUBMISSION INSTRUCTIONS

Your submission must include a Word document with the logic for each problem in this assignment as well as each problem's source code. Zip your final project and submit the zip file.

Work must be submitted in the correct file type and be properly labelled as per the College naming convention:
NAME_COURSE_ASSIGNMENT. E.g. XuXiaLing_FM50D_A01.

GRADING CRITERIA

Assignment Value: **15%**

Grading Criteria	Grading
Problem 1	/10
Problem 2	/10
Problem 3	/10
Problem 4	/10
Problem 5	/10
Problem 6	/20
Problem 7	/20
Problem 8	/10
TOTAL	**/100**