COSC1337 Programming Fundamentals II /ITSE2357 Advanced Object-Oriented Programming

The submission opens two days before the due day, and closes right at the due time on due day. One submission is allowed per project. It is students' responsibilities to submit timely and correctly in order to get credits. Students MUST start each project when it is first discussed in class in order to complete it on time. All projects are individual projects. Project due days may be changed to better align with lectures.

If one of the following is true, not credits will be given:

- The project is late.
- The algorithm or class design is missing.
- The project has syntax errors.
- There are no comments (Javadoc format required) at all in your code.
- Wrong files are submitted.
- **A project is a copy and modification of another student's project. (Both will receive 0.)**

Files to be submitted through Canvas:

- The UML design (UML class diagram)
- All source codes and driver programs
- All supporting files if any

Software Development Project Rubric: (Design is worth 20%; the rest is worth 80 %.)
Analysis: There will be no credit if the software doesn't meet customer specification at all.

- Does the software meet the exact customer specification?
- Does the software read the exact input data and display the exact output data as they are shown in sample runs?
- Does each class include all corresponding data and functionalities?

Design: There will be no credit if the design is missing.

- Is the design (a UML class diagram) an efficient solution?
- Is the design created correctly?

Code: There will be no credit if there are syntactic errors.

- Are there errors in the software?
- Are code conventions and name conventions followed?
- Does the software use the minimum computer resource (computer memory and processing time)?
- Is the software reusable?

Debug:

- Are there bugs in the software?

Documentation: There will be no credit if comments are not included.

- Are there enough comments included in the software?
- Class comments must be included before a class header.
- Method comments must be included before a method header.
- More comments must be included inside each method.
- All comments must be written in Javadoc format.

Project 5 Exceptions   A Triangle – A special geometric object

Part I: Normal Execution Flow

Geometric object:

A geometric object is an object that has a color (white by default), and a fill (false by default). A geometric object is modeled in the following class.

```
private String color;
private boolean filled;
```

A default geometric object has white color; it is not filled.

A geometric object can be created with any color and filled of true or false. Calculating area and perimeter of a geometric object is a common behavior; therefore include two abstract methods that are completed by the sub classes.

```
public abstract double getArea();
public abstract double getPerimeter();
```

Complete this class with all of other methods.

Triangle Object:

Design a class named Triangle as a subclass of class *GeometricObject*. A triangle can be created as default geometric object with all three sides are length of 1. A triangle can be created as a geometric object with a color, a fill of true or false, and specified integer values for *sideOne*, *sideTwo*, and *sideThree*. Add other methods such as *getters/setters*, *toString*, and *equals*. Complete abstract methods *getArea()* and *getPerimeter().*

Part I: Exception Execution Flow

A program can be separated into a normal execution flow and an exception execution flow. Codes for an exception execution flow must be included in a program. A program can handle Java exceptions and/or user-defined exceptions.

In a triangle, the sum of any two sides is greater than the third side. The Triangle class must adhere to this rule. If a triangle is created with sides that violate the rule, an exception such as *IllegalTriangelException* should be thrown. Create an *IllegalTriangelException* class as a subclass of *java.lang.Exception* class. It has a constructor that constructs a new exception with the specified message

```
public class IllegalTriangleException extends Exception  {
      public IllegalTriangleException(String message) { …  }
}
```

Now it is time to add exception execution flow in *Triangle* class to handle illegal triangle exceptions. Check each method, include the codes that may throw exceptions in a try block, and then add exception handlers. Always add an generic exception handler at the end to catch any exceptions.

In *Triangle* class, if a method forms a triangle or change any side of a triangle, this method may throw an illegal triangle exception. Therefore, illegal triangle exceptions should be caught in the method.

Catch and Handle Exceptions:

An exception can be

- caught and handled where it occurs.
- propagated to be handled in next method in calling hierarchy.

In this project, catch and handle *IllegalTriangleException* and others where they occur.

How to catch and handle an exception in the same method?

In the following constructor, if three sides don't construct a legal triangle, an *IllegalTriangelException* object should be thrown, and it is caught in the first exception handler.

```
public Triangle(int sideOne, int sideTwo, int sideThree){
    try{
         if a legal triangle can be formed with sideOne, sideTwo, and sideThree
             Assign three sides
         else
             Throw an IllegalTriangelException object using a throw statement.
             This object is caught and handled by the matching exception handler.
   }
   catch (IllegalTriangelException ex){
       Display stack trace
   }
   catch (Exception ex){
       All other exceptions handler
   }
   finally{
        Cleanup code if any
   }
}
```

Driver Program:

Store integers, three as a group, in a file. Scan each group of integers to form triangle. Create at least ten legal triangles, and put them into an array list.

Method *nextInt* can throw any one of the following three Java exceptions in `main` method.

InputMismatchException - if the next token does not match the *Integer* regular expression, or is out of range
NoSuchElementException - if input is exhausted
IllegalStateException - if this scanner is closed

> `public int nextInt()`
> Scans the next token of the input as an `int`.
> An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.
> Returns:
> the `int` scanned from the input
> Throws:
> `InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range
>
> `NoSuchElementException` - if input is exhausted
> `IllegalStateException` - if this scanner is closed

Handle all three exceptions thrown by *nextInt* in main method where they occur. The exceptions handlers can simply display the stack trace.

Note:
In this project, exceptions are caught and handled where they occur. Here is information on propagating an exception:

How to propagate an exception?

An exception can be thrown, and a method further up the call stack will handle it. Throwing an exception is better when there is not enough information to handle the exception properly by the time the code is written. Or the exception can be handled differently by different users of this class.

To specify a method can throw exceptions, add a *throws* clause to the method header for the method.

A *throws* clause consists of reserved word *throws* and the data type of the exception. Add

```
                  throws IllegalTriangleException
```

to the end of the method header to specify that this method can throw an `IllegalTriangleException` object.

If a method specifies that it can throw exceptions,   a *throw* statement must be added the exception is thrown. A *throw* statement consists of reserved word *throw* and an object of the exception class.

```
                  throw new IllegalTriangleException(“Illegal Triangle: …”)
```

The following codes show the entire example:

```
public Triangle(int sideOne, int sideTwo, int sideThree)
                  throws IllegalTriangleException{
      if a legal triangle can be formed with sideOne, sideTwo, and sideThree{
            //make a triangle/Assign three sides
      }else{
            //Throw an IllegalTriangelException object using a throw statement.
            //The object is passed to and handled by a method further up the call stack
                 throw new IllegalTriangleException(“Illegal Triangle: …”)
      }
}
```

Whenever a triangle is initially created or recreated (setter), *IllegalTriangleException* may be occurred. Throw an *IllegalTriangleException* object from constructors and setters, and let a method further up the call stack handle it