CSCI 470/502 - Assignment 1 - Spring 2023

This assignment practices writing Java classes, adding variables and methods to the class, as well as exception throwing and handling.

Study the Line class and test program given below and compile and run it. Then see the "Enhancements" below (after the code) and add the features described there to this program. Compile and test the new version and hand it in as Assignment 1. Update the documentation to reflect the changes you have made. A nice way to do this is to add a section after the existing documentation, headed "Code Changes and Enhancements 01/25/2023". You should also include your name.

```
/*************************************************************
This program demonstrates a simple "Line" class.
Here, a Line class is defined with its properties and
interface (i.e., its methods).
A main class then creates instances of this Line class
and calls on the methods to demonstrate its behavior.
*************************************************************/
import java.io.*;

public class Line
{
  private int x1, y1, x2, y2;     //coordinates of the line

  //Constructor
  //Receives 4 integers which are the Line's start and end points.
  public Line(int left, int top, int right, int bottom)
  {
    // each of these validates its argument - see below.
    setLeft(left);
    setTop(top);
    setRight(right);
    setBottom(bottom);
  } // end constructor

  //***********************************
  //method draw() calls another method called drawLine(),
  //which is assumed to be a graphics primitive on the
  //system.  However, since this program will be
  //run in console mode, a text description of the Line
  //will be displayed.
  //
  public void draw()
  {
    drawLine(x1, y1, x2, y2);
  }

  //***********************************
  //method drawLine() simulates drawing of a line for console mode.
  //It should describe all the important attributes of the line.
  //In a graphics mode program, we would delete this and use the
  //system's Graphics library drawLine().
  //
  private void drawLine(int x1, int y1, int x2, int y2)
  {
    System.out.println("Draw a line from x of " + x1 + " and y of " + y1);
    System.out.println("to x of " + x2 + " and y of " + y2 + "\n");
  }
```

```
//***********************************
//Method setLine() allows user to change the points of the
//already existing Line.
//
public void setLine(int left, int top, int right, int bottom)
{
  setLeft(left);
  setTop(top);
  setRight(right);
  setBottom(bottom);
}

// -- the individual setXXXX methods that prevent
//    any line's coordinate from being offscreen.
//    In the event of an invalid (offscreen) value,
//    that value is (silently) set to 0.
//**************************
public void setLeft(int left)
{
  if (left < 0 || left > 639)
    x1 = 0;
  else
    x1 = left;
}
//**************************
public void setTop(int top)
{
  if (top < 0 || top > 479)
    y1 = 0;
  else
    y1 = top;
}
//**************************
public void setRight(int right)
{
  if (right > 639 || right < 0)
    x2 = 0;
  else
    x2 = right;
}
//**************************
public void setBottom(int bottom)
{
  if (bottom > 479 || bottom < 0)
    y2 = 0;
  else
    y2 = bottom;
}
//Now for some "get" Access methods to get individual values
//**************************
public int getLeft()
{
  return x1;
}

//**************************
public int getTop()
{
  return y1;
}

//**************************
```

```
  public int getRight()
  {
    return x2;
  }

  //*************************
  public int getBottom()
  {
    return y2;
  }
} // end class Line

/*************************************************************
 Now we will define a class with main() where execution will begin. It is this
 class, and this code, that will create instances of the Line and call its
 methods.
 As a test module, this code would be improved with additional
 System.out.println() statements that explain what is being attempted and
 what the results should be, for example:
 "About to change l1 to an invalid value and then redraw it.  Line position
  should not change: "
*/
//*************************************************************
class TestLine
{
  public static void main(String args[])
  {
    Line l1 = null, l2 = null;   //declare 2 instances of Line class
    //create 1 Line object
    l1 = new Line (10, 10, 100, 100);
    //draw it
    l1.draw();
    //change start point with valid values
    l1.setLine(5, 5, l1.getRight(), l1.getBottom());
    //draw it again with new start point
    l1.draw();
    //try to change left (x1) to an illegal value
    l1.setLeft(3000);
    //draw the line...x1 should now be zero
    l1.draw();
    //create a second Line instance, or object
    l2 = new Line(100, 100, 400, 400);
    //draw 2nd line
    l2.draw();
    //set a new valid bottom for line 2
    l2.setBottom(479);
    //draw 2nd line again
    l2.draw();
  } // end of main
}  // end class TestLine
```

CSCI 470/502 Assignment 1: Line Class Enhancement

Modify the above Java program to accomplish the following:

1) Add additional methods and variables to the Line class to implement the following behaviors:

a. Add an instance variable to hold the width (thickness) of the line. You may implement the width as an int. Then add 2 access methods for this color variable: a) set the width for a Line. b) get a Line's width.
b. Add an instance variable to hold the color of the line. For this assignment, the data type of the color could be an int value or a String (Later we will talk about a Color class in graphics but it is not required in this assignment). Then add 2 access methods for this color variable: a) set the color for a Line. b) get a Line's color.
c. Calculate and return (get) the length of a Line. Add a method for calculating the length based on its coordinates. The method returns a double. The formula is sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)). You can use the java.lang.Math class for calculating the square root (The method is sqrt(), you need to check the JDKAPI documentation for its usage). Do ***not*** add an instance variable to hold the length.

Add code to *main()* of TestLine to test these new features.

2) Write another class and add another constructor for the Line class.

Write a class called *TwoDPoint*. It is very simple: It has 2 ***public*** data members called x and y and a constructor that accepts two integer arguments and stores them in x and y. That's all there is to the class. It has no other methods. You can either add this class to the same source file (e.g. Line.java, in which case this class TwoDPoint can't be public) or save it in its own file called TwoDPoint.java.

Add another constructor for Line that will accept 2 *TwoDPoint* objects instead of 4 ints.

This second Line constructor should simply call the first constructor (the one that accepts 4 ints). So extract the x and y ints from each TwoDPoint and send them on to the other constructor. Keyword "this" can be used calling one constructor from another. Find the details in the book.

Add code to *main()* to test this new constructor. You need only test a successful case here.

3) Add exception creation/throwing, passing and handling.

Instead of setting a coordinate to zero when an invalid value is encountered, now have the appropriate code in the Line class detect invalid values and *throw* or *pass* an Exception that will be caught by the *calling code* (i.e. the code that calls the method throwing the exception).

In particular, the setters in Line class will detect the problem and *throw* the exception (a generic Exception object with message passed in to its constructor). The constructor(s) will *pass* the exception to the calling code. The calling codes will *handle* the exception. -- In the case of this assignment, the calling codes are the statements in the main() of TestLine.

Modify the calling code in the test program (i.e. TestLine) to catch the exception and handle it:

- when trying to *alter* an existing Line's position. In this case, simply display the error message and state that the Line was not moved (but you can still use the Line object later, since it already exists).
- when *creating* a new Line. In this case, if the constructor fails, the calling (catching) code should just display the message about the failure and terminate the program (*System.exit(0);*). Test this feature ***last*** in the program, since nothing more will execute.

You should structure your code so that every *try* block attempt one or a few related exception-raising method calls. Do not put too many tests in one try block. – Because when there is an exception caught, the subsequent statements in the same block will not be executed. When that is *not* what you want, move those subsequent statements out of the try block.

The output of the program should have informational messages after (and/or before) each operation so the user knows what is being attempted and what the result was. Reading the output from the program should make it clear to the reader what the sequence of operations was and what happened. Make use of information (which you should supply to every thrown exception) in each exception object's message. You will lose some credit if this is done poorly.

Submit one version of the modified program (.java files) which implements and tests all the features mentioned above via Blackboard. Remember to apply applicable coding, formatting, and documentation standards!

IMPORTANT

Documentation - Read this carefully and follow these guidelines

Programs must be neatly and consistently formatted and documented. Write your code for human readers who cannot read your mind. If someone has to ask you "what does this method do?" or "what does this variable represent?" you have probably failed to do this well. Here are a few things we will look for:

- stuff that does nothing or is not needed or is redundant
- code where you tried something but aren't using it anymore but it's still there (and maybe commented out)
- documentation from another program or a previous version of the program that is irrelevant or just plain wrong for this program
- the same code in more than one place - usually should be a method and called from several places
- inconsistent indenting: if you indent {} by 2 spaces in one place, indent by 2 everywhere.
- don't indent too much. 2 - 4 spaces is plenty. Don't use tabs.
- lack of spaces after operands (usually) and commas in argument lists (always)
- *who* wrote this pgm, and *when*, and *what* does it do. The "how does it work" is usually part of detailed doc at the method level, not the overall program documentation. If "how" is particularly import, include it as a "Programming Note" rather than as part of the top-level description.
- meaningful variable names (except for loop counter vars, etc.)
- consistent naming convention:
- blank lines between methods and other logical divisions
- explanations of non-obvious code
- a line or two describing the purpose of methods
- use standard Java capitalization: underscores are rarely used; rather, each important word in a name is capitalized, but while class names start with a capital letter, variables and method names do not.

We *really* do not want to have to remind you of these things. Your program is not finished until you have addressed these points.

In general, prepare your source code for publication in a poetry magazine. It should look that good. Read the document called "Java Coding and Documentation Style Guidelines" several times before you submit your final program for inspection grading. These are the standards that will be followed in grading ***all of your programs*** this semester.