CSE1IOO/CSE4IOO Semester 1, 2020
Programming Assignment (40%)

Assessment: This assignment is worth 40% of the final mark for this subject. Do not be overwhelmed by the length of this file. A lot of its contents are examples, to help you understand the problem a bit better. Focus on one task at a time. It is a good idea to refer to the Java classes that have been provided with the assignment as a good starting point, after you have *read the entire specification carefully*.

Due Date: Monday 1 June 2020, at 10 am.

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime.

This is an individual Assignment. You are not permitted to work as a group when writing this assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Information Technology treats academic misconduct seriously. When it is detected, penalties are strictly imposed. Refer to the unit guide for further information and strategies you can use to avoid a charge of academic misconduct. All submissions will be electronically checked for plagiarism. ***You may be asked to give an interview through zoom and explain your code, if found suspicious by the plagiarism checker on the submission server. Failing to demonstrate proper understanding of your program may result in zero marks awarded for the assignment.***

Objectives: The general aims of this assignment are:

- To analyze a problem in an object-oriented manner, and then design and implement an object-oriented solution that conforms to given specifications
- To practise using inheritance in Java
- To practise file input and output in Java
- To make implementations more robust through mechanisms such as exception handling.

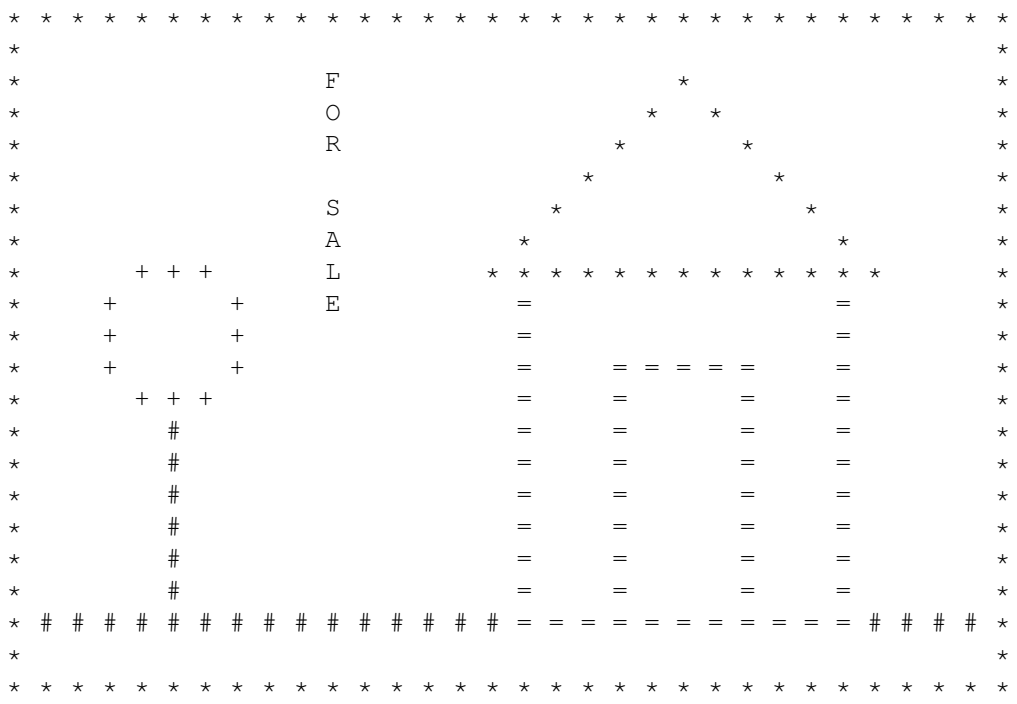
Submission Details and Marking Scheme: Instructions on how to submit electronic copies of your source code files from your latcs8 account and a marking scheme overview are given at the end. If you have not been able to complete a program that compiles and executes containing all functionality, then you should submit a program that compiles and executes with as much functionality as you have completed. (You may comment out code that does not compile.)

NOTE: While you are free to develop the code for this assignment on any operating system, your solution must run on the latcs8 system.

NOTE: You can use arrays or ArrayList or LinkedList of the Java API. If you use arrays, assume that we never have more than 50 shapes in a drawing.

Problem Description

In this assignment, you are to develop a basic text-based drawing application. A sample drawing is shown below.



It has

- Borders which form a rectangle around a drawing area of 20 by 30 (drawn with the asterisk character in this example)
- Two lines (the ground and the lamp post)
- One circle (the lamp)
- Two rectangles (the house walls and the door)
- One triangle (the roof)
- A string of text ("FOR SALE" displayed vertically)

Initially, your main task is to develop the classes whose instances are objects that make up a drawing. Essentially, they are classes that represent windows and shapes.

You will then develop a menu program that allows the user to progressively create a drawing (by adding shapes and removing shapes), save the definition of the drawing in a text file and read it back to reconstruct it.

Windows

Each drawing is a window on which shapes can be drawn. A window has, among other things,

- the **height**
- the **width**
- the **border character** that is used to draw the borders

The window's height and width represent the drawing area, ***excluding*** the borders. That is, a window of height 20 and width 30 has 20 x 30 = 600 cells each of which contains a blank or a non-blank character.

Shapes

There are five kinds of shapes that your programs can draw:

- Lines
- Rectangles
- Triangles
- Circles
- Texts (regarded as a kind of shape)

Each shape has

- a **base point** (that will be further explained as we describe the specific shapes), and
- a **display character** i.e. the character that is used to display the shape

Now, let us get into the detail characteristics of each shape. Read carefully, as these descriptions will help you design your classes correctly.

Lines

Consider, as an example, the line below:

Suppose the top point of the line is at position 10, 15. And suppose we take it to be the base point of the line. Then the line can be specified as follows:

- The **row position** of its base point, which is 10
- The **column position** of its base point, which is 15
- Its **length**, which is 4
- Its **row increment** is 1
- Its **column increment** is 1
- And its **drawing character**, which is '*' (i.e., the asterisk character)

The row increment 1 and column 1 signifies that the line goes down and goes to the right from the base point. For our drawing board, row increment and column increment can only take values -1, 0, or 1. In other words, a line can be horizontal, vertical or can incline at an angle of 45 degrees.

Rectangles

Consider, as an example, the rectangle below:

Suppose the top-left point is at position 10, 15. This point will be taken to be the base point. The line can be specified as follows:

- The **row position** of its base point, which is 10
- The **column position** of its base point, which is 15
- Its **height**, which is 3
- Its **width** is 6
- And its **drawing character**, which is '*'

Triangles

Our drawing board will draw only isosceles (at least, two equal sides) triangle. Consider, as an example, the triangle below:

Suppose the left-most point is at position 10, 15. This point, where the two equal sides meet, will be referred to as the *base point* of the triangle. The directed perpendicular line segment from the base point to the opposite side is referred to as the *height vector* of the triangle. The height vector can be in one of four possible directions. It can go:

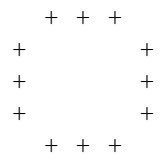
1. *Right* (as in the above example). We specify this direction by taking its row increment to be 0 and column increment to be 1.
2. *Left*, with row increment 0 and column increment -1.
3. *Up*, with row increment -1 and column increment 0.
4. *Down*, with row increment 1 and column increment 0.

The triangle in the above example can be specified as follows:

- The **row position** of its base point, which is 10
- The **column position** of its base point, which is 15
- Its **height**, the length of the height vector, which is 3
- Its **row increment** of the height vector, which is 0
- Its **column increment** of the height vector, which is 1
- And its **drawing character**, which is '*'

Circles

Consider the circle displayed below, which is a circle of radius 2:



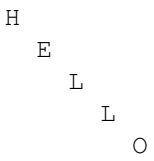
The base point of a circle is its center. A circle is specified by

- The **row position** of its base point (i.e., the center)
- The **column position** of its base point (i.e., the center)
- Its **radius**
- And its **drawing character**, which is '+'

For our drawing boards, we draw a circle by plotting 20 points on its circumference. When a circle is of small size, some of the 20 points overlap. Obviously, the display is often only a rather crude approximation of a circle. (Hint: In reference to a coordinate system whose origin is the center O of the circle, a point M on the circle has coordinates $x = R\ cos(\alpha)$ and $y = R\ sin(\alpha)$, where R is the radius and α is the angle between the x-axis to OM).

"Text Shapes"

Consider, as an example, the text display below:



Suppose letter 'H' is at position 10, 15. This is the base point. This "text shape" can be specified as follows:

- The **row position** of its base point, which is 10
- The **column position** of its base point, which is 15
- Its **text** itself, which is "HELLO"
- Its **row increment** is 1
- Its **column increment** is 1

The row increment 1 and column 1 signifies that the line goes down and goes to the right from the base point. For our drawing board, as for the lines, row increment and column increment of a text shape can take values -1, 0, or 1.

Signatures for the Classes

The Window class must have

- The constructor

  ```
  Window(int numberOfRows, int numberOfColumns, char borderCharacter)
  ```

  Please note, the numerOfRows and numberOfColumns are numbers of rows and columns respectively, excluding the border.

- The method to add a shape

  ```
  void addShape(Shape shape)
  ```

- The method to remove a shape

  ```
  void removeShape(String id)
  ```

- The method to display the drawing on the screen

  ```
  void display()
  ```

Row value increases from top to bottom, and column value increases from left to right. The top-left cell has row = 1 and column = 1.

The Line class must have the constructor

```
Line(int rowBase, int colBase, int length, int rowIncrement,
   int colIncrement, char drawingCharacter)
```

The Rectangle class must have the constructor

```
Rectangle(int rowBase, int colBase, int height, int width,
    char drawingCharacter)
```

The Triangle class must have the constructor

```
Triangle(int rowBase, int colBase, int height, int rowIncrement,
   int colIncrement, char drawingCharacter)
```

The Circle class must have the constructor

```
Circle(int rowBase, int colBase, int radius, char drawingCharacter)
```

The Text class must have the constructor

```
Text(int rowBase, int colBase, String text, int rowIncrement,
   int colIncrement)
```

In addition, every shape class (Line, Rectangle, Triangle, Circle, Text) must also have the method to draw itself on a Window

```
void draw(Window window)
```

Task 1

a) Design and implement

- The class to represent a drawing window
- The abstract class `Shape`
- The class that represents a line

b) Test your classes with the `EightLines` program below. The program must be able to run with your classes without any change.

```
public class EightLines {
   public static void main(String [] args){
      Window window = new Window(20, 20, '*');

      int row = 10, column = 10, length = 5;

      Line line = new Line(row, column, length, 0, 1, '1');
      window.addShape(line);
      line = new Line(row, column, length, 1, 1, '2');
      window.addShape(line);
      line = new Line(row, column, length, 1, 0, '3');
      window.addShape(line);
      line = new Line(row, column, length, 1, -1, '4');
      window.addShape(line);
      line = new Line(row, column, length, 0, -1, '5');
      window.addShape(line);
      line = new Line(row, column, length, -1, -1, '6');
      window.addShape(line);
      line = new Line(row, column, length, -1, 0, '7');
      window.addShape(line);
      line = new Line(row, column, length, -1, 1, '8');
      window.addShape(line);

      window.display();
   }
}
```

The program should produce the output:

```
* * * * * * * * * * * * * * * * * * * *
*                                       *
*                                       *
*                                       *
*                                       *
*         6         7         8         *
*           6       7       8           *
*             6     7     8             *
*               6   7   8               *
*                 6 7 8                 *
*         5 5 5 5 5 8 1 1 1 1 1         *
*                 4 3 2                 *
*               4   3   2               *
*             4     3     2             *
*           4       3       2           *
*         4         3         2         *
*                                       *
*                                       *
*                                       *
*                                       *
*                                       *
* * * * * * * * * * * * * * * * * * * *
```

Task 2

a) Design and implement the rest of the shapes

b) Test your classes with the program `HouseForSale` below. It should produce the drawing on page 2.

```
import java.util.*;
import java.io.*;
public class HouseForSale
{
   public static void main(String [] args) throws Exception
   {
      {
         // Create a window
         Window w = new Window(20, 30, '*');

         // Draw the ground
         Line ground = new Line(19, 1, 29, 0, 1,'#');
         w.addShape(ground);

         // Draw the post
         Line post = new Line(12, 5, 6, 1, 0, '#');
         w.addShape(post);

         // Draw the light
         Circle light = new Circle(10, 5, 2, '+');
         w.addShape(light);

         // Draw the house
         Rectangle house = new Rectangle(8, 16, 11, 10, '=');
         w.addShape(house);

         // Draw the door
         Rectangle door = new Rectangle(11, 19, 8, 4, '=');
         w.addShape(door);

         // Draw the roof
         Triangle roof = new Triangle(2, 21, 6, 1, 0, '*');
         w.addShape(roof);

         // Display text message
         Text msg = new Text(2,10, "FOR SALE", 1, 0);
         w.addShape(msg);

         w.display();
      }
   }
}
```

It is required that the classes you develop must allow the HouseForSale program to run <u>without</u> any changes.

Task 3

(a) In the class Window, implement the following method

```
void writeSpecToFile(String fileName)
```

The method saves the specification of the drawing window (as opposed to the display image of the drawing) in the specified text file.

(b) Write a program, called `T3Main.java`, to generate a drawing (just like the one in the class HouseForSale), display it on the screen and save it to a text file called `T3Drawing.txt`.

The format of the output text file must conform to the specification illustrated by the example below (the example specification is for our House For Sale drawing), *minus* the comments.

```
20 30        // number of rows and columns of the drawing window
*            // character for border
.            // a dot to signal the end of the ''record''
line         // the shape is a line
19 1 29 0 1  // row & column positions of base, length, row increment, column increment
#            // display character
.
line
12 5 6 1 0
#
.
circle
10 5 2       // row position of base, column position of base Base, radius
+            // display character
.
rectangle
8 16 11 10   // row position of base, column position of base Base, height, width
=            // display character
.
rectangle
11 19 8 4
=
.
triangle
2 21 6 1 0    // row & column positions of base, height, row increment, column increment
*
.
text
2 10          // row & column positions of base
FOR SALE      // the text itself
1 0           // row increment, column increment
.
```

Task 4

(a) In the class Window, implement the following static method (class method)

```
Window readSpecFromFile(String fileName)
```

The method reads the text file, constructs and returns the Window object representing the drawing specified in the text file.

The input text file for this method has the same format as described in Task 3.

(b) Write a program, call it `T4Main.java`, to read the description of a drawing from a text file, e.g. `T3Drawing.txt`, and display the drawing on the screen.

Task 5

Add to the class Window two methods, which will be used in Task 6. The first method

```
public void addGrid()
```

adds numbers to the borders to indicate the row and column indexes of the cells, as shown in the example below. The numbers on the edges would help us manipulating the shapes.

```
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 *
1                                         1
2                                         2
3                                         3
4                                         4
5                                         5
6                                         6
7                                         7
8                                         8
9                                         9
0                                         0
1                                         1
2                                         2
3                                         3
4                                         4
5                                         5
6                                         6
7                                         7
8                                         8
9                                         9
0                                         0
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 *
```

The second method has the signature:

```
public void refreshImage()
```

This method first makes all the cells of the drawing image blank (except those on the borders). Then it asks each shape of the window to draw itself on the window.

This method would be called when we want to be sure that the image is up to date.

Task 6

For this task, you implement a program called `DrawingBoard.java`. (You may also need to add some simple methods to the existing classes to support this program.)

- This program first allows the user to create a new drawing window or to load an existing one.
- It then allows the user
  - To add a shape to the drawing,
  - To delete a shape from the drawing,
  - To select a shape and move it (up, down, left, right) or make it bigger or smaller,
  - To save the specification of the current drawing window to a text file.

  For simplicity, we will only be concerned with lines. To be clear, your menu program should be able to read and display drawing that have shapes other than line (as you will use what you'll develop in Task 3 and 4). However, the program provides options to add, delete, move, and change sizes of *lines only.*

The interactions between the user and the program are described below.

1. Create a new drawing window or load an existing one

At the start, the program prompts the user to enter NEW in order to create a new drawing window or to enter a fie name to load an existing drawing window.

Here is a sample run to create a new drawing window (inputs by the user are displayed in bold):

```
> java DrawingBoard
Enter the window file name (or NEW):
NEW
Enter mumner of rows, number of columns and character (separated by space):
10 30 *


* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 *
1                                                             1
2                                                             2
3                                                             3
4                                                             4
5                                                             5
6                                                             6
7                                                             7
8                                                             8
9                                                             9
0                                                             0
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 *
Add Erase Select Write Quit
Up Down Left Right + -
```

And here is a sample run to load an existing drawing window from a text file:

```
> java DrawingBoard
Enter the window file name (or NEW):
SpecIn.txt


* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *           *        6
7          *           *        7
8            *       *          8
9              * * *            9
0                      *        0
1                        *      1
2                          *    2
3                            *  3
4                               4
5                               5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
```

2. Menu options

The last two lines in the display above are the *reminders* of the menu options.

The first line is meant to remind the user of the following options: Add (to add a shape), Erase (to delete a shape), Select (to select a shape, the selected shape can be moved or have its size changed as will be seen shortly), Write (to write the specification of the window to a text file), and Quit.

The second line displays reminders for options that can be applied to the selected shape: Up (to move the shape up), Down (to move the shape down), Left (to move the shape left), Right (to move the shape right), + (to increment the size of the shape) and - (to decrement the size of the shape).

3. Adding shapes

The way this option works is summarized in the table below:

Reminder	Add
Purpose	To add a shape (a line)
To choose the option	Enter **a** and press ENTER
System's response	Display the format to enter a line
User's response	Enter details for a line
System's response	Add the shape to the drawing window, display the window and the menu option reminders (ready for the next option)

Below is a sample run.

```
> java DrawingBoard
Enter the window file name (or NEW):
SpecIn.txt


* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *           *        6
7          *           *        7
8            *       *          8
9              * * *            9
0                      *        0
1                        *      1
2                          *    2
3                            *  3
4                               4
5                               5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
a
line rowBase colBase length rowIncrement colIncrement character
line 6 9 5 1 0 *


* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *       *   *        6
7          *       *   *        7
8            *     * *          8
9              * * *            9
0                  *   *        0
1                  *     *      1
2                          *    2
3                            *  3
4                               4
5                               5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
```

4. Erasing (Deleting) shapes

Reminder	Erase
Purpose	To erase (delete) a shape (a line)
To choose the option	Enter **e** and press ENTER
System's response	Display the indexes and details of the shapes
User's response	Enter the index of the shape to be erased (user can only select a Line)
System's response	Erase the shape from the drawing, display the drawing and the menu option reminders (ready for the next option)

Below is a sample run (which continues from the previous sample run display). In this sample run, we choose to erase the second shape (at index 1).

```
e
0: circle(6,8)(3)(*)
1: line(10,11)(3)(1,1)(*)
2: line(6,9)(5)(1,0)(*)
1


* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *       *   *        6
7          *       *   *        7
8            *     * *          8
9              * * *            9
0                  *            0
1                  *            1
2                               2
3                               3
4                               4
5                               5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
```

5. Selecting shapes

We can select a shape to move it or to change its size.

Reminder	Select
Purpose	To select a shape
To choose the option	Enter **`s`** and press ENTER
System's response	Display the indexes and details of the shapes
User's response	Enter the index of the shape to be selected (user can only select a Line)
System's response	Mark the shape as being selected (you don't need to show any information on screen about which shape was selected), display the drawing and the menu option reminders

Below is a sample run (continues from the previous display). In this sample run, we select the second shape (index 1).

```
s
0: circle(6,8)(3)(*)
1: line(6,9)(5)(1,0)(*)
1


* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *       *   *        6
7          *       *   *        7
8            *     * *          8
9              * * *            9
0                  *            0
1                  *            1
2                               2
3                               3
4                               4
5                               5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
```

6. Moving selected shapes and changing their sizes

To move the previously selected shape up:

Reminder	Up
Purpose	To move the shape up one row (by moving the base point of a line)
To choose the option	Enter **`u`** and press ENTER
System's response	Reduce the row base of the shape by 1, display the drawing and the menu option reminders

To move the selected shape down, left and right: similar.

To increase the size of the selected shape:

Reminder	+
Purpose	To increase the size shape by 1. For a line, its base point remains the same and its length is increased by 1.
To choose the option	Enter **+** and press ENTER
System's response	Increase the size of the shape by 1, display the drawing and the menu option reminders

To decrease the size of the selected shape: similar.

Note: while moving, or increasing the size of a line, if it reaches the border, your application should not allow the move/increment operation.

Below is a sample run, which continues from the previous one. In this sample run, we move the line, which was previously selected, to the left (the entry is actually letter **l**), and then increase it size by 1 (by entering the **+** sign).

```
l

* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *     *     *        6
7          *     *     *        7
8            *   *   *          8
9              * * *            9
0                *              0
1                *              1
2                               2
3                               3
4                               4
5                               5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
+

* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *     *     *        6
7          *     *     *        7
8            *   *   *          8
9              * * *            9
0                *              0
1                *              1
2                *              2
3                               3
4                               4
5                               5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
```

7. Saving the specification of a drawing window to a file

Reminder	Write
Purpose	To save the specification of the window to a text file
To choose the option	Enter **w** and press ENTER
System's response	Prompt for the input file name
User's response	Enter the file name
System's response	Write the window's details to the file (overwrite), display the drawing window and the menu option reminders

Below is a sample run that continues from the previous one. In this sample run, we save the drawing window and then choose **q** to quit.

```
w
File name: SpecOut.txt


* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
1                               1
2                               2
3              * * *            3
4            *       *          4
5          *           *        5
6          *     *     *        6
7          *     *     *        7
8            *   *   *          8
9              * * *            9
0                *               0
1                *               1
2                *               2
3                                3
4                                4
5                                5
* 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 *
Add Erase Select Write Quit
Up Down Left Right + -
q
Thank You!
```

8. Quiting the program

To quit:

Reminder	Quit
Purpose	To quit the program
To choose the option	Enter **q** and press ENTER
System's response	Display "Thank You!" and terminate the program

9. Making the program more robust

Once we start the program and take a few options, we do not want the program to crash. To prevent this, put the actions for each of the menu options in a try/catch block.

Task 7 (for CSE4IOO Students Only)

- Design and implement the class to represent an oval.
- Write program `T7Tester.java` to test your class.
- Submit both `Oval.java` and `T7Tester.java`.

This task is worth 10 marks, taking the total mark for CSE4IOO assignment to 110. (The total mark for CSE1IOO is 100.)

Electronic Submission of the Source Code

- Submit all the Java files that you have developed in the tasks above.
- The code has to run under Unix on the latcs8 machine.
- If you want to copy a file from your local machine (laptop, desktop at home) to the latcs8 server, use WinSCP (for windows) or Cyberduck (for mac).
- You submit your files from your latcs8 account. Make sure you are in the same directory as the files you are submitting.
- Submit each file separately using the submit command. For example, for a file called (say) Window.java:

```
submit IOO Window.java
```

- After submitting the files, you can run the following command that lists the files submitted from your account:

```
verify
```

- You can submit the same filename as many times as you like before the assignment deadline; the previously submitted copy will be replaced by the latest one.

Marking Scheme Overview

The assignment has the total of 100 marks, which are distributed as follows:

- Implementation (Execution of code) 90 marks (Do all parts of the programs execute correctly? Note your programs must compile and run to carry out this implementation marking. So, comment out the non-working code in your submission.)
- Code Design, Layout and Documentation 10 marks (Does the program conform to specifications? Does the program solve the problem in a well-designed manner? Does the program follow good programming practices? Does the indentation and code layout follow a good, consistent standard? Are the identifiers meaningful? Are comments useful in explaining what and how the program works? (Javadoc comments are optional.)
- A detail marking rubric will be published on LMS soon.

■