SENG2200 Programming Languages & Paradigms
School of Electrical Engineering and Computing
Semester 1, 2020

Assignment 1 (100 marks, 10%) - Due March 22, 23:59

1. Objectives

This assignment aims to build the understanding of Java programing in topics of *class*, *inheritance, interface* and *circular doubly-linked data structure*. A successful outcome should be able to demonstrate a solution of the assignment tasks with correct Java implementation, and report.

2. Problem Statement

You are to write a program for a Stock and Station Agent based in Denman, NSW. Your program will be used to compare different land parcels for their value and how appropriate they are for particular types of agriculture as well as soil conservation and possible pasture improvement. Unfortunately, this firm, while willing to trust your programming expertise, finds software development very confronting, and so does not trust things like standard libraries for data structures.

Consequently, all linked list structures to be used are to be designed and coded from scratch and may not use generics or libraries. You need to write your own *non-generic* version of the `Comparable<T>` interface, unless you would like to use the following:

```
public interface ComparePoly {
    boolean ComesBefore(Object o); // true if this < param
}
```

2.1 Written Report

As you design, write and test your solution, you are to keep track of and report (max. 3 pages) on the following:

1. Keep track of how much time you spend designing, coding and correcting errors, and how many errors you need to correct.
2. Keep a log of what proportion of your errors come from design errors and what proportion from coding/implementation errors.
3. Given what we have covered in Topic 3 (Inheritance), how could you treat Rectangles and Squares as special cases of this assignment?

2.2 Point Class

Design and write a ***Point*** class. The ***Point*** class simply has two double-precision floating point values for x and y coordinate values. It has a method that will calculate the distance of the point from the origin. Your ***Point*** class should also contain a **`toString( )`** method which will allow the conversion of a ***Point*** object into a String of the form **`(x , y)`** – include the open and close parentheses and the comma in the String. Note that `toString()` method (throughout this assignment) should return a `string` rather than print it out. The **x**

and **y** values will appear in the string formatted according to the **4.2f** specification. The string returned, will be used for the output of your results.

2.3 Polygon Class

Design and write a ***Polygon*** class which contains a sequence of ***Point*** objects representing the ordered vertices of the polygon. Your ***Polygon*** class should contain a **`toString( )`** method which will allow the conversion of a ***Polygon*** object into a String of the form

$$\mathbf{[point_0 point_1 \ldots point_{n-2}]: area}$$

For the area values, the format to be used is **6.2f**. This will be used for output of your results. The ***Polygon*** class has a method that will calculate the **`area`** of the polygon, given by the formula in Section 3.

You will also need a method to return the **`distance`** from the origin of the $\mathbf{point_i}$ vertex which is closest to the origin. ***Polygon*** will implement a **`ComparePoly`** interface as per the above, and the ***Polygon*** comparison specification given below.

For any two polygons, if their area difference is within 0.1% of the smaller polygon, then they are assumed to have equal area. In these cases of equal areas, the polygon with the lower minimum vertex distance from the origin, takes precedence (comes first in your list – Section 2.4).

2.4 Data Structure

You are required to implement a *circular doubly-linked list* data structure with the class name of ***MyPolygons***, using a *single* **`sentinel`** node to mark the start/finish of what will become a container for ***Polygon*** objects. The ***MyPolygons*** class must provide distinct methods to

- `prepend` items into the start of the list (`current` item is the new first in list),
- `append` items into the end of the list (`current` item is the first in list),
- `insert` before a specified (`current`) item,
- step to the `next` item (making it the *current* item),
- `reset` the `current` item variable to the start of your list, and,
- take (then `remove`) an item from the head of the list.

Note: that you may not have to use all the above methods in this assignment.

2.5 Required Processing

- Read polygon specifications (until end of file, from standard input) and place them into an instance of your container, in input order.
- Parse and store each polygon specification. A polygon will be specified in the input by the letter **P**, followed by the number of sides the polygon has (ie: n-1 in the formula above), and then pairs of numbers which represent the respective vertices on the Cartesian plane (x-value then y-value), which means vertices $\mathbf{point_0}$ to $\mathbf{point_{n-2}}$ from the above formula. You do not have to worry about any of the data being missing, or out of order. (Hint: it will be best for your polygon objects to have an array that will contain all n points, that is, explicitly including the last vertex as a copy of the first, as this will allow the easiest implementation of the area formula, but you may use an alternate means of storing the polygon vertices if you wish.)

- Then, you are to produce a second list, which contains the same set of ***Polygon*** objects, but this time, sorted into *increasing area order*. Implement an *insertion sort* algorithm to sort the polygon objects.
- Output for your assignment is a complete print of both your lists, ie: the polygons in input order, and then the polygons in sorted order, listing the area of each polygon in each case, as per the ***Polygon*** specification of **`toString( )`** given above.

2.5 Input and Output

All input (coming from standard input) of polygons will be from a user-defined text file. That is, your program should be able to read polygons from a text file. A sample test file is as follows (you should create one for self-testing before submission):

```
P 6 4 0 4 8 7 8 7 3 9 0 7 1
P 3 4.1 0.0 4.0 8.2 7.3 8.4
P 5 4.0 0 4 8.1 7.2 8 7 3 9 0
```

All output is to the standard output. A sample output of the above input is as follows:

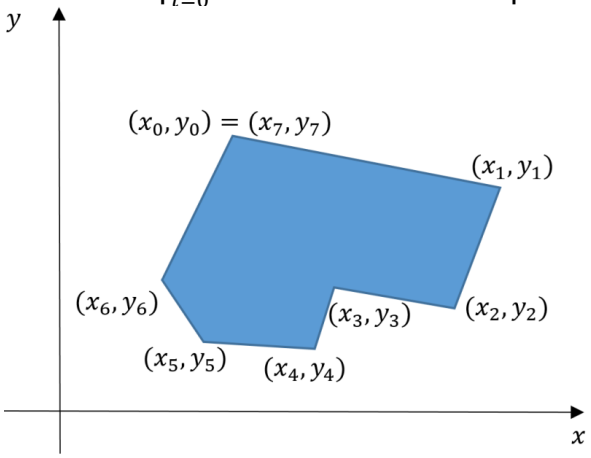
```
Unsorted list
[(4.00 , 0.00)(4.00 , 8.00)(7.00 , 8.00)(7.00 , 3.00)(9.00 , 0.00)(7.00 , 1.00)]:  24.50
[(4.10 , 0.00)(4.00 , 8.20)(7.30 , 8.40)]:  13.54
[(4.00 , 0.00)(4.00 , 8.10)(7.20 , 8.00)(7.00 , 3.00)(9.00 , 0.00)]:  27.66
Sorted list
[(4.10 , 0.00)(4.00 , 8.20)(7.30 , 8.40)]:  13.54
[(4.00 , 0.00)(4.00 , 8.00)(7.00 , 8.00)(7.00 , 3.00)(9.00 , 0.00)(7.00 , 1.00)]:  24.50
[(4.00 , 0.00)(4.00 , 8.10)(7.20 , 8.00)(7.00 , 3.00)(9.00 , 0.00)]:  27.66
```

You may only use Java libraries for input and output.

3. Polygon Area Calculation

The area of a polygon, specified on the Cartesian plane by its vertices, is given as follows:

$$A = \frac{1}{2}\left|\sum_{i=0}^{n-2}(x_{i+1} + x_i)(y_{i+1} - y_i)\right|$$



Note: that for a seven-sided figure, such as the one shown, $n = 8$, because the last point describing the polygon is equal to the first (it is always the same Cartesian point).

You may verify your polygon formula implementation using

- https://rechneronline.de/pi/simple-polygon.php

4. Submission

Submission is through the Assessment tab for SENG2200 on Blackboard under the entry Assignment 1. If you submit more than once then only the latest will be graded. Every submission should be ONE ZIP file (*not a .rar, or .7z, or etc*) named **`c9999999.zip`** (where c**`9999999`** is your student number) containing:

- Assessment item cover sheet.
- Report (PDF file): addresses the tasks in Section 2.1.
- Program source files.

Programming is in Java. Name your startup class PA1 (capital-P capital-A number-1), that is, the marker will compile your program with the command:

```
javac PA1.java
```

…and to run your program with the command:

```
java PA1 test.dat
```

…within a Command Prompt window.

Note that **`test.dat`** *is a placeholder; your program will have to handle different names and extensions for testing.*

If your program cannot be compiled and executed (incl. run-time errors) by using the above commands, you may receive ZERO mark without being examined.

5. Marking Criteria

This guide is to provide an overview of Assignment 1 marking criteria. It briefly shows the mark distribution. This guide is subject to adjustment without notice.

Program Correctness

- ***ComparePoly*** Interface **(10%)**
- ***Point*** class **(10%)**
    - *Incl. required methods*
- ***Polygon*** class **(15%)**
    - *Incl. required methods and insertion sort algorithm.*
- ***MyPolygons*** class
    - *Circular doubly-linked data structure* **(20%)**
    - *Other methods (e.g., prepend and append)* **(15%)**
- ***Input/Output*** **(10%)**

Report **(10%)**

Miscellaneous

- Code Format & Comments **(10%)**

Deductions

- Errors with filenames, submission, execution, mathematic results will attract penalties

** The mark for an assessment item submitted after the designated time on the due date, without an approved extension of time, will be reduced by 10% of the possible maximum mark for that assessment item for each day or part day that the assessment item is late. Note: this applies equally to week and weekend days.*

*** The assignment (code and report) will be checked for plagiarism. A plagiarized assignment will receive a zero mark and will be reported to SACO.*

Nan and Dan
2020-02-01 v1.0