SENG2200 Programming Languages & Paradigms
School of Electrical Engineering and Computing
Semester 1, 2019

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

1. Objectives

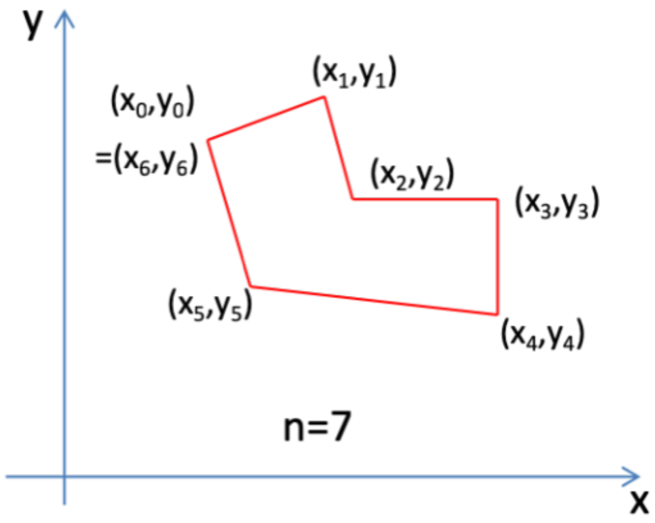
This assignment aims to build the understanding of Java programing in topics of *class*, *interface*, *inheritance* and *circular doubly-linked data structure*. A successful outcome should be able to demonstrate a solution of assignment tasks with correct Java implementation and report.

2. Initial Information

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

$$A = ½ \left| \sum_{i=0}^{n-2} (x_{i+1} + x_i)(y_{i+1} - y_i) \right|$$

e.g.



Note that for a six sided figure, such as the one shown, n is 7, because the last point describing the polygon is equal to the first (it is always the same Cartesian point).

3. 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. Hence, we haven't even bothered to ask them about things like *generics* being used in data structures or interfaces.

Consequently, all linked list structures to be used are to be designed and coded from scratch and may not use generics or libraries. You will even have 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
}
```

Design and write two classes ***Point*** and ***Polygon***. The ***Point*** class simply has two double-precision floating point values for x and y coordinate values. It should have 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.

The ***Polygon*** class 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 **[point$_0$…. point$_{n-2}$]: area**. For the area, the format to be used is 5.2f. 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 2. You will also need a method to return the *distance* from the origin of the **point$_i$=(x$_i$,y$_i$)** vertex which is closest to the origin. ***Polygon*** will implement the *ComparePoly* interface (or your own interface) as per the above, and the ***Polygon*** comparison specification given below.

You are also required to implement a *circular doubly-linked* 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 will contain methods to

- *prepend* items into the list (*current* item is the new first in list),
- *append* items into 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),
- resetting the *current* item to designate the start of your list, and,
- taking (then remove) an item from the head of the list.

Note: 4.2f and 5.2f: They are about the formatting of output. It is in form of <width>.<precision>. 5.2f means it reserves **at least** 5 character places (including ".") for output and there are two digits after the dot. For example, if we have a number 3.456, the output of 5.2f is x3.46, where x is a space.

4. Input, Required Processing and Output

The first main task is to read polygon specifications (until end of file, from standard input) and place them into an instance of your container, in input order.

Each 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 $\mathbf{p_0}$ to $\mathbf{p_{n-2}}$ from the above formula. You do not have to worry about any of the data being missing, or out of order. 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.

You are then to produce a second list, which contains the same set of ***Polygon*** objects, but this time, sorted into *increasing area order*. This is probably done most simply by placing the polygons into the second list using an *insertion* sort algorithm. If any two ***Polygon*** objects have areas within 0.05% of each other, 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 sorted list).

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.

Example input specification for a 6 sided polygon is:

```
P  6  4 0  4 8  7 8  7 3  9 0  7 1
```

That is, the letter P, then the (integer) number of sides (6), then 6 (in this case) pairs of values for the 6 vertices. While this example data only uses integer data, floating point values are possible for the **x** and **y** values.

All input of polygons will be from `test.dat` file. That is, your program should be able to read polygons from a text file. A sample test.dat 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 0 4 8 7 8
P 5 4 0 4 8 7 8 7 3 9 0
```

5. The Written Report

As you design, write and test your solution, you are to keep track of and report on the following:

1. For each of the programs 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 now treat Rectangles, and Squares as special cases in this problem?

6. Submission

Submission is through the Assessment tab for SENG2200 on Blackboard under the entry Submit PA1. If you submit more than once then only the latest will be graded. Every submission should be ONE ZIP file named c9999999PA1.zip (where 9…9 is your student number) containing:

- Assessment item cover sheet.
- Report (PDF file): addresses the tasks of Section 5.
- Program source files.

Programming is in Java. Name your startup class PA1 (capital-P capital-A number-1), that is, the marker will expect to be able 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.

You may only use Java libraries for input and output. All input to the program will come from *standard input*. All output is to *standard output*.

7. Marking Criteria

*** If your submission cannot be compiled or has run-time errors (e.g., segmentation fault), you may receive ZERO mark.*

*** 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.*

This guide is to provide an overview of Assignment 1 marking criteria. It briefly shows marks of each section. This guide is subject to be adjusted.

Program Correctness (80%)

- ***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 (10%)

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

  Note: the issues from either code format or comments may cause the maximum deduction from this component.