SENG2200 Programming Languages & Paradigms
School of Electrical Engineering and Computing
Semester 1, 2021

Assignment 2 (100 marks, 15%) - Due April 30, 23:59

1. Objectives

This assignment aims to build the understanding of Java programming in topics of *interface, polymorphism*, *generics* and *iterator*. A successful submission should be able to demonstrate a solution of the assignment tasks, with correct Java implementation and report.

2. Problem Statement

The firm in Denman still believes that data structures that are hand-coded will run faster than those used from the Java libraries (*still n00bs*), and so this is still required for your container types. However, once the true nature of **generic structures** was explained to them, they have decided that containers will use **generic specifications and standard iterator *interfaces***.

NOTE: You can reuse the code from your Assignment 1 as much as possible.

2.1 Written Report

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

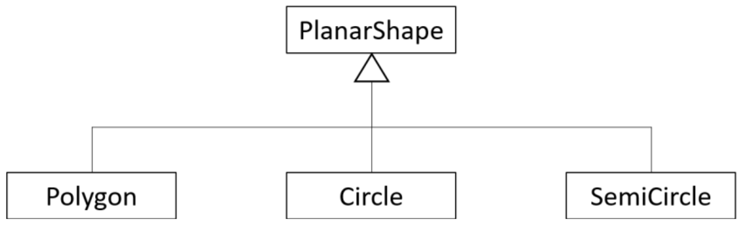
1. For each of the programs keep track of how much time you spend designing, coding and correcting errors, and how many errors you need to correct. Sum these numbers once your implementation is complete.
2. Keep a log of what proportion of your errors come from design errors and what proportion from coding/implementation errors. Address any trends you noticed from your logs and results.
3. Provide a (*brief*) design of how you would further extend your **PA2** so that it specifically included **`Triangle`** and **`Square`** figures. Draw the UML class diagram for this new program (*intricate detail not required*). What attribute data do you need in each case?
4. Investigate the mathematical structure of an **`Ellipse`** on the Cartesian plane. How would you model the **`Ellipse`**? How would you then calculate its area and **`originDistance()`**? How would this be incorporated into your program? Draw another UML class diagram to show this.

2.2 Point Class

The **Point** class should be exactly the same as for Assignment 1. (Please refer to *Section 2.2 of Assignment 1*, if needed.)

2.3 Redesign

In Assignment 1, we didn't take much thought to realise that there are many shapes other than polygons (*even when we remember that rectangles and squares are also polygons*). To provide the program extensibility, you need to redesign in a completely different approach for below.



2.4 The Abstract Class - PlanarShape

Design and write an abstract **`PlanarShape`** class. It will have an abstract **`toString()`** method for printing results, an abstract **`area()`** method and an abstract **`originDistance()`** method. Because we need to compare PlanarShape objects, we also need to use (*implement*) the standard **`Comparable<T>`** interface.

`PlanarShape` has an ordering based on **`area()`** and **`originDistance()`**as previously was the case for **`Polygons`**. If any two **`PlanarShape`** objects have areas *within 0.05% units* of each other, then they are assumed to have *equal area*, in which case, the planar shape with the lower **`originDistance()`** takes precedence.

2.5 Polygon Class

The **`Polygon`** class is similar to Assignment 1 but will require a re-design because of Section 2.3 and 2.4 (*ie: you can reuse your code, but it will be changed somehow*). A special thing about a **`Polygon`** now, is that it is a **`PlanarShape`**.

Input data specification for a polygon will be exactly the same as Assignment 1. The output format (*ie:* **`toString`**() *method*) will be

$$\texttt{POLY=[point}_0\texttt{point}_1\texttt{…point}_{n-2}\texttt{]: area\_value}$$

For the area values, the format to be used is **5.2f**.

2.6 Circle Class

Design and write a **`Circle`** class. Circle object is specified by a **`Point`** which is the centre of the circle and a double type value ***r*** which is the radius of the circle. The area of a circle is the usual πr^2. The **`originDistance()`** of a circle is given as the distance from the origin of the centre *minus* the radius (*possibly negative*). Its **`toString()`** method produces the string:

$$\texttt{CIRC=[point}_0\texttt{ r]: area\_value}$$

Input data specification for a circle object is **C x_0 y_0 r** (E.g., **`C 5.1 4.0 3.2`**). For the radius values and the area values, the format to be used are **3.2f** and **5.2f**, respectively.

2.7 Semi-Circle Class

Design and write a **`SemiCircle`** class. **`SemiCircle`** object is specified by a **`Point`** which is the centre of the base of the semicircle, and a second **`Point`** which specifies the point on the semi-circle at the end of the perpendicular to the base.

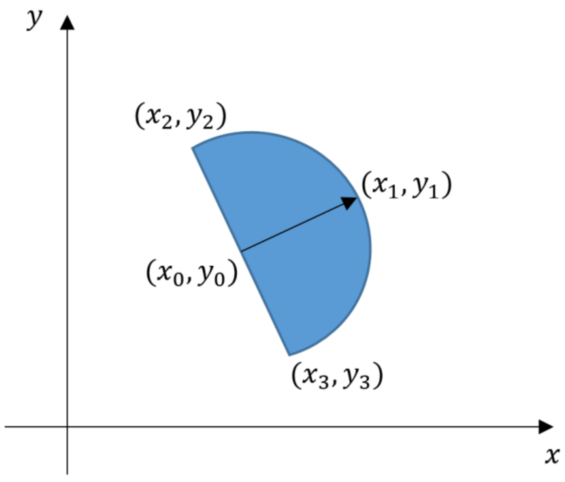
The radius of the semi-circle is the length of the perpendicular vector, and so the area of the semi-circle is simply $\pi r^2/2$.

The **`originDistance()`** of a semi-circle is given as the distance from the origin of the closest of the two data points and the two base extremity points (x_2, y_2) and (x_3, y_3).

It's **`toString()`** method produces the string:

`SEMI=[point₀ point₁]: area_value`

Input data specification for a semi-circle object is **S x_0 y_0 x_1 y_1** (E.g., **`S 4 3 5.1 4.2`**). For the area values, the format to be used is **5.2f**.



$$x_2 = x_0 - |y_0 - y_1|, \qquad y_2 = y_0 + |x_0 - x_1|$$
$$x_3 = x_0 + |y_0 - y_1|, \qquad y_3 = y_0 - |x_0 - x_1|$$

2.8 Data Structure

You are required to implement a *Circular Doubly-Linked List with Sentinel*, using **generics** and **iterators**, which you should call **`LinkedList`**. It should contain methods to both **`prepend()`** and **`append()`** items into the list, and a means of taking items from the **`head`** of the list. It *will not need a current* item as a class member because any functionality requiring this should migrate to the **`iterator`** class for the **`LinkedList`**. The sentinel node reference should be called **`sentinel`**.

The **`LinkedList`** requires **`PlanarShape`** objects (*references to them at least*) to be the basis of the list, and we need to *add type protection* for this by way of a generic specification of the list and instantiation of the list so that it *only allows* **`<PlanarShape>`** objects to be inserted and accessed. Your **`LinkedList`** must be **`Iterable`** and so you need to provide an iterator for your **`LinkedList`** class. *This will only implement the standard iterator methods.*

2.9 Sorting

You are required to implement an **`insertInOrder`**(…) method (in your **`LinkedList`** class) to sort a linked list sorted into *decreasing area order*. It will need to properly use the **`comparable`** interface implementation of the **`compareTo<PlanarShape>`** method from within the **`PlanarShape`** class.

2.10 Required Processing

- Read input data specifications (*until end of file, from standard input*) and place them into an instance of your *circular doubly-linked list*, in input order. You may make use of the standard **Java Scanner library** (*and associated Libraries, such as File*)
- Parse and store each "shape" specification. Design and implement a **factory method pattern** to create different types of objects. You do not have to worry about any of the data being missing, or out of order. You may make use of the basic maths functions like **PI**, **abs()** and **sqrt()** from the standard **Java Math library**.
- Then, you are to produce a second empty list, which you will populate using the same set of ***PlanarShape*** objects, added one at a time using the **`insertInOrder(…)`** method.
- Output for your assignment is a complete print of both your lists, *ie: the shapes in input order*, and then *the shapes in sorted order*, listing the area of each shape in each case, as per the specification of **`toString()`** given above. The output should also be produced using *iterators* to visit the objects in the lists.

2.11 Input and Output

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

```
P 5 4.0 0 4 8.1 7.2 8 7 3 9 0
C 5.1 4.0 3.2
S 4 3 5.0 4.0
```

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

```
Unsorted list
POLY=[(4.00 , 0.00)(4.00 , 8.10)(7.20 , 8.00)(7.00 , 3.00)(9.00 , 0.00)]:  27.66
CIRC=[(5.10 , 4.00))  3.20]:  32.17
SEMI=[(4.00 , 3.00)(5.00 , 4.00)]:  3.14
Sorted list
CIRC=[(5.10 , 4.00))  3.20]:  32.17
POLY=[(4.00 , 0.00)(4.00 , 8.10)(7.20 , 8.00)(7.00 , 3.00)(9.00 , 0.00)]:  27.66
SEMI=[(4.00 , 3.00)(5.00 , 4.00)]:  3.14
```

For input and output and formatting, you may only use Java libraries. If you feel you need extra libraries, you can a) discuss in tutorial, or b) email Dan.

4. Submission

Submission is through the Assessment tab for SENG2200 on Blackboard under the entry Assignment 2. 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. (*No Longer Required*)
- Report (PDF file): addresses the tasks in Section 2.1.
- Program source files (each Class in a separate **.java** file)

Programming is in Java - *Current University Lab Environment is Java 11.0*. Name your startup class PA2 (capital-P capital-A number-2), that is, the marker will compile your program with the command:

```
javac PA2.java
```

…and to run your program with the command:

```
java PA2 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 marks without being examined.

5. Marking Criteria

This guide is to provide an overview of Assignment 2 marking criteria. It briefly shows the mark distribution. **This guide is subject to adjustment without notice.**

Program Correctness

- ***Comparable*** Interface **(5%)**
- ***Polymorphism*** **(20%)**
  - *Incl. required methods*
  - *Incl. factory pattern*
  - *If polymorphism concept has not been correctly implemented, you will significantly lose mark here.*
- ***Generics*** **(10%)**
  - *Your program should properly use "generics" concept in (at least) the required classes*
- ***Iterator*** **(20%)**
  - *Implementation of standard Iterator interface*
  - *Each required method implementation may have different marks.*
- ***Input/Output*** **(10%)**
  - *Your program should be able to correctly read text (shape) specifications.*
  - *Your program output should follow any format specified, e.g., 5.2f.*
- ***General*** **(15%)**
  - *Incl. required general methods, such as area calculation and sorting.*
  - *OO design – your program should follow the **basic** principles of OO design, while you will NOT be marked specifically on the design.*

Report **(10%)**

Miscellaneous **(10%)**

- Code Format & Comments

Deductions

- Errors with filenames, submission, execution, mathematic results, and any other issue not covered above 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, Dan and Terry
2021-03-15 v1.0
2021-04-12 v1.1