ENSF 409 – Principles of Software Development
Winter 2020

Lab Assignment #2:

Due Dates	
In-lab	Submit electronically on D2L **before end of lab period (4:20 PM) on Friday January 31**
Post-Lab	Submit electronically on D2L **before 2:00 PM on Friday February 7**

The objectives of this lab are to:

1) write and test a Java application with multiple classes.
2) understand the concept of class relationships such as association, and aggregation, and composition.
3) implement classes in Java with the association, aggregation, and composition relationships among them.
4) Become familiar with modeling concepts and modeling tools


The following rules apply to this lab and all other lab assignments in future:

1. Before submitting your lab reports, take a moment to make sure that you are handing in all the material that is required. If you forget to hand something in, that is your fault; you can't use `I forgot' as an excuse to hand in parts of the assignment late.
2. **20% marks** will be deducted from the assignments handed in up to **24 hours** after each due date. It means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.


In Lab (12 marks)

In-Lab Exercise - 1: Drawing a Class Diagram (6 Marks)

What to do: Download files `Point.java`, `Line.java`, `Polygon.java`, and `Drawing.java` from D2L. These files belong to a Java program that you will use in this exercise and the following exercise (Exercise 2).

You don't need to write any java code in this exercise, but you have to read the given files carefully, and draw a UML class-diagram for the classes in these files, using tools such as StarUML or Microsoft Visio (if you have already installed them on your computer), or online drawing tools such as: https://www.draw.io or trial version of https://www.gliffy.com/ .If you are using any tool you need to be able to save your work as .jpg file or take screenshot of your design.

Note: in this simple exercise, you don't need to show the attributes and the operations of each class. Just focus on the relationship among the classes.

What to Hand in: Insert your class diagram into the document, save it as PDF file and submit it on D2L before the end of your scheduled lab.

In-Lab Exercise - 2: Running a Java Program with Multiple Files (6 Marks)

What to Do: Run the files `Point.java`, `Line.java`, `Polygon.java`, and `Drawing.java` that you downloaded for Exercise-1, using one of the following methods:

- To run this program from command line, enter the following command:

```
javac Point.java Line.java Polygon.java Drawing.java
```

and, then run your program using the following command:

```
java Drawing
```

- If you are using an IDE such as Eclipse (I advise using an IDE for this lab and all future labs), you must first create a Java-project and then import these files under your project. When all the files are imported, run the project and see the program's output in the console window. (Note: You are also free to choose other IDEs such as NetBeans, IntelliJ, etc.)


The program is supposed to show the information about three polygons (in this example three triangles), but it doesn't, because the `toString()` methods of three of the classes in the given files are incomplete. Your job in this exercise is to complete the missing code in the three `toString()` methods. If you complete the definition of the `toString()` methods properly, the program should display the following output on the console window:

```
The lines in polygon 1 are:
 Line 1: starts at (20, 30), and ends at (50, 100)
 Line 2: starts at (50, 100), and ends at (105, 30)
 Line 3: starts at (105, 30), and ends at (20, 30)
The perimeter of the polygon 1 is 250.18:
The lines in polygon 2 are:
 Line 4: starts at (120, 130), and ends at (150, 200)
 Line 5: starts at (150, 200), and ends at (200, 130)
 Line 6: starts at (200, 130), and ends at (120, 130)
The perimeter of the polygon 2 is 242.18:
The lines in polygon 3 are:
 Line 7: starts at (320, 330), and ends at (250, 400)
 Line 8: starts at (250, 400), and ends at (400, 330)
 Line 9: starts at (400, 330), and ends at (320, 330)
The perimeter of the polygon 3 is 344.52
```

What to hand in: Submit a PDF file that contains the copy of your modified Java files `Point.java`, `Line.java`, `Polygon.java`, and also take a screenshot of your program's output that shows an output identical to the output shown, above.

How to submit: Include all your files for the in-lab section in one folder, zip your folder and upload it in D2L before the deadline.


Post Lab (40 marks)

Post-Lab Exercise – 3 - Identifying classes and drawing class diagram (15 Marks)

In this exercise, you are going to:

Task 1: Determine what classes should be used and identify attributes and methods for each class from the requirement description.

Task 2: Draw the class diagram using UML notations.

Requirement description - A small retail shop that sells tools requires an application to manage inventory of different types of tools it sells. The store owner wants to be able to modify the store's inventory by adding new tools, and deleting tools. The owner also wants to be able to search the inventory for tools by tool name, and by tool id. Currently, the information about tools available in the shop and suppliers is stored in two text files which are given on D2L: `items.txt`, and `suppliers.txt`.

The order and type of data given in these files are:

`items.txt`:

```
(id; description or name of tool; quantity in stock; price;
supplier id number)
```

`Suppliers.txt`:

```
(id; company name; address; sales contact)
```

The owner would also like to check the quantity of each item in stock. If the quantity of each item in stock goes below 40 items, then the program should automatically generate an order line for that item. The order line will have the supplier information and the required quantity for that item (The default quantity ordered by each item = 50 – number of existing items). All items ordered each day should be included in an order which has a randomly generated 5-digit id, and the date that was ordered. The order should be written to a text file called `orders.txt`. A sample order file is as follows:



```
**********************************************************************
ORDER ID:                15181
Date Ordered:            January 18, 2016

Item description:        Nic Nacs
Amount ordered:          250
Supplier:                Widgits Inc.

Item description:        Twinkles Inc.
Amount ordered:          50
Supplier:                Air Drills
**********************************************************************
ORDER ID:                26490
Date Ordered:            January 26, 2016

Item description:        Wog Wits
Amount ordered:          100
Supplier:                Winork Manufacturing Inc.
**********************************************************************
```

What to hand in: Submit a PDF file that contains your class diagram for the above problem.

Post-Lab Exercise- 4: From Model (i.e. Design) to Implementation (i.e. Code) (25 Marks)

What to Do: In this exercise, you are going to implement a Tic-Tac-Toe game (text-based). Here is the class diagram for this project.

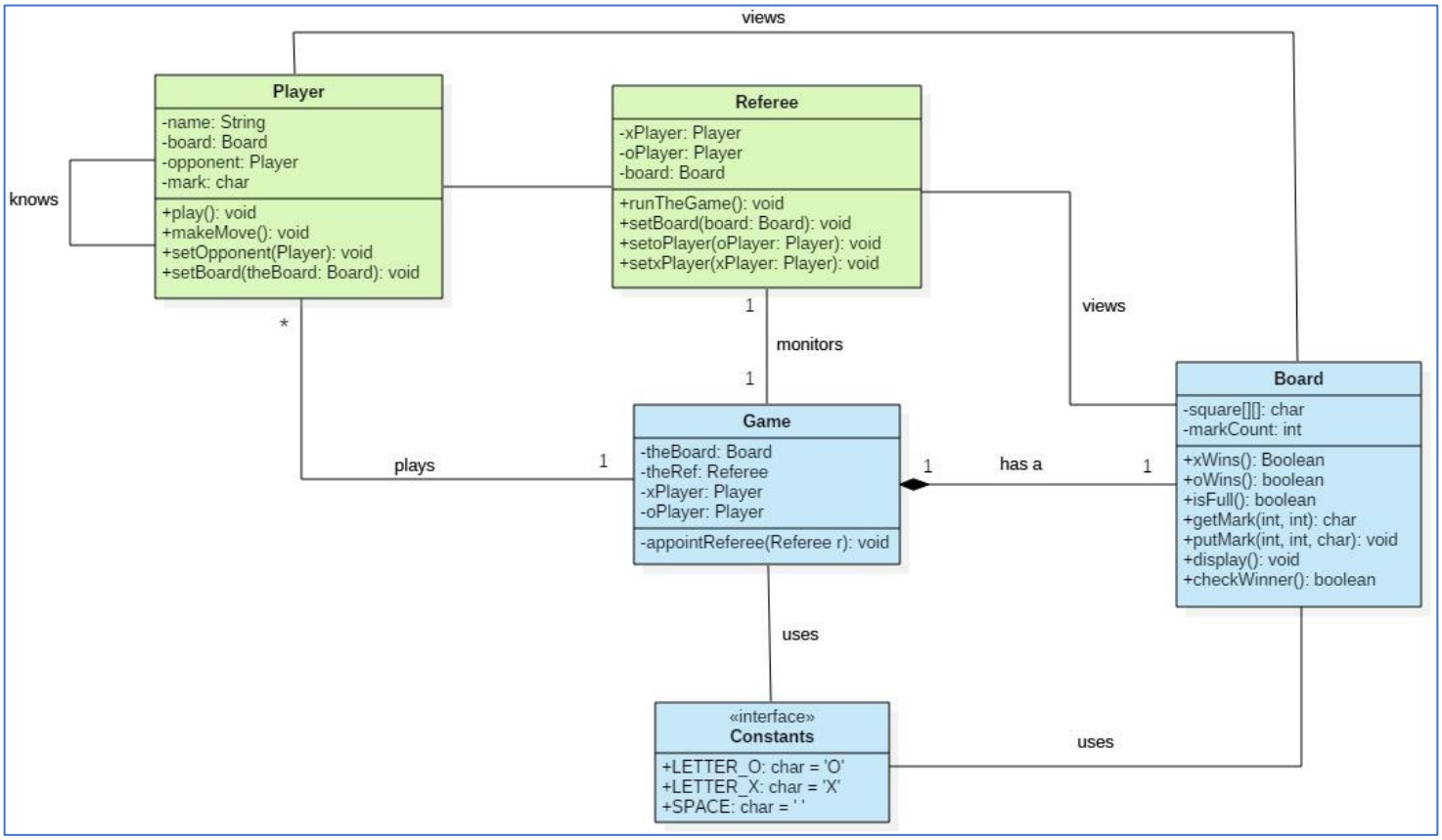

Figure 1 Class diagram for Tic-Tac-Toe

Following classes are given to you. Download them from D2L:

- `Class Board.java` that includes most of the games logic,
- `Class Game.java` that is the starting point of the program,
- `Interface Constants.java` that contains a few constant variables

Therefore, what is missing is the definition of the two other classes: class **`Referee`**, and class **`Player`**.

Task 1: One of your jobs in this exercise is to define and implement these classes.


Task 2: Your other job in this exercise is to add Javadoc comments to ALL classes and interfaces in this program, using the guidelines posted on the D2L. You are strongly advised to start with the documentation of the given classes (`class Board and class Game`). This will help you to understand how these classes work before you start writing code for the other two classes.

In the following section, further details about responsibilities of classes `Referee` and `Player` is given.

`Class Referee`: This class, in addition to its constructor, must have a method called `runTheGame` that calls the `setOpponent` method of class `Player` to set the opponents of the X- and O- players. Then, initiates the game by displaying the board, and calling the play method for the X-Player who is always the first player.

`Class Player`: Each `Player object` must have a name, a mark ('X' or 'O'), and should know its opponent and the board. And, in addition to its constructor and its getter and setter functions as needed, it must have at least the following three methods:

- Method `setOpponent()` that connects the other player to this player.
- Method `play()` that calls method `makeMove()`, as long as methods `xWin()` and `oWin()`, and `isFull()` in class `Board` returns false (If any of these conditions change (turns true), it announces that the game is over and either displays the name of the winner of the game or indicates that the game ended in a 'tie'). Then, displays the board after each move, checks for the winner and then passes the turn to the other player.
- Method `makeMove()` that asks the player to make a move by entering the row and column numbers, and puts a 'X' or 'O' mark on the board, by calling method `addMark` in `class Board`.
- You may add additional methods, if needed.

What to hand in: As part of your post-lab report (PDF format), submit the copy of your java files with the `javadoc` comments, and a sample run of your program output. In addition to the above-mentioned report you need to submit two zipped files:

- A zipped file containing the HTML files generated by `javadoc` tool
- A zipped file containing your actual source codes:
    - `Game.java`



- `Referee.java`
- `Board.java`
- `Player.java`
- `Constant.java`

How to submit: Include all your files for the post-lab section in one folder, zip your folder and upload it in D2L before the deadline.

Appendix A

The Program's Sample Run: To give you a better idea about how the program is supposed to work, a sample run of the program and its dialog with players is given on the next two pages. Note: User inputs are in red.

Please enter the name of the 'X' player: Mike

Please enter the name of the 'O' player: Judy

Referee started the game...

	col 0	col 1	col 2
row 0			
row 1			
row 2			



Mike, what row should your next X be placed in? 1

Mike, what column should your next X be placed in? 1

	col 0	col 1	col 2
row 0			
row 1		X	
row 2			

Judy, what row should your next O be placed in? 2

Judy, what column should your next O be placed in? 0

	col 0	col 1	col 2
row 0			
row 1		X	
row 2	O		

Mike, what row should your next X be placed in? 0

Mike, what column should your next X be placed in? 0

	col 0	col 1	col 2
row 0	X		
row 1		X	
row 2	O		



Judy, what row should your next O be placed in? 2

Judy, what column should your next O be placed in? 2

	col 0	col 1	col 2
row 0	X		
row 1		X	
row 2	O		O

Mike, what row should your next X be placed in? 0

Mike, what column should your next X be placed in? 1

	col 0	col 1	col 2
row 0	X	X	
row 1		X	
row 2	O		O

Judy, what row should your next O be placed in? 2

Judy, what column should your next O be placed in? 1

	col 0	col 1	col 2
row 0	X	X	
row 1		X	
row 2	O	O	O

THE GAME IS OVER: Judy is the winner!
Game Ended ...