ENSF 607 – Principles of Software Development
Fall 2020

Lab Assignment #4: Inheritance and polymorphism

Due Dates	
Post-Lab	Submit electronically on D2L **before 11: 59 PM on Thursday October 22nd**

This is a Group Assignment: In this lab you can work with a partner. Only **groups of two are allowed**. If you are working with a partner, **you must ONLY submit one copy with both names. Please do not submit two copies**. Submitting the same document with two different names is considered plagiarism

The objectives of this lab are to:

1) understand the concept of class relationships, particularly inheritance.
2) get familiarize with more details in a UML diagram, including syntax for attributes and operations.
3) understand the concept of polymorphism
4) implement classes in Java with the inheritance, association, aggregation, and composition relationships among them.


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.

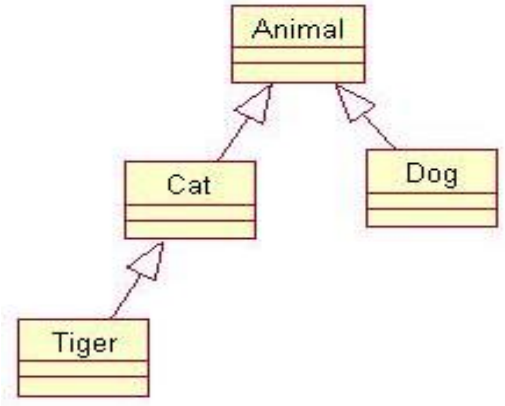

Post Lab (40 marks) + 5 Possible Bonus marks

Exercise - 1: Drawing a Class Diagram (10 Marks)

Read This First – Generalization/specialization: During lectures, we learned that a relationship among classes can be mainly classified as *association*, *composition, aggregation*, and *inheritance*, and the focus of our previous lab was on the first three relationships. This lab focuses on *inheritance*.

What is Inheritance?

Inheritance is an important type of relationship between classes which is also referred to as a generalization/specialization relationship, or a "kind-of" relationship. An example for this type of relationship is the relationship between class Animal and class Cat. Where, Cat is a "kind of" animal. This type of relationship allows a class to be derived from an existing class. In other words, the derived class inherits all the properties (data fields), and behaviors (methods of its base class). UML uses the following notation (a line with a big triangular arrowhead, pointing from sub-class (child) to the super-class (parent), to demonstrate this type of relationship.



What to Do: First, download 8 java files (`Colour.java`, `Point2.java`, `Text.java`, `Shape.java`, `Rectangle.java`, `Circle.java`, `Prism.java`, and `Geometry.java`) from D2L. Then, read the content of these files and try to reverse-engineer a class diagram that shows the relationship among these classes. You are expected to draw a


detailed class diagram that not only shows the relationships among the classes (including multiplicities, etc.), but also the following two sets of information is required:

1. Attributes of each class.
2. Major operations in each class (constructor, getter, setter, and toString methods are not required)

Note: abstract methods are shown in italic in UML.

What to hand in: Submit a PDF file that contains a detailed class diagram.


Exercise - 2: Shapes (10 Marks)

What to Do: First compile the java file you downloaded for exercise 1, and run the program. Your program should produce the following output.

```
Shape name: R1
Origin: X_coordinate: 3.0
Y-coordinate: 4.0
Black point
Width: 6.0
Length: 5.0

Shape name: C1
Origin: X_coordinate: 13.0
Y-coordinate: 14.0
Green point
Radius: 15.0

Shape name: R2
Origin: X_coordinate: 23.0
Y-coordinate: 24.0
Black point
Width: 26.0
Length: 25.0

Shape name: C2
Origin: X_coordinate: 33.0
Y-coordinate: 34.0
Yellow point
Radius: 35.0

Shape name: P1
Origin: X_coordinate: 43.0
Y-coordinate: 44.0
White point
Width: 46.0
Length: 45.0
height: 47.0

Shape name: P2
Origin: X_coordinate: 53.0
Y-coordinate: 54.0
Gray point
Width: 56.0
Length: 55.0
height: 57.0
```

Then, uncomment the last few lines of code in the `Geometry.java` file, under the line, which is labeled as “SECTION 2”. If you try to compile the program now it gives you a few compilation errors because methods: `add`, `showAll`, and `calculator` are missing. Your task in this exercise is to write the definition of the missing methods as follows:



- `Method add`: that adds objects of classes such as `Rectangle, Circle` and `Prisms` to a `TreeSet list`, declared in class `Geometry`.
- Method `showAll`: that displays on the screen the information about those objects that are stored in the `TreeSet` list.
- Method `calculator`: that calculates and displays the area, perimeter, and volume of each object on the screen.

Hint: as we discussed in lectures, to make `TreeSet` list working, you also need to implement `Comparable interface`.

If you add the definition of the above methods your program is expected to display the following outputs in addition to the previous outputs:

```
Adding Rectangle, Circle, and Prism objects to the list...

Showing information about objects added to the list:

Shape name: C1
Origin: X_coordinate: 13.0
Y-coordinate: 14.0
Green point
Radius: 15.0

Shape name: C2
Origin: X_coordinate: 33.0
Y-coordinate: 34.0
Yellow point
Radius: 35.0

Shape name: P1
Origin: X_coordinate: 43.0
Y-coordinate: 44.0
White point
Width: 46.0
Length: 45.0
height: 47.0

Shape name: P2
Origin: X_coordinate: 53.0
Y-coordinate: 54.0
Gray point
Width: 56.0
Length: 55.0
height: 57.0

Shape name: R1
Origin: X_coordinate: 3.0
Y-coordinate: 4.0
Black point
Width: 6.0
Length: 5.0

Shape name: R2
Origin: X_coordinate: 23.0
```



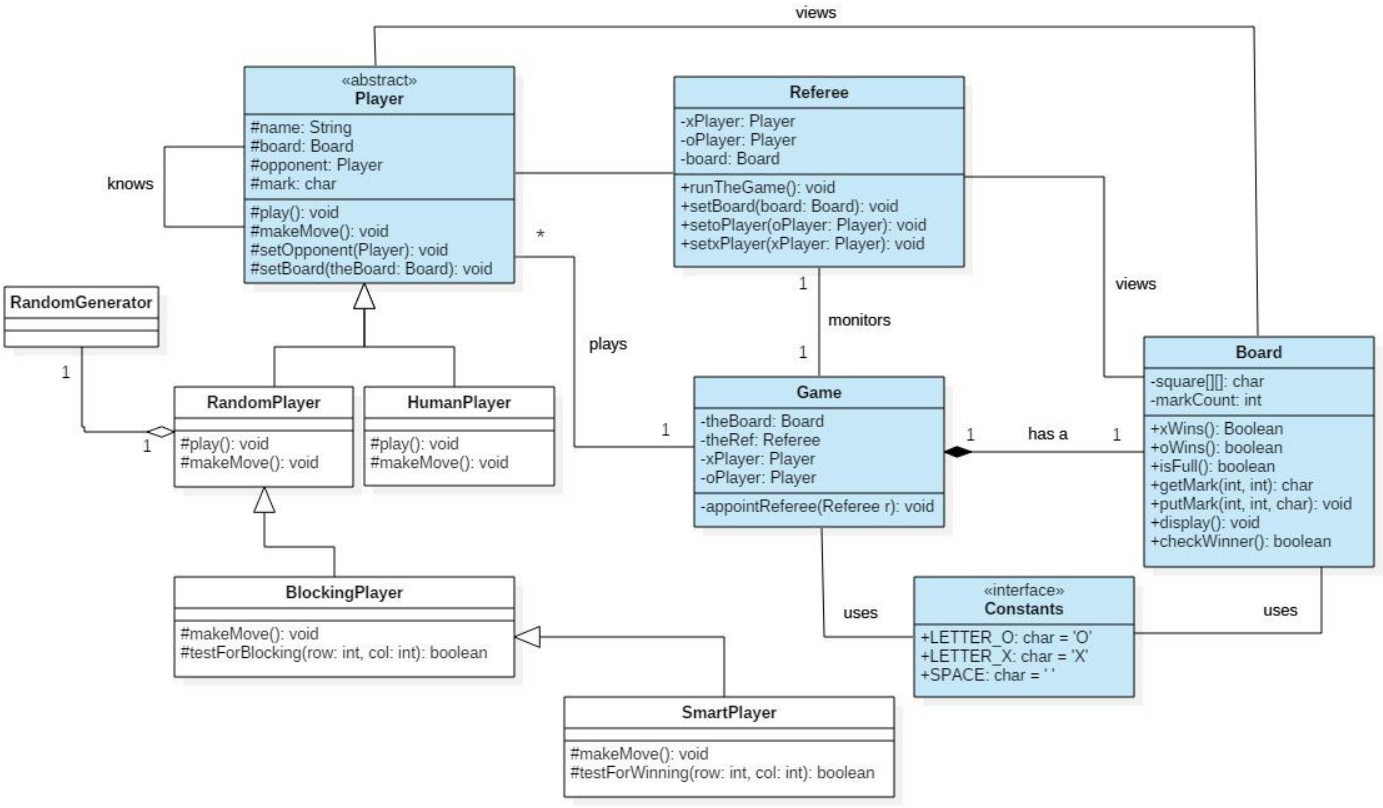
```
Y-coordinate: 24.0
Black point
Width: 26.0
Length: 25.0

Showing area, perimeter, and volume of objects in the list:
The area, perimeter, and volume of C1 are: 706.86, 94.25, 0.00.
The area, perimeter, and volume of C2 are: 3848.45, 219.91, 0.00.
The area, perimeter, and volume of P1 are: 12694.00, 182.00, 97290.00.
The area, perimeter, and volume of P2 are: 18814.00, 222.00, 175560.00.
The area, perimeter, and volume of R1 are: 30.00, 22.00, 0.00.
The area, perimeter, and volume of R2 are: 650.00, 102.00, 0.00.
```

What to hand in: Submit all the `.java` files in your project.



Exercise 3: Tic-Tac-Toe Game with the Inheritance (20 marks + 5 Bonus marks)

In this exercise, you are supposed to add a few classes to your Tic-Tac-Toe Game that developed for Lab 2. To help you understand how to implement these classes the following class diagram, a sample run of the program, and a brief note about the responsibilities of each class are provided. Also, to reduce your workload the definition of class `RandomGenerator` and a modified version of class `Game` is available to download from D2L. You can make any changes to the code in these two files, or use a different way to solve this problem.



A Brief Description on Class `HumanPlayer`: `HumanPlayer` is a sub-class of class `Player`. It is a concrete class and it has to define the abstract methods inherited from class `Player`. As the following sample run shows in this new version of the game the players should have the option of selecting the type of the players. In this example, Mike and Judy, the two players selected the first option (i.e. human player). Which in fact is identical to what the game was doing in lab 2.



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

What type of player is Mike?
  1: human
  2: Random Player
  3: Blocking Player
  4: Smart Player
Please enter a number in the range 1-4: 1

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

What type of player is Judy?
  1: human
  2: Random Player
  3: Blocking Player
  4: Smart Player
Please enter a number in the range 1-4: 1
...
...
<<< game is continued similar to lab 2>>>
```

A Brief Description on Class `RandomPlayer`: `RandomPlayer` is also a kind of `Player`. In fact, this is a computer-player that uses a random generator and picks a vacant spot on the board, randomly. The following example shows part of the game between a human (Mike) and his computer. As you can see in the sample run, Mike's Computer plays randomly and places an O-mark in a randomly selected empty place immediately after Mike moves, and without any interaction for entering the row or the column number.

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

What type of player is Mike?
  1: human
  2: Random Player
  3: Blocking Player
  4: Smart Player
Please enter a number in the range 1-4: 1

Please enter the name of the 'O' player: Mike's Computer

What type of player is Mike's Computer?
  1: human
  2: Random Player
  3: Blocking Player
  4: Smart Player
Please enter a number in the range 1-4: 2

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  |      |      |      |
           |      |      |      |
           +------+------+------+

           |col 0|col 1|col 2
           +------+------+------+
           |      |      |      |
    row 0  |      |      |      |
           |      |      |      |
           +------+------+------+
           |      |      |      |
    row 1  |      |  X   |      |
           |      |      |      |
           +------+------+------+
           |      |      |      |
    row 2  |      |      |  O   |
           |      |      |      |
           +------+------+------+
Mike, what row should your next X be placed in?    1
Mike, what column should your next X be placed in? 0

...
     <<< game is continued >>>
```

Hint: You need to override the definition of method `makeMove`. Feel free to come up with your own algorithm, but here is a suggestion:

> Call the method `discrete` from class `RandomGenerator` twice to return random values between 0 and 2 (if you have 3*3 board) for `i` and `j`, if the `board[i][j]` is available (i.e. empty), then mark it. If the cell is not available, repeat the procedure, until you find an empty spot.

Once your method `makeMove` is completely defined, test it by compiling and running your program a few times. Do not move to the next step until definition of the methods in this class is complete and error free.



A Brief Description on Class BlockingPlayer: A `BlockingPlayer` is kind of `RandomPlayer` that first looks at the board for a move that would block its opponent from winning on the next move. If it can't find any such move, it picks a vacant spot at random. Therefore, before making any move, this class needs to call a method called `testForBlocking`. This method should return true if there is a situation that needs to be blocked. In other words, the process is to traverse through the board, and call `testForBlocking` method for each spot (i^{th} row and j^{th} column). If the function returns true for any of the i^{th} row and j^{th} column, put a mark in that spot, otherwise select an empty random spot (same as `RandomPlayer`).

Optional (Bonus) - Implementation of class `SmartPlayer` is optional.

A Brief Description on Class `SmartPlayer`: A `SmartPlayer` is a kind of `BlockingPlayer`, but slightly smarter and takes the following steps to move:

1. First looks at board, if it can find a move to win immediately, it makes that move.
2. Otherwise, it looks for a way to block its opponent's from winning on the next move.
3. Otherwise, it picks a vacant square at random.

What to hand in: Submit all your source file (.java files). You do not need to submit any javadoc HTML files for this exercise.

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