Object-Oriented Programming

Lab session #1

Before we start, please note:

- Remember, Object-Oriented Programming (OOP) is a programming style where **your class is modeled after real world objects.** In this lab, try to solve questions by creating the relevant objects (**class**), what kind of data (**attribute**) it stores and the tasks (**method**) that it can do.
- As you begin to explore class design, you will sometimes make mistakes and organize your code into the wrong set of classes. **It's OK to get the design wrong**. No matter how much experience you have with class design, and no matter how many rules and "best practices" you've learned, you will still sometimes get it wrong.
- A good programmer will be able to recognize when they've accidentally structured their code into the wrong classes and will refactor and rearrange it so that it is better. **As you get more practice, this process will become easier**, but you will still make mistakes here and there. **Feel free to be wrong and try again!**
- **You should even try to solve lab questions without looking at my sample UML diagram so you can be creative and different in your approach.** The UML diagram is optional but you **should try to compare your UML to my sample UML diagram**. In fact, you **can generate an UML using a plugin** in your chosen IDE (the guide is at the end of this lab file).

To learn more about UML class diagram rules:

- https://en.wikipedia.org/wiki/Class_diagram

To create an UML class diagram (optional for this lab), you can use Lucidchart website:

- https://www.lucidchart.com/pages/uml-class-diagram

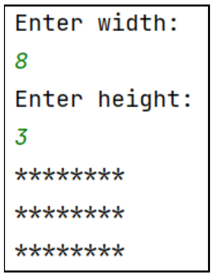
Question 1: Rectangle Visualization (25 points)

Write a **class Rectangle** which has

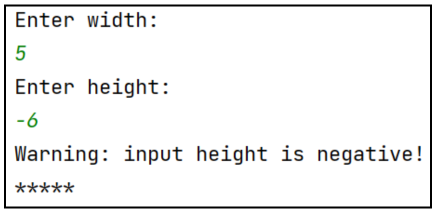
- Two attributes *width* and *height* with appropriate getter methods
- A constructor that requires values for width and height of the rectangle. If either of the inputs is negative, print an error message and set the corresponding attribute to 1.
- A *visualize* method to display the rectangle using * symbol

Write a **class TestRectangle** with a *main* method to instantiate 5 different rectangles and visualize them.

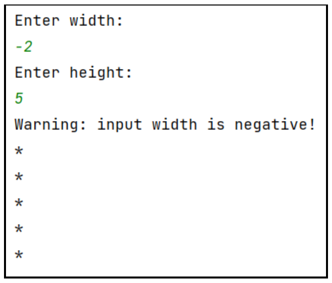
Sample Outputs (yours can be different):



```
Enter width:
8
Enter height:
3
********
********
********
```

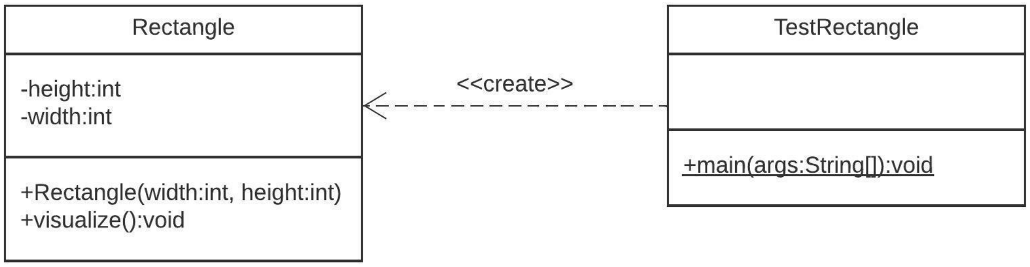


```
Enter width:
5
Enter height:
-6
Warning: input height is negative!
*****
```



```
Enter width:
-2
Enter height:
5
Warning: input width is negative!
*
*
*
*
*
```

Here is an optional sample UML example (yours can be different):

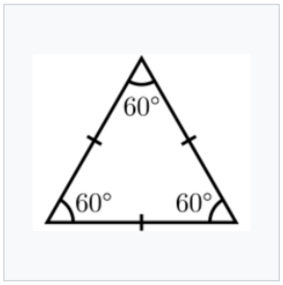


Question 2: Triangle Verification (25 points)

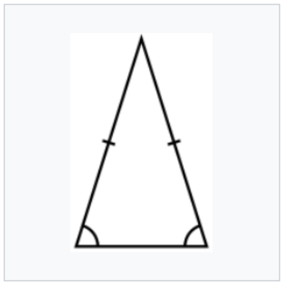
Write a **Triangle class** which has 3 attributes, the length of 3 sides. The class has appropriate constructor and get methods. This class has a method **verify()** to check and return the String type of the Triangle. The types have 4 possibilities: **Not Triangle**, **Equilateral**, **Isosceles** or **Scalene**. Create another class which has a main() method to ask for input lengths of 3 sides, verify it and display the result.

Hint:

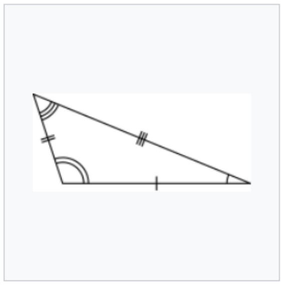
- To check if lengths of three sides to make a triangle: https://www.wikihow.com/Determine-if-Three-Side-Lengths-Are-a-Triangle
- **Equilateral Triangle**: has three sides of the same length (https://en.wikipedia.org/wiki/Equilateral_triangle)
- **Isosceles Triangle**: has two sides of equal length (https://en.wikipedia.org/wiki/Isosceles_triangle)
- **Scalene Triangle**: has all its sides of different lengths.



Equilateral Triangle



Isosceles triangle



Scalene triangle

Sample Outputs (yours can be different):

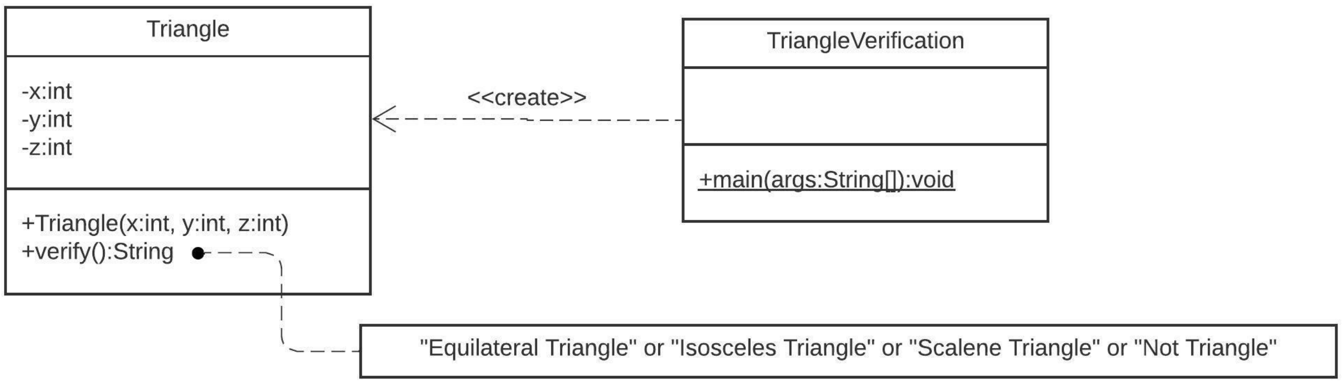
```
Please enter 3 numbers for the sides of a triangle:
3 6 100
Not Triangle
```

```
Please enter 3 numbers for the sides of a triangle:
6 6 9
Isosceles Triangle
```

```
Please enter 3 numbers for the sides of a triangle:
25 25 25
Equilateral Triangle
```

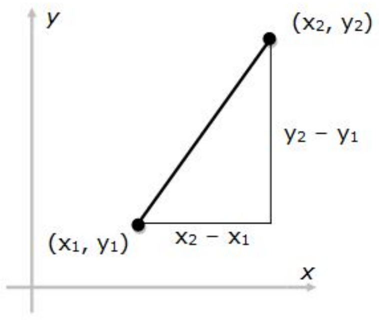
```
Please enter 3 numbers for the sides of a triangle:
3 4 5
Scalene Triangle
```

Here is an optional sample UML example (yours can be different):


Triangle
-x:int
-y:int
-z:int
+Triangle(x:int, y:int, z:int)
+verify():String
<<create>>
TriangleVerification
+main(args:String[]):void
"Equilateral Triangle" or "Isosceles Triangle" or "Scalene Triangle" or "Not Triangle"

Question 3: Distance (25 points)

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



Write a **Point class** that has private attributes for coordinates **x** and **y**. The class has constructor to get values for x and y of the point. In the class, write a method **distance** with the method header to be:

public double distance(Point target)

to compute the distance from the current point and the given target point.
Note: the distance *d* between two points A and B can be computed with the following formula

$$d = \sqrt{\left(x_A - x_B\right)^2 + \left(y_A - y_B\right)^2}$$

Write a class with a main method to test the class Point and the distance method.

Hint:

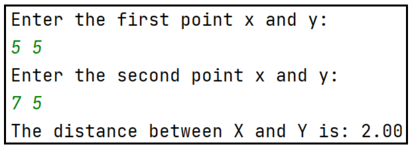
so that the way we calculate distance between two points can be as easy as below:

Point x = new Point(..., ...);

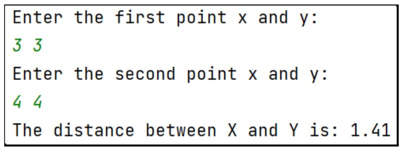
Point y = new Point(..., ...);

System.out.print(x.**distance**(y))

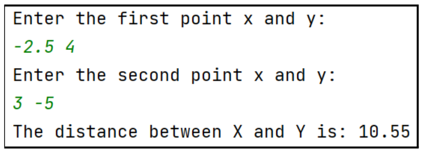
Sample Outputs (yours can be different):



```
Enter the first point x and y:
5 5
Enter the second point x and y:
7 5
The distance between X and Y is: 2.00
```

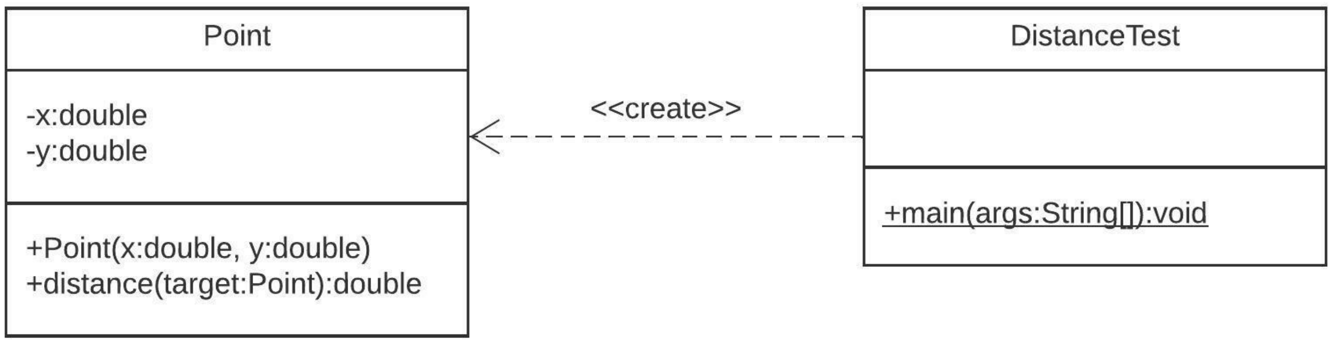


```
Enter the first point x and y:
3 3
Enter the second point x and y:
4 4
The distance between X and Y is: 1.41
```



```
Enter the first point x and y:
-2.5 4
Enter the second point x and y:
3 -5
The distance between X and Y is: 10.55
```

Here is an optional sample UML example (yours can be different):


Point
-x:double
-y:double
+Point(x:double, y:double)
+distance(target:Point):double
<<create>>
DistanceTest
+main(args:String[]):void

Question 4: E-commerce Order (25 points)

You are required to program **Order class** and **Item class** satisfied the requirements as follows:

1. Each order has an ID

2. Each order has a list of Items

3. Each item has an ID, a name and a price

4. Each class has appropriate constructors, get and set methods

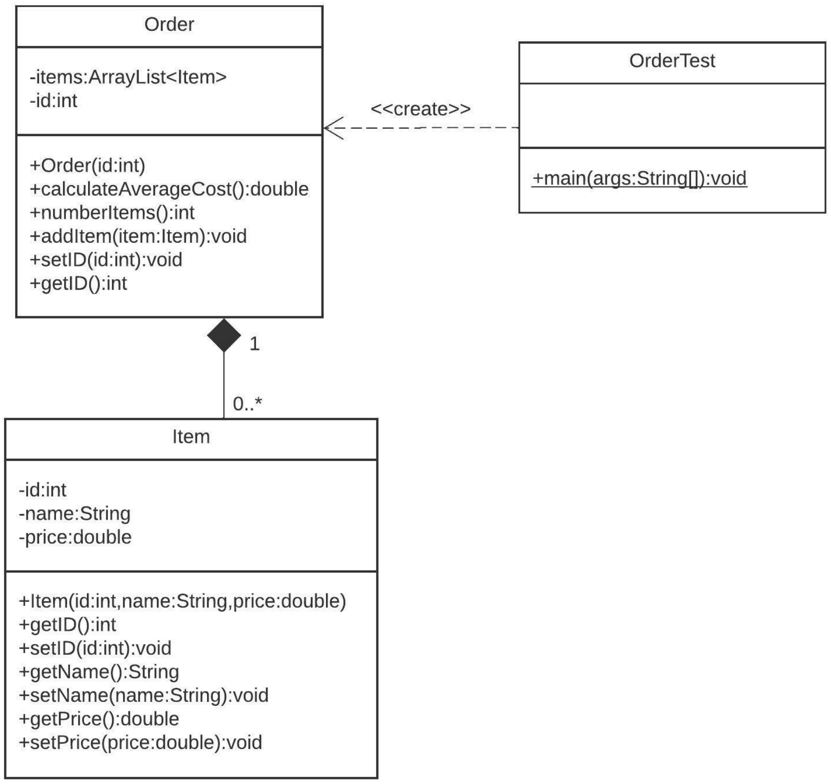
5. Class Order has a method double calculateAverageCost() to calculate the average of the cost of all items in an order

6. Create a class which has a main() method to receive the inputs of items of an order from keyboard and display the averageCost

Sample Outputs (yours can be different):

```
Enter a new number ID for order: 45625
How many items in the order: 3
Please enter the ID for item 1: 123
Please enter the name for item 1: chicken
Please enter the price for item 1: 2.5
Please enter the ID for item 2: 456
Please enter the name for item 2: icecream
Please enter the price for item 2: 7.4
Please enter the ID for item 3: 789
Please enter the name for item 3: ipad
Please enter the price for item 3: 263
You have a new order with ID: 45625
In the order, you have 3 items.
The average price in the order is: 90.97
```

Here is an optional sample UML example (yours can be different):



Question 5 (Bonus): big and small Dice Game (tai sai/dai siu/high low) (Tài Xỉu) (25 points)

During Tet holiday, big and small is a very popular dice game to play. It’s a game of pure chance that moves quickly. There’s no strategy or skill involved. Also, this kind of game is played all over the world.

You will try to implement the simple version of the dai siu game.

A normal dice will have 6 numbers from 1 to 6.

So there are two parties: the house (nhà cái - Tom) and the player (you).

At the beginning of any round, you will put what you want to bet.

The house will roll 3 dice at the same time. The result will be hidden from you until you pick your choice.

The player will pick either Big (Tài) or Small (Xỉu).

Scenario 1: The sum of 3 dice being from 4 to 10: You will win the amount you have bet if you choose Big. Otherwise, you will lose, the house will take your bet.

Scenario 2:The sum of 3 dice being from 11 to 17: You will win the amount you have bet if you choose Small. Otherwise, you will lose, the house will take your bet.

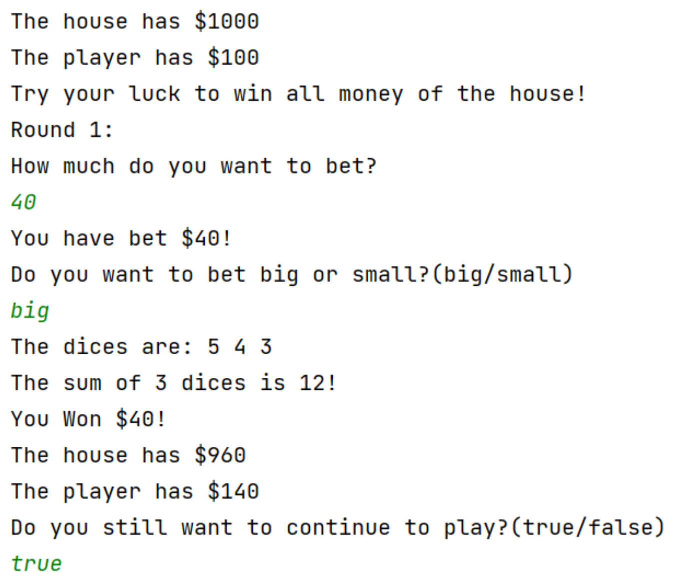
Scenario 3: If all numbers of each dice are the same then you will lose regardless of what you pick Big or Small.

Two parties can start with any money (usually you have much less).

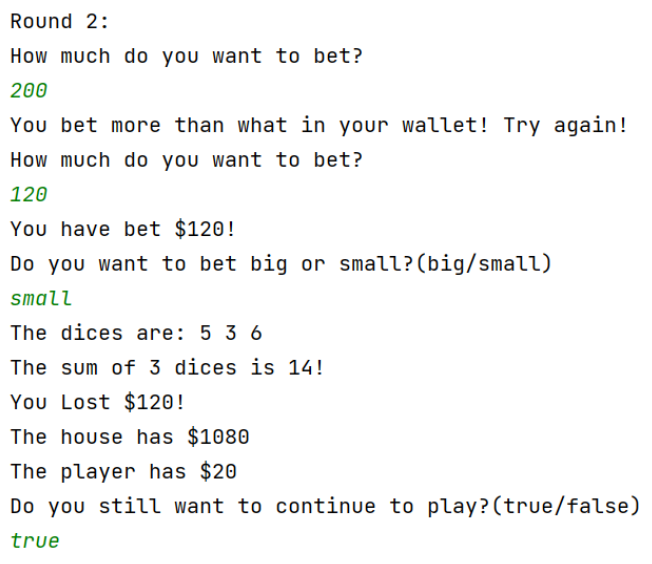
The player can keep playing until either the house or the player runs out of money.

Please try your luck to win all the money for the house.

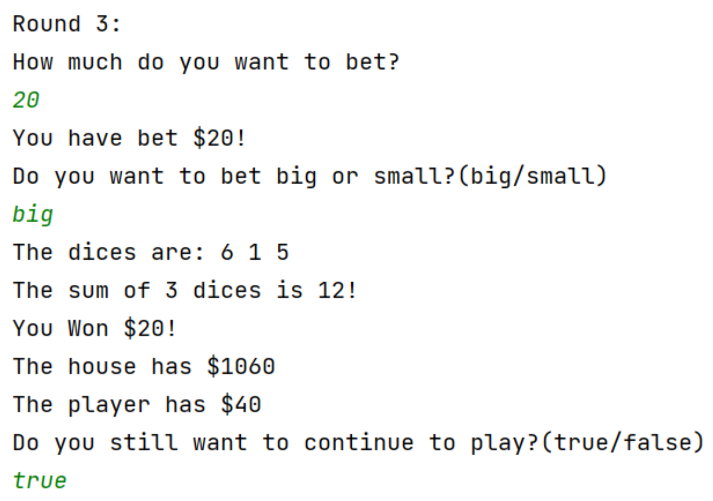
Sample Outputs (yours can be different):



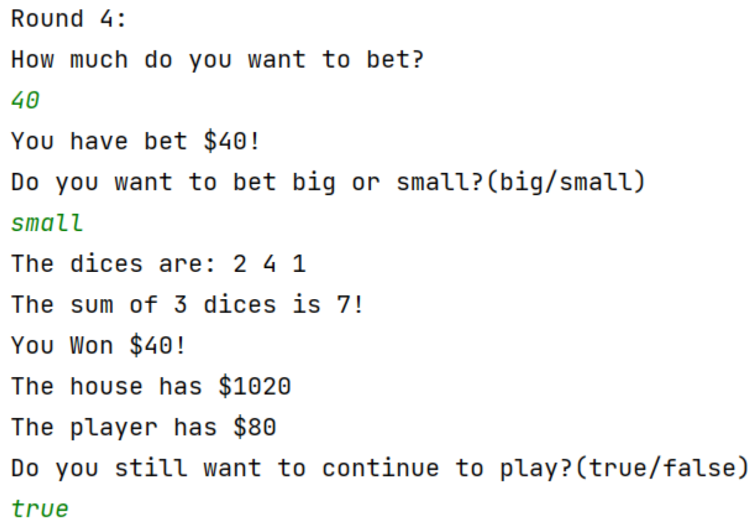
```
The house has $1000
The player has $100
Try your luck to win all money of the house!
Round 1:
How much do you want to bet?
40
You have bet $40!
Do you want to bet big or small?(big/small)
big
The dices are: 5 4 3
The sum of 3 dices is 12!
You Won $40!
The house has $960
The player has $140
Do you still want to continue to play?(true/false)
true
```



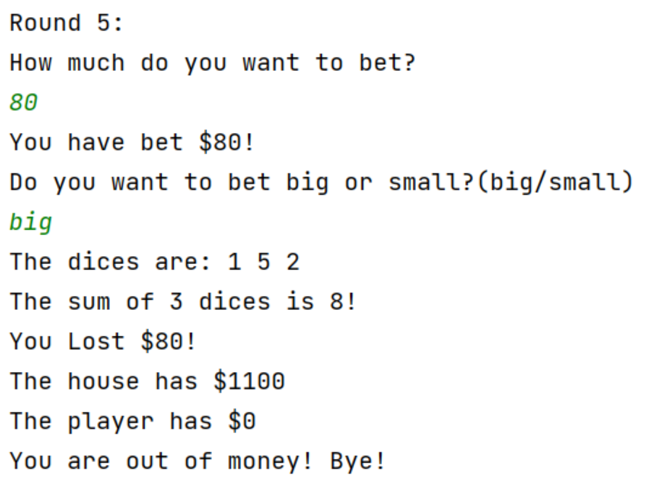
```
Round 2:
How much do you want to bet?
200
You bet more than what in your wallet! Try again!
How much do you want to bet?
120
You have bet $120!
Do you want to bet big or small?(big/small)
small
The dices are: 5 3 6
The sum of 3 dices is 14!
You Lost $120!
The house has $1080
The player has $20
Do you still want to continue to play?(true/false)
true
```



```
Round 3:
How much do you want to bet?
20
You have bet $20!
Do you want to bet big or small?(big/small)
big
The dices are: 6 1 5
The sum of 3 dices is 12!
You Won $20!
The house has $1060
The player has $40
Do you still want to continue to play?(true/false)
true
```

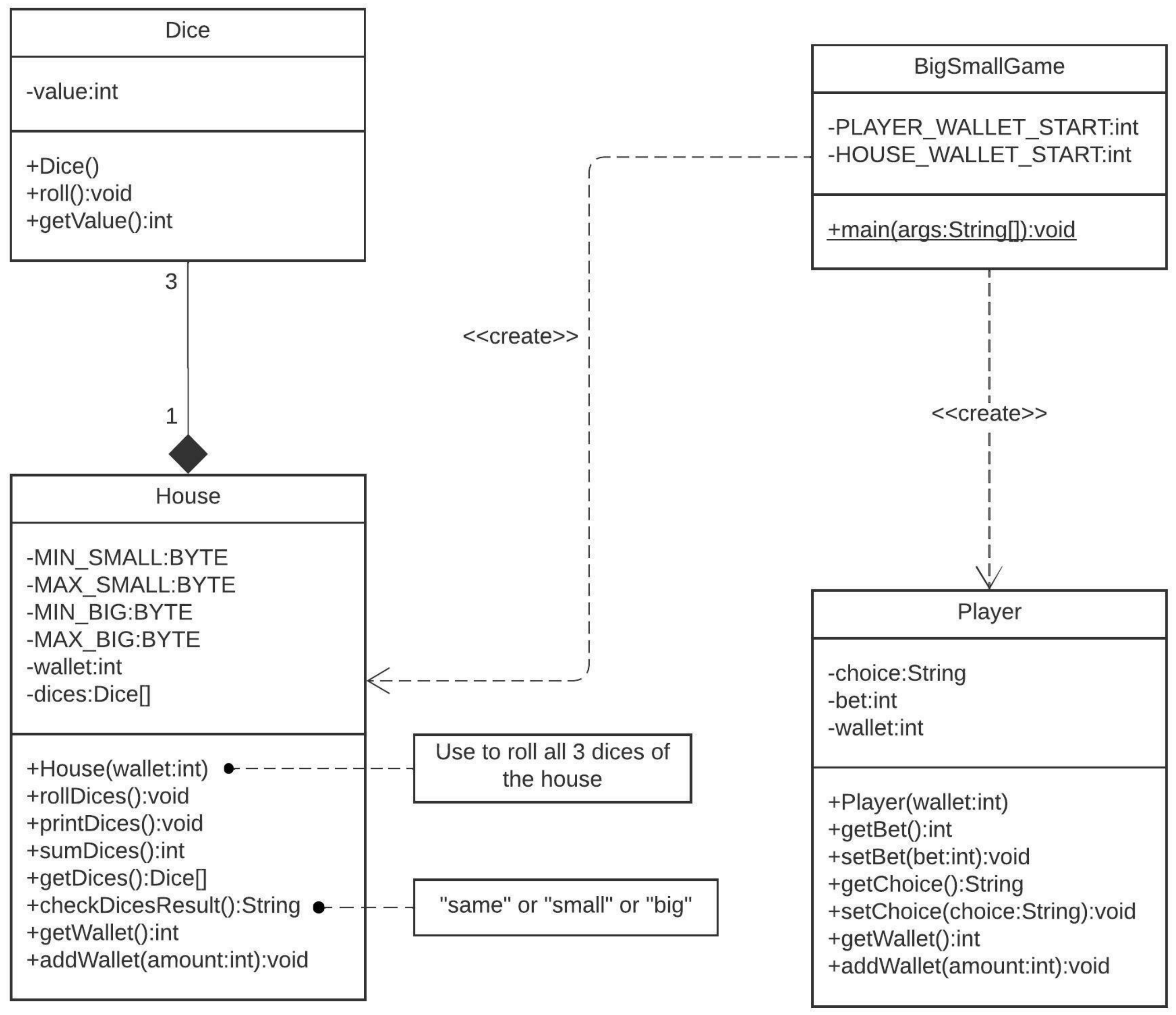


```
Round 4:
How much do you want to bet?
40
You have bet $40!
Do you want to bet big or small?(big/small)
small
The dices are: 2 4 1
The sum of 3 dices is 7!
You Won $40!
The house has $1020
The player has $80
Do you still want to continue to play?(true/false)
true
```



```
Round 5:
How much do you want to bet?
80
You have bet $80!
Do you want to bet big or small?(big/small)
big
The dices are: 1 5 2
The sum of 3 dices is 8!
You Lost $80!
The house has $1100
The player has $0
You are out of money! Bye!
```

Here is an optional sample UML example (yours can be different):


Dice
-value:int
+Dice()
+roll():void
+getValue():int
3
1
House
-MIN_SMALL:BYTE
-MAX_SMALL:BYTE
-MIN_BIG:BYTE
-MAX_BIG:BYTE
-wallet:int
-dices:Dice[]
+House(wallet:int)
+rollDices():void
+printDices():void
+sumDices():int
+getDices():Dice[]
+checkDicesResult():String
+getWallet():int
+addWallet(amount:int):void
Use to roll all 3 dices of the house
"same" or "small" or "big"
<<create>>
BigSmallGame
-PLAYER_WALLET_START:int
-HOUSE_WALLET_START:int
+main(args:String[]):void
<<create>>
Player
-choice:String
-bet:int
-wallet:int
+Player(wallet:int)
+getBet():int
+setBet(bet:int):void
+getChoice():String
+setChoice(choice:String):void
+getWallet():int
+addWallet(amount:int):void