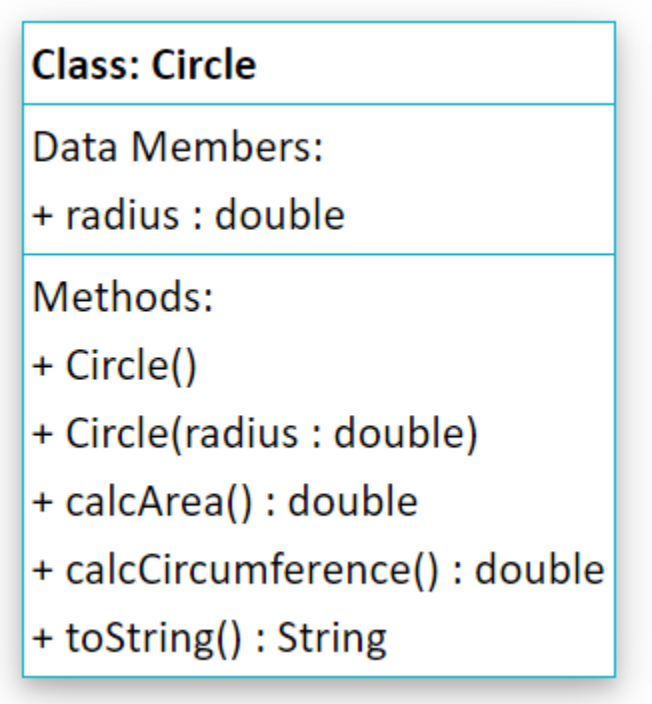
Exercises

Create each of the following classes based on the class diagram below (in UML, something that starts with a + is to be declared as public; if it starts with a - (minus), it is to be declared as private. We haven't talked about the private modifier yet, so for now we'll declare everything as public. Note that this will change for certain members as we learn more about OOP):

1.

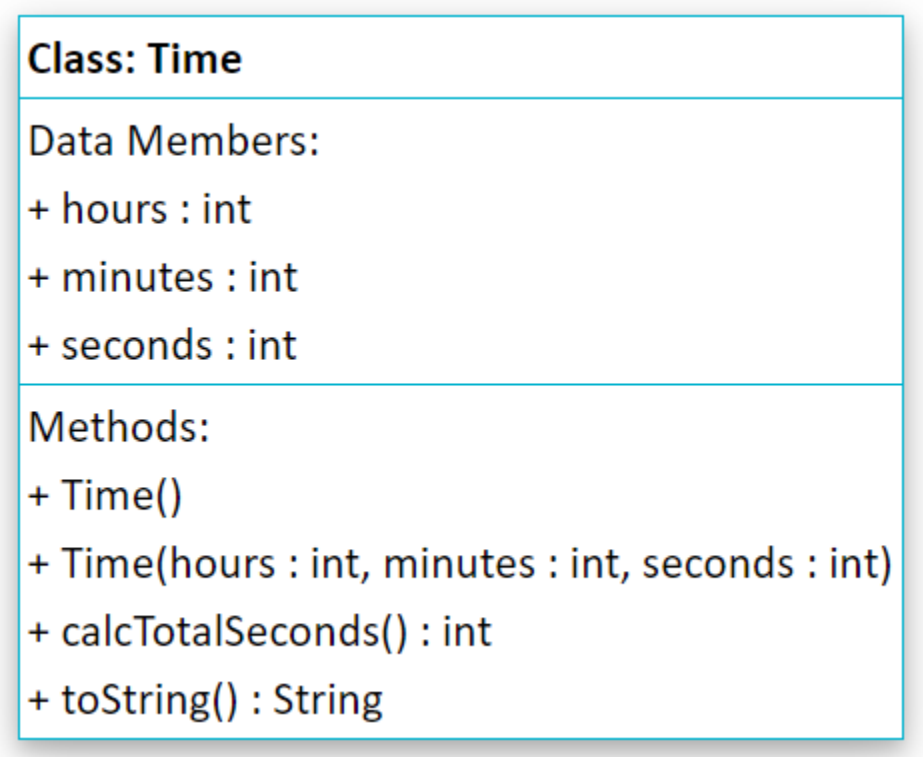


The calcArea() and calcCircumference() methods calculate and return the area and circumference of the circle (Use Google if you need to find the formulas). The default constructor sets radius to a default value of 1. The toString() method returns a string representation of the circle in the form:

```
Circle: radius=x.x
```

(where x.x is the radius value, formatted to 1 decimal place)

2.

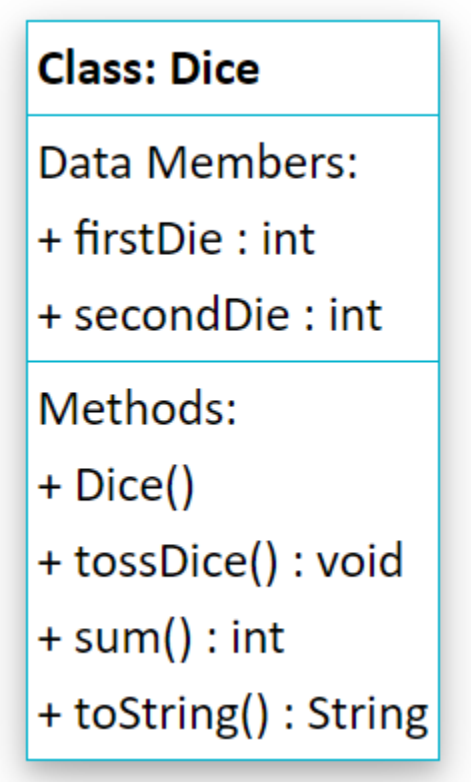


This class models a specific time stamp on a clock. The calcTotalSeconds() method calculates and returns the total number of seconds from midnight to the time represented by the time object (use the normal calculation for total number of seconds: hours * 3600 + minutes * 60 + seconds). The default constructor sets the time to midnight by setting each attribute to 0. The toString() method returns a string representation of the time object in the form:

`hh:mm:ss`

(where hh, mm, and ss are the values of hours, minutes and seconds in the time object, formatted to show exactly 2 digits)

3.



The tossDice() method generates to random integers from 1 to 6 and places one in each instance variable. The sum method returns the sum of the two dice values. The default constructor tosses the dice so that they are initialized with a random set of values. The toString() method returns a string representation of the pair of die in the form:

```
x, y
```

(where x and y are the two instance variable values)

Test the dice class in a Main class: construct the dice and allow the user to play a dice game that is a variation on the classic dice game called "Chicago": A player rolls a pair of dice 11 times. In those 11 rolls, they must attempt to roll each value from 2 to 12. At the end of the 11 rolls, total up the successful rolls to get their score.

In each round, display the dice roll and the sum of that roll in parentheses. For example, if the user rolled a 4 and a 3, then the value 4, 3 (7) is displayed.

After all rounds are completed, display a list of results: List each value from 2 to 12 and whether or not the user managed to roll that value during their 11 rounds.

After displaying the results of the 11 rounds, display the total points earned.

The sample program interaction and output below can better demonstrate how the game works:

```
Starting the game!
Round 2...
6, 3 (9)
Round 3...
1, 5 (6)
Round 4...
4, 4 (8)
Round 5...
6, 1 (7)
Round 6...
3, 1 (4)
Round 7...
1, 1 (2)
Round 8...
4, 3 (7)
Round 9...
4, 4 (8)
Round 10...
3, 4 (7)
Round 11...
5, 5 (10)
Round 12...
5, 2 (7)

Your results:
 2: yes
 3: no
 4: yes
 5: no
 6: yes
 7: yes
 8: yes
```

```
 9: yes
10: yes
11: no
12: no
Your total score: 46
```

Hint: use a boolean array to keep track of which values from 2 to 12 the player has already rolled. Each round, check to see if the array element corresponding to that round number is set to false: if so, then that number hasn't been rolled yet, so you can count it.