



## Java Lab – Classes & Constructors

**IMPORTANT!** Save all your work to a safe location such as oneDrive.

Create a folder for SDPD into which you will save all your work for this module, arranged how you wish. Ideally you should create a folder each week for your lab exercises. Note that you should create **a separate file** for each exercise.

## Exercise 1

Create a class called *Circle* with one private field called *radius*. Create a **no-arguments** constructor for the class that sets the default value of *radius* to 2, e.g.:

```
public class Circle
{
    private int radius;

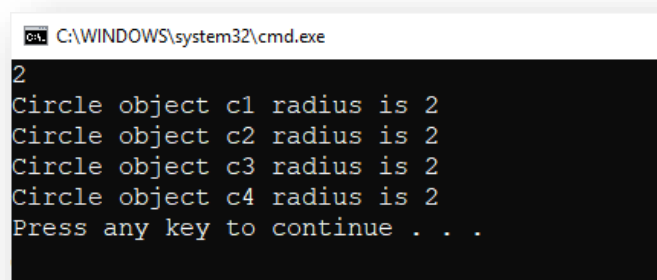
    public Circle()
    {
        radius = 2;
    }

    public int getRadius()
    {
        return radius;
    }
}
```

Create a program with a main method that creates an instance of the circle, as shown below.

```
public class Exercise1
{
    public static void main(String[] args)
    {
        //Create an instance of the Circle class
        Circle c1 = new Circle();

        //Call the getRadius() method for the c1 object
        System.out.println(c1.getRadius());
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
2
Circle object c1 radius is 2
Circle object c2 radius is 2
Circle object c3 radius is 2
Circle object c4 radius is 2
Press any key to continue . . .
```

## Exercise 2

Create a class called *Rectangle* with two private fields, width and length. Create a ***no-arguments*** constructor for the class, that sets default values of 1 for both the width and length fields, e.g.:

```
public class Rectangle
{
    private int width;
    private int length;

    public Rectangle()
    {
        width = 1;
        length = 1;
    }
}
```

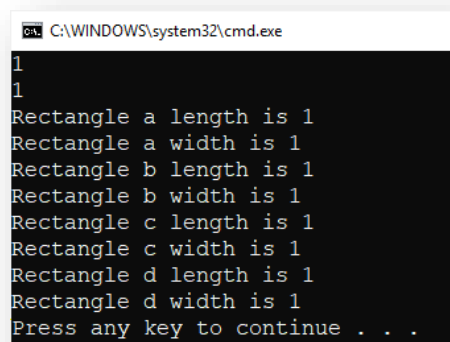
Write a Java program with a main method that creates an object using the rectangle class.

```
public class RectangleExercise
{
    public static void main(String[] args)
    {
        Rectangle a = new Rectangle();

        System.out.println(a.getLength());
        System.out.println(a.getWidth());
    }
}
```

Write appropriate ***getter*** methods for the class to verify the default length and width of the Rectangle object. Your object should display a length and width of 1.

Amend your program so that it creates four instances of the Rectangle in total. Confirm that they all have the same default Length and Width by outputting an appropriate message to the console.



```
C:\WINDOWS\system32\cmd.exe
1
1
Rectangle a length is 1
Rectangle a width is 1
Rectangle b length is 1
Rectangle b width is 1
Rectangle c length is 1
Rectangle c width is 1
Rectangle d length is 1
Rectangle d width is 1
Press any key to continue . . .
```

## Exercise 3

Create a class called *Employee* with two private fields, name and age. Create a constructor with one parameter (a String for the name), similar to as shown:

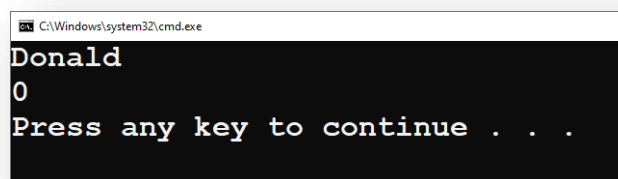
```
public class Employee
{
    private String name;
    private int age;

    public Employee(String name)
    {
        this.name = name;
    }
}
```

Create appropriate accessor and mutator methods as required in the class. Write a program with a main method that creates an instance of the employee class, using the constructor function, e.g:

```
Employee emp1 = new Employee("Donald");
```

Use *getter* methods to confirm that your constructor function works as required by outputting the values of both fields to the console, e.g.:

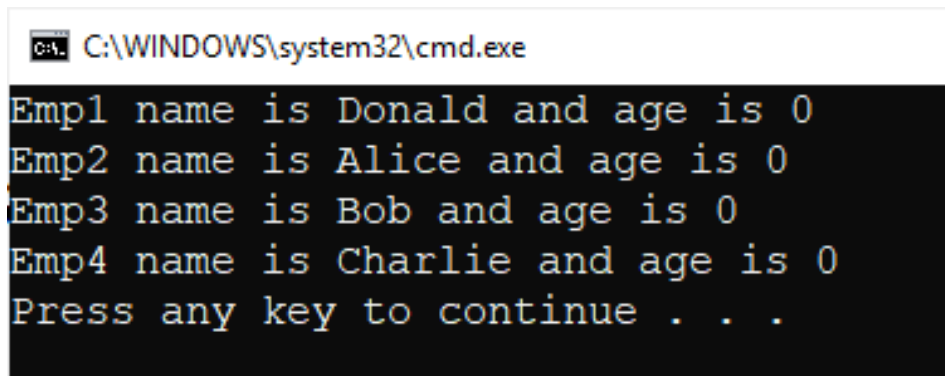


```
C:\Windows\system32\cmd.exe
Donald
0
Press any key to continue . . .
```

Create another three Employee instances for Alice, Bob and Charlie, e.g:

```
Employee emp1 = new Employee("Donald");
Employee emp2 = new Employee("Alice");
Employee emp3 = new Employee("Bob");
Employee emp4 = new Employee("Charlie");
```

Output both fields for each object to the console, e.g;



```
C:\WINDOWS\system32\cmd.exe
Emp1 name is Donald and age is 0
Emp2 name is Alice and age is 0
Emp3 name is Bob and age is 0
Emp4 name is Charlie and age is 0
Press any key to continue . . .
```

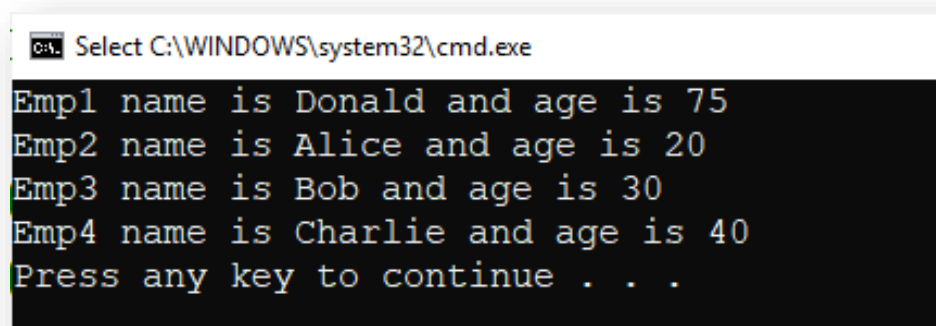
Amend your constructor so that it accepts two arguments – a String for the name and an int for the age, e.g.:

```
public Employee(String name, int age)
{
    this.name = name;
    this.age = age;
}
```

Test your program again using your new constructor, e.g.:

```
Employee emp1 = new Employee("Donald", 75);
Employee emp2 = new Employee("Alice", 20);
Employee emp3 = new Employee("Bob", 30);
Employee emp4 = new Employee("Charlie", 40);
```

Output your results to the console again to confirm that this worked correctly:

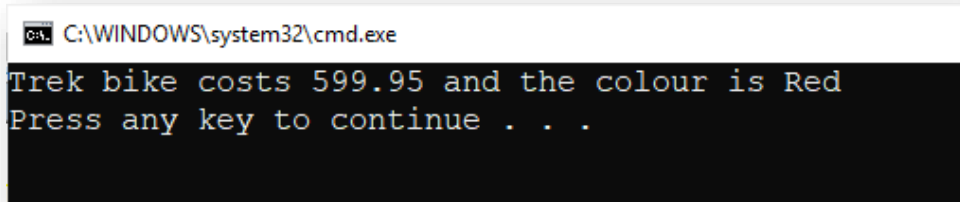


```
Select C:\WINDOWS\system32\cmd.exe
Emp1 name is Donald and age is 75
Emp2 name is Alice and age is 20
Emp3 name is Bob and age is 30
Emp4 name is Charlie and age is 40
Press any key to continue . . .
```

## Exercise 4

Create a class called Bike with two fields – one for the price(double), and one for the colour (String). Create a constructor method for the class that has two parameters – one for the price and one for the colour. Create an object from this class for a 'Trek' bike that costs 599.95 and that is red. Output the results to the console using a *getter* method for each field, eg:

```
System.out.println("Trek bike costs " + trek.getPrice() + " and the colour is " + trek.getColour());
```



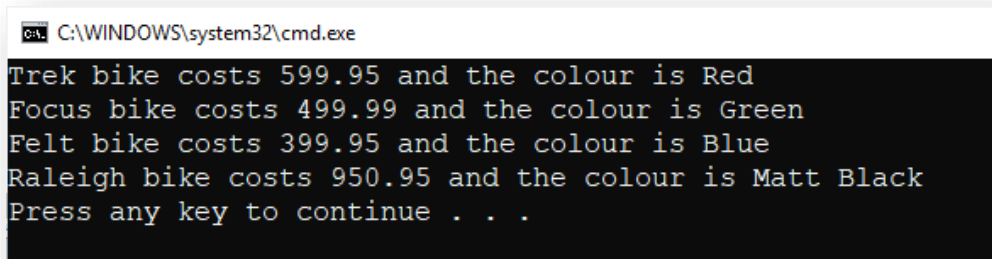
```
C:\WINDOWS\system32\cmd.exe
Trek bike costs 599.95 and the colour is Red
Press any key to continue . . .
```

Create an additional three objects for the following:

Felt, cost 399.95 and colour is Blue

Raleigh cost is 950.95 and colour is Matt Black

Output the results to the console using a *getter* method for each field:



```
C:\WINDOWS\system32\cmd.exe
Trek bike costs 599.95 and the colour is Red
Focus bike costs 499.99 and the colour is Green
Felt bike costs 399.95 and the colour is Blue
Raleigh bike costs 950.95 and the colour is Matt Black
Press any key to continue . . .
```

## Exercise 5

Create a class called *Student* with three fields for first name (String), surname (String) and age (integer). Create **two** constructor methods for the class. One of the methods should accept a single string as an argument to be used for the first name field. The second constructor should accept two string arguments to be used for both the first name and the surname. For example:

```
public class Student
{
    private String firstname;
    private String surname;
    private int age;

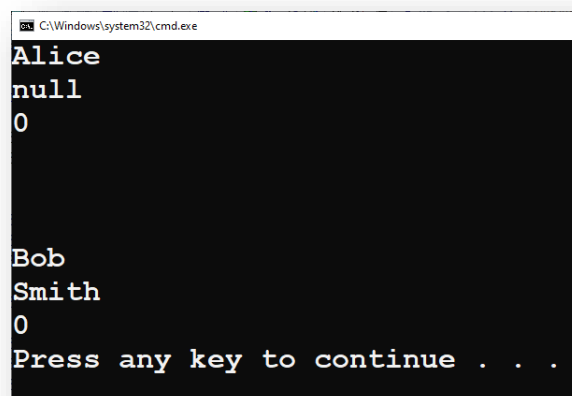
    public Student(String firstname) ← First constructor, with one parameter
    {
        this.firstname = firstname;
    }

    public Student(String firstname, String surname) ← Second
    {                                     constructor,
        this.firstname = firstname;         with two
        this.surname = surname;             parameters
    }
}
```

Write appropriate accessor and mutator methods as required for the student class. Create a program to test that the class works as expected. In this program, create two student objects using **both** constructors. For example:

```
Student s1 = new Student("Alice"); ← This will use the first constructor
Student s2 = new Student("Bob", "Smith"); ← This will use the second constructor
                                         as two parameters
                                         are supplied
```

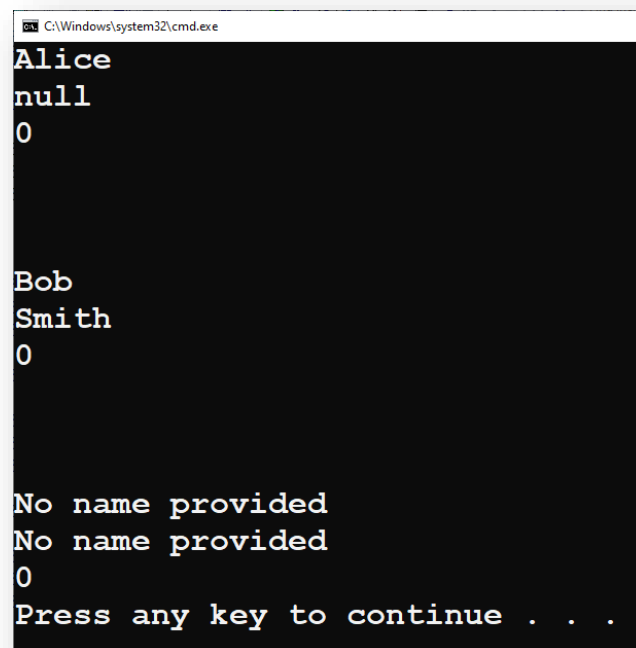
Confirm that both constructors work by outputting the result to the console:



```
C:\Windows\system32\cmd.exe
Alice
null
0

Bob
Smith
0
Press any key to continue . . .
```

Create a third **no arguments** constructor that sets the value of first name and surname to “No name provided”. Create a third object called s3 to test this constructor:



```
C:\Windows\system32\cmd.exe
Alice
null
0

Bob
Smith
0

No name provided
No name provided
0
Press any key to continue . . .
```



## Exercise 6

Create a class named *Car*. Data fields should be a String for car make, a String for the model, a double for the engine size, an int for the year of manufacture, and a String for colour. The class should have getters and setters for each field.

In addition, the class should also contain three constructor methods, as follows:

A **no-argument** constructor that will set the year to 2022;

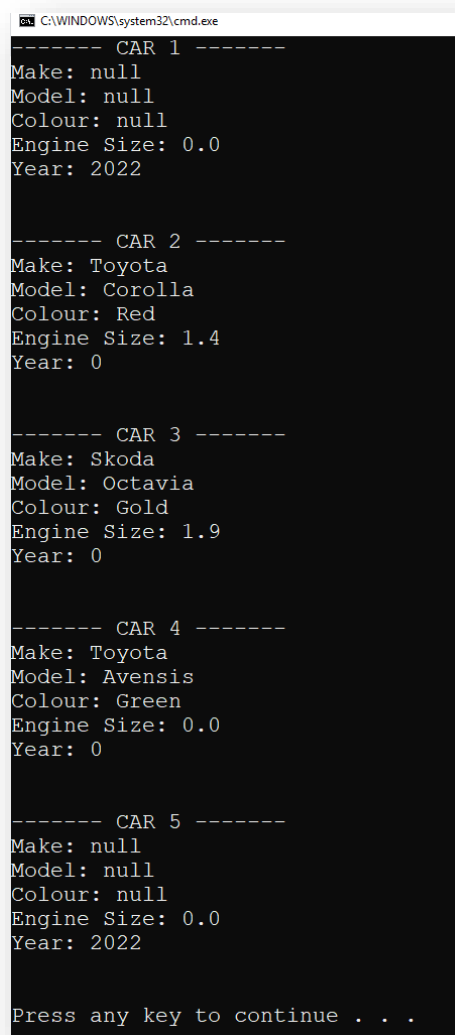
A **3-argument** constructor with parameters for make, model and colour.

A **5-argument** constructor with parameters for make, model, engine size, year and colour.

Create a java program called exercise6test, and create five Car objects with the following:

```
Car car1 = new Car();
Car car2 = new Car("Toyota", "Corolla", 1.4, 2019, "Red");
Car car3 = new Car("Skoda", "Octavia", 1.9, 2019, "Gold");
Car car4 = new Car("Toyota", "Avensis", "Green");
Car car5 = new Car();
```

Confirm that all 5 objects have been created successfully by outputting the fields of each object to the console:



```
C:\WINDOWS\system32\cmd.exe
----- CAR 1 -----
Make: null
Model: null
Colour: null
Engine Size: 0.0
Year: 2022

----- CAR 2 -----
Make: Toyota
Model: Corolla
Colour: Red
Engine Size: 1.4
Year: 0

----- CAR 3 -----
Make: Skoda
Model: Octavia
Colour: Gold
Engine Size: 1.9
Year: 0

----- CAR 4 -----
Make: Toyota
Model: Avensis
Colour: Green
Engine Size: 0.0
Year: 0

----- CAR 5 -----
Make: null
Model: null
Colour: null
Engine Size: 0.0
Year: 2022

Press any key to continue . . .
```

Amend the details for car5 only by using the setter methods so that the details after updating are as follows:

```
C:\WINDOWS\system32\cmd.exe

----- CAR 1 -----
Make: null
Model: null
Colour: null
Engine Size: 0.0
Year: 2022

----- CAR 2 -----
Make: Toyota
Model: Corolla
Colour: Red
Engine Size: 1.4
Year: 0

----- CAR 3 -----
Make: Skoda
Model: Octavia
Colour: Gold
Engine Size: 1.9
Year: 0

----- CAR 4 -----
Make: Toyota
Model: Avensis
Colour: Green
Engine Size: 0.0
Year: 0

----- CAR 5 -----
Make: Nissan
Model: Qashqai
Colour: Black
Engine Size: 1.6
Year: 2012

Press any key to continue . . .
```

## Exercise 7

Create a class named *Sandwich*. Data fields should include a String for the main ingredient (such as “tuna”), a String for bread type (such as “wheat”, “White” or “Ciabatta”), and a double for price (such as 4.99). Include methods to get and set values for each of these fields. Save the class as **Sandwich.java**.

Create an application named TestSandwich that instantiates one Sandwich object and demonstrates the use of the *set* and *get* methods. Save this application as **TestSandwich.java**.

```
C:\WINDOWS\system32\cmd.exe
You have ordered a tuna sandwich on wheat bread, and the price is 4.99
Press any key to continue . . .
```

Create constructors so that sandwich objects can be created by passing the following:

main ingredient, bread type and price

or

or

price

You should also create a **no arguments** constructor that will assign the following default values to the object:

*main ingredient: cheese*

*bread type: sliced white*

*price: 3.50*

Sandwich 2 should use the 3 args constructor

Sandwich 3 should use the 1 arg constructor (for ingredient)

Sandwich 4 should use the 1 arg constructor (for price)

Sandwich 5 should use the no args constructor

Your output from these should be similar to as shown below:

```
C:\Windows\system32\cmd.exe
You have ordered a tuna sandwich on wheat bread, and the price is 4.99
Sandwich 2: You have ordered a Tuna sandwich on Brown bread, and the price is 2.99
Sandwich 3: You have ordered a Cheese sandwich on null bread, and the price is 0.0
Sandwich 4: You have ordered a null sandwich on null bread, and the price is 3.99
Sandwich 5: You have ordered a Cheese sandwich on Sliced White bread, and the price is 3.5
Press any key to continue . . .
```

## Exercise 8

Create a class named *Customer*. Data fields should include a String for the name, a String for the address, and a long for the phone number. Create getter and setter methods for all fields.

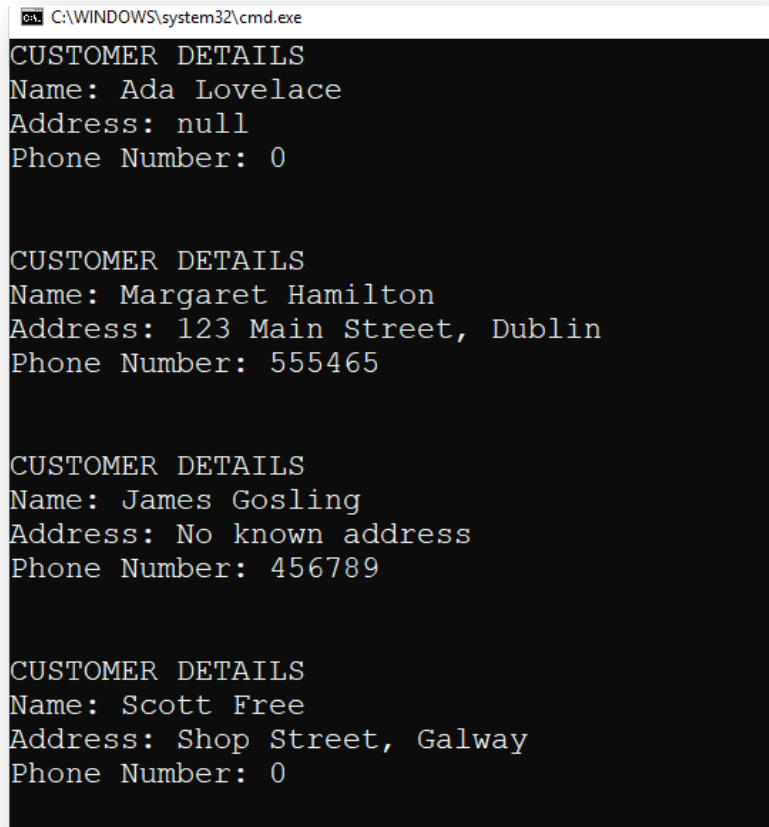
Create four constructor methods as follows:

- 1-arg constructor using the name only as a parameter
- 2-arg constructor with the name and address as parameters
- 2-arg constructor with the name and phone number as parameters
- 3-arg Constructor with the name, address, and phone number as parameters

Using the above constructors, create four customer objects with the following data:

- "Ada Lovelace"
- "Margaret Hamilton", "123 Main Street, Dublin", 555465
- "James Gosling", 456789
- "Scott Free", "Shop Street, Galway"

Confirm that the objects were created correctly but outputting to the console using the appropriate getter methods:



```
C:\WINDOWS\system32\cmd.exe
CUSTOMER DETAILS
Name: Ada Lovelace
Address: null
Phone Number: 0

CUSTOMER DETAILS
Name: Margaret Hamilton
Address: 123 Main Street, Dublin
Phone Number: 555465

CUSTOMER DETAILS
Name: James Gosling
Address: No known address
Phone Number: 456789

CUSTOMER DETAILS
Name: Scott Free
Address: Shop Street, Galway
Phone Number: 0
```

Amend your program to include a fifth customer object. Using Scanner, prompt for a name, address and phone number. Then create a new object using the 3-arg constructor, and confirm that this worked as expected by outputting to the console:

```
C:\WINDOWS\system32\cmd.exe
CUSTOMER DETAILS
Name: Ada Lovelace
Address: null
Phone Number: 0

CUSTOMER DETAILS
Name: Margaret Hamilton
Address: 123 Main Street, Dublin
Phone Number: 555465

CUSTOMER DETAILS
Name: James Gosling
Address: No known address
Phone Number: 456789

CUSTOMER DETAILS
Name: Scott Free
Address: Shop Street, Galway
Phone Number: 0

Enter name for customer 5: Joe Bloggs
Enter address for customer 5: 456 Sea View Street
Enter phone for customer 5: 456123

CUSTOMER DETAILS
Name: Joe Bloggs
Address: 456 Sea View Street
Phone Number: 456123

Press any key to continue . . .
```

## Exercise 9

The following table lists the freezing and boiling points of several substances.

Substance	Freezing Point	Boiling Point
Ethyl Alcohol	−113	78
Oxygen	−219	−188
Water	0	100

Design a class that stores a temperature in a temperature field and has the appropriate accessor and mutator methods for the field. In addition to appropriate constructors, the class should have the following methods:

- **isWaterFreezing**. This method should return the boolean value true if the temperature stored in the temperature field is at or below the freezing point of water. Otherwise, the method should return false.
- **isWaterBoiling**. This method should return the boolean value true if the temperature stored in the temperature field is at or above the boiling point of water. Otherwise, the method should return false.
- **isEthylFreezing**. This method should return the boolean value true if the temperature stored in the temperature field is at or below the freezing point of ethyl alcohol. Otherwise, the method should return false.
- **isEthylBoiling**. This method should return the boolean value true if the temperature stored in the temperature field is at or above the boiling point of ethyl alcohol. Otherwise, the method should return false.
- **isOxygenFreezing**. This method should return the boolean value true if the temperature stored in the temperature field is at or below the freezing point of oxygen. Otherwise, the method should return false.
- **isOxygenBoiling**. This method should return the boolean value true if the temperature stored in the temperature field is at or above the boiling point of oxygen. Otherwise, the method should return false.

Write a program that demonstrates the class. The program should ask the user to enter a temperature, and then display a list of the substances that will freeze at that temperature and those that will boil at that temperature. For example, if the temperature is 0 degrees, the class should report that water will freeze and oxygen will boil at that temperature.

Sample output:

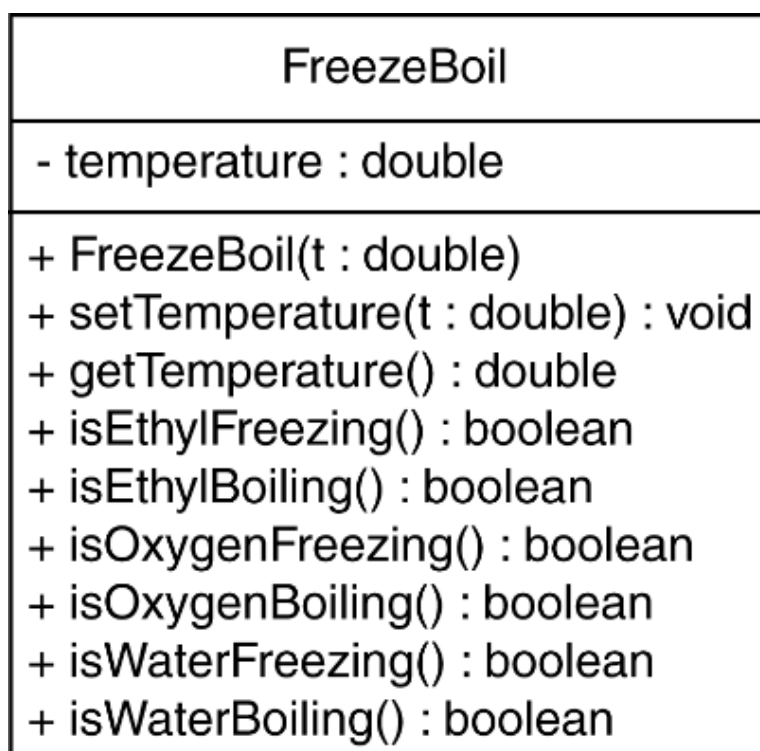
```
C:\Windows\system32\cmd.exe
Enter a temperature: 0
Water will freeze.
Oxygen will boil.
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Enter a temperature: 100
Ethyl alcohol will boil.
Oxygen will boil.
Water will boil.
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Enter a temperature: -250
Ethyl alcohol will freeze.
Oxygen will freeze.
Water will freeze.
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Enter a temperature: 50
Oxygen will boil.
Press any key to continue . . .
```

UML Diagram:



## Exercise 10

Write a class named Coin. The Coin class should have the following field:

- A String named sideUp.

The sideUp field will hold either “heads” or “tails” indicating the side of the coin that is facing up.

*The Coin class should have the following methods:*

- A no-arg constructor that randomly determines the side of the coin that is facing up (“heads” or “tails”) and initialises the sideUp field accordingly.
- A void method named toss that simulates the tossing of the coin. When the toss method is called, it randomly determines the side of the coin that is facing up (“heads” or “tails”) and sets the sideUp field accordingly.
- A method named getSideUp that returns the value of the sideUp field.

Write a program that demonstrates the Coin class. The program should create an instance of the class and display the side that is initially facing up. Then, use a loop to toss the coin 20 times. Each time the coin is tossed, display the side facing up. The program should keep a count of the number of times heads is facing up and the number of times tails is facing up, and display those values after the loop finishes.

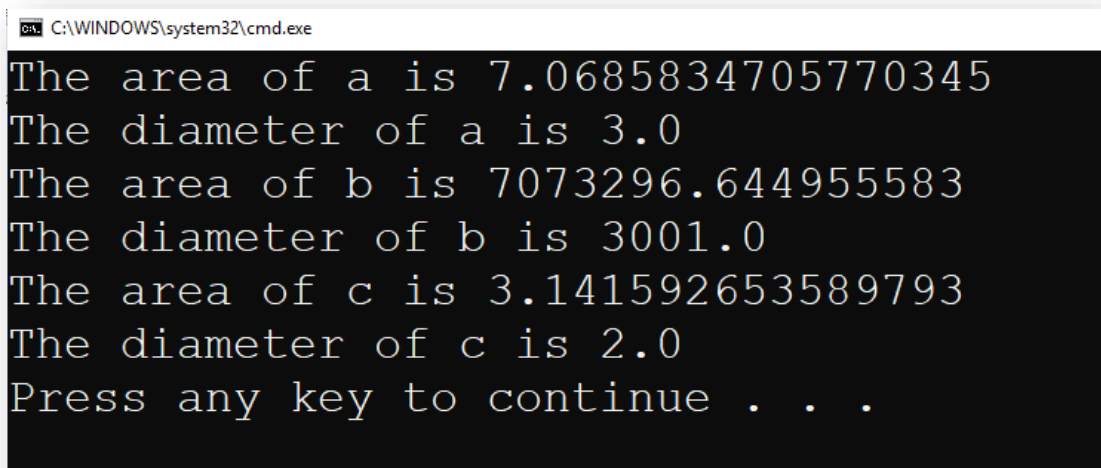
```
Select C:\Windows\system32\cmd.exe
The side initially facing up: heads
Now I will flip the coin 20 times.
Toss:  tails
Toss:  heads
Toss:  tails
Toss:  heads
Toss:  heads
Toss:  tails
Toss:  heads
Toss:  heads
Toss:  tails
Toss:  tails
Toss:  heads
Toss:  heads
Toss:  tails
Toss:  tails
Toss:  heads
Toss:  heads
Toss:  heads
Toss:  heads
Toss:  tails
Toss:  tails
Press any key to continue . . . _
```



## Exercise 11

Create a class named `Circle` with fields named `radius`, `diameter`, and `area`. Include a constructor that sets the radius to 1 and calculates the other two values. Also include methods named `setRadius()` and `getRadius()`. The `setRadius()` method not only sets the radius, it also calculates the other two values. The diameter of a circle is twice the radius, and the area of a circle is  $\pi$  (3.1415) multiplied by the square of the radius. Save the class as **Circle.java**.

Create a class named `TestCircle` whose `main()` method declares several `Circle` objects. Using the `setRadius()` method, assign one `Circle` a small radius value, and assign another a larger radius value. Do not assign a value to the radius of the third circle; instead, retain the value assigned at construction. Display all the values for all the `Circle` objects. Save the application as **TestCircle.java**.



```
C:\WINDOWS\system32\cmd.exe
The area of a is 7.0685834705770345
The diameter of a is 3.0
The area of b is 7073296.644955583
The diameter of b is 3001.0
The area of c is 3.141592653589793
The diameter of c is 2.0
Press any key to continue . . .
```

## Exercise 12

For this exercise, you will write a program that lets the user play against the computer in a variation of the popular blackjack card game. In this game variation, two six-sided dice are used instead of cards. The dice are rolled, and the player tries to beat the computer's hidden total without going over 21.

*Here are some suggestions for the game's design:*

- Each game round is performed as an iteration of a loop that repeats as long as the player agrees to roll the dice, and the player's total does not exceed 21.
- At the beginning of each round, the program will ask the user whether or not they want to roll the dice to accumulate points.
- During each round, the program simulates the rolling of two six-sided dice. It rolls the dice first for the computer, and then it asks the user whether they want to roll.
- The loop keeps a running total of both the computer's and the user's points.
- The computer's total should remain hidden until the loop has finished.
- After the loop has finished, the computer's total is revealed, and the player with the most points, without going over 21, wins.

*Sample Output:*

```
C:\Windows\system32\cmd.exe
Roll the dice? (y/n) : y
Points: 8
Roll the dice? (y/n) : y
Points: 19
Roll the dice? (y/n) : n

Game Over

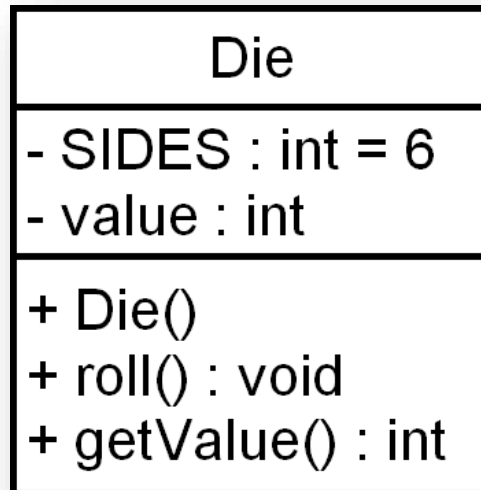
User's Points: 19
Computer's Points: 17
The user wins!
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Roll the dice? (y/n) : y
Points: 6
Roll the dice? (y/n) : y
Points: 17
Roll the dice? (y/n) : y
Points: 27

Game Over

User's Points: 27
Computer's Points: 11
The computer wins!
Press any key to continue . . .
```

UML Diagram:



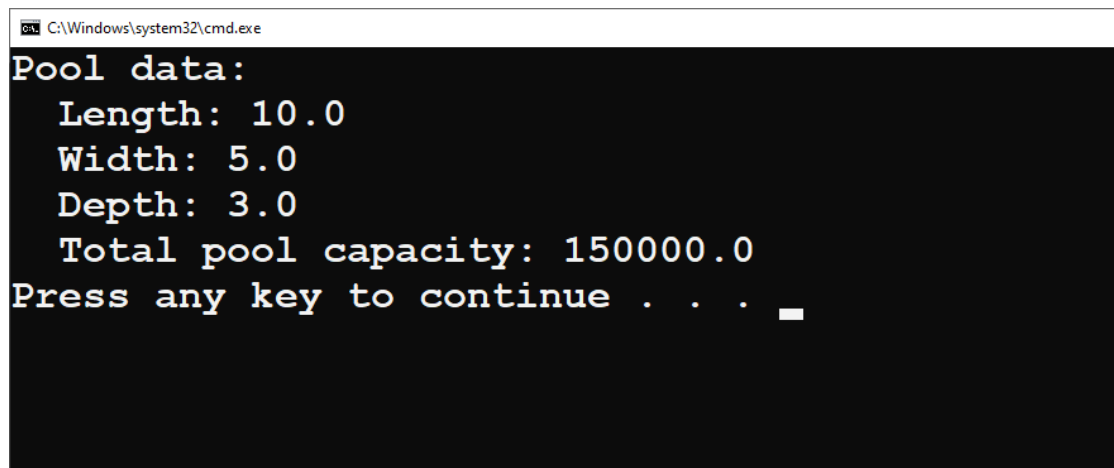
## Exercise 13

Write the definition of a `class`, `swimmingPool`, to implement the properties of a swimming pool. Your class should have the instance variables to store the length (in meters), width (in meters), depth (in meters).

Add a method to determine the amount of water needed to fill the pool:

The amount of water in litres required can be calculated with the following formula:

$\text{width} * \text{length} * \text{depth} * 1000;$



```
C:\Windows\system32\cmd.exe
Pool data:
Length: 10.0
Width: 5.0
Depth: 3.0
Total pool capacity: 150000.0
Press any key to continue . . .
```

## Exercise 14

For this exercise, you will write a program that simulates a fishing game. In this game, a six-sided die is rolled to determine what the user has caught. Each possible item is worth a certain number of fishing points. The points will remain hidden until the user is finished fishing, and then a message is displayed congratulating the user, depending on the number of fishing points gained.

*Here are some suggestions for the game's design:*

- Each round of the game is performed as an iteration of a loop that repeats as long as the player wants to fish for more items.
- At the beginning of each round, the program will ask the user whether or not they want to continue fishing.
- The program simulates the rolling of a six-sided die (use a class for this)
- Each item that can be caught is represented by a number generated from the die; for example, 1 for "a huge fish", 2 for "an old shoe", 3 for "a little fish", and so on.
- Each item the user catches is worth a different amount of points.
- The loop keeps a running total of the user's fishing points.
- After the loop has finished, the total number of fishing points is displayed, along with a message that varies depending on the number of points earned.

```
switch(itemNumber)
{
    case 1:
        itemName = "an old boot";
        points = 0;
        break;

    case 2:
        itemName = "a huge fish";
        points = 100;
        break;

    case 3:
        itemName = "a leaf";
        points = 1;
        break;

    case 4:
        itemName = "a little fish";
        points = 10;
        break;

    case 5:
        itemName = "a rock";
        points = 2;
        break;

    case 6:
        itemName = "weeds";
        points = 0;
        break;
}
```

Your output can be similar to the following:

```
C:\Windows\system32\cmd.exe
Let's go fishing!

You cast your line into the water...
You caught [a huge fish].

Try fishing again? (Y or N): y

You cast your line into the water...
You caught [a leaf].

Try fishing again? (Y or N): y

You cast your line into the water...
You caught [a little fish].

Try fishing again? (Y or N): y

You cast your line into the water...
You caught [weeds].

Try fishing again? (Y or N): n

You got a total of 111 fishing points.
Great Job! You are a natural!
Press any key to continue . . .
```

Display a message depending on the total fishing points earned.

If the total is 100 or greater, output the message - "Great Job! You are a natural!"

If the total is greater than 50, output the message - "That's some pretty fine fishing."

Otherwise, output the message - "The fish just aren't biting today. Maybe next time."