Sukkur Institute of Business Administration University
Department of Computer Science
BS – II (CS/SE/AI) Spring 2024

Object Oriented Programming
Lab # 05: To become familiar with Static Methods and Recursion

Instructor: Engr. Zainab Umair Kamangar

Lab Report Rubrics (Add the points in each column, then add across the bottom row to find the total score)					**Total Marks**
S.No	**Criterion**	**0.5**	**0.25**	**0.125**	
1	Accuracy	☐ Desired output	☐ Minor mistakes	☐ Critical mistakes	
2	Timing	☐ Submitted within the given time	☐ 1 day late	☐ More than 1 day late	

<u>Submission Profile</u>

Name:	Submission date (dd/mm/yy):
Enrollment ID:	Receiving authority name and signature:
Comments:	

Instructor Signature

Note: Submit this lab hand-out in the next lab with attached solved activities and exercises

Objectives

After performing this lab, students will be able to understand,

- Intro to heap and stack memory
- String advanced
  - Immutable Strings
  - Mutable strings (String Buffer and String builder)
  - String Regular Expressions
- Input ways
  - Scanner
  - JOptionPane
  - Java Command line arguments
  - Buffered reader
- Methods
- Types of Static Methods
- Method Overloading
- Recursion

Intro to heap and stack memory

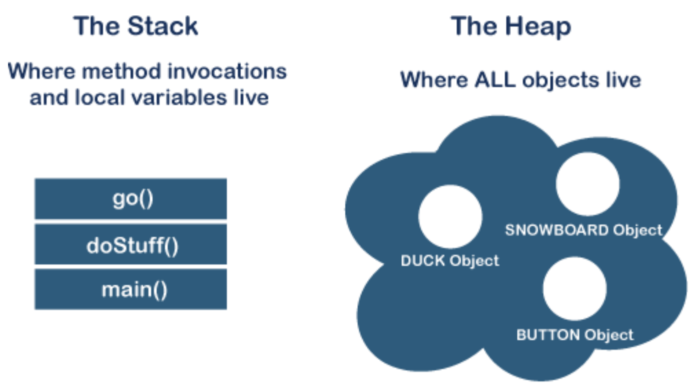
When we talk about **"objects,"** we often mean instances of classes that are referred to using **reference variables.** However, the reference type in C++ is different than the reference type in Java.

Explore More…

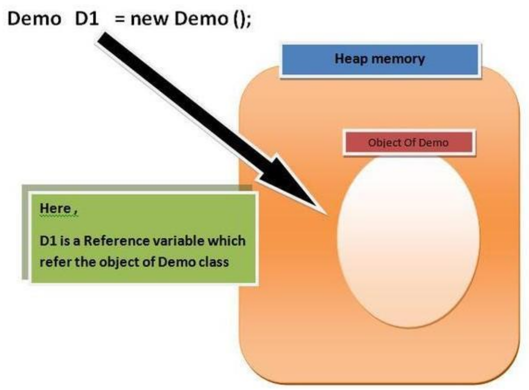
https://www.geeksforgeeks.org/reference-variable-in-java/

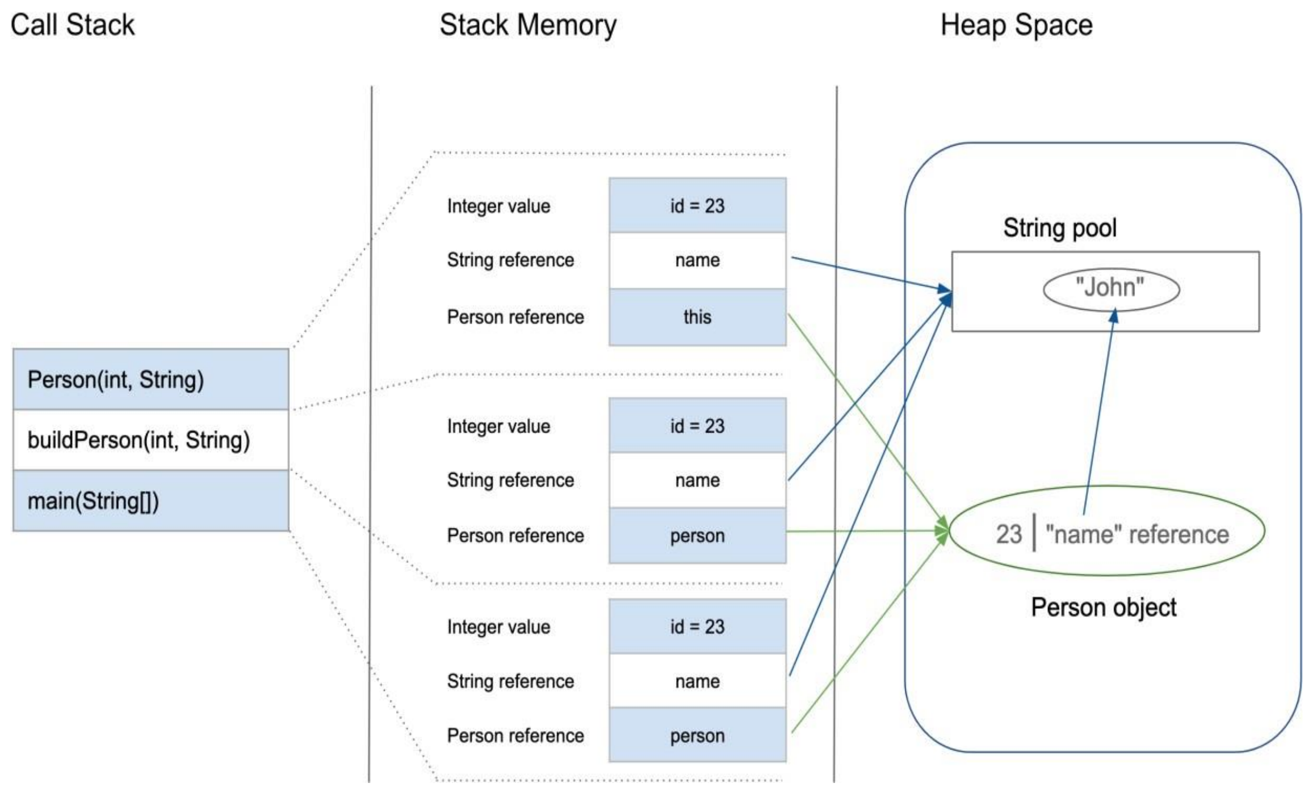
https://www.tutorialspoint.com/C-Cplusplus-Pointers-vs-Java-references

https://www.geeksforgeeks.org/is-there-any-concept-of-pointers-in-java/

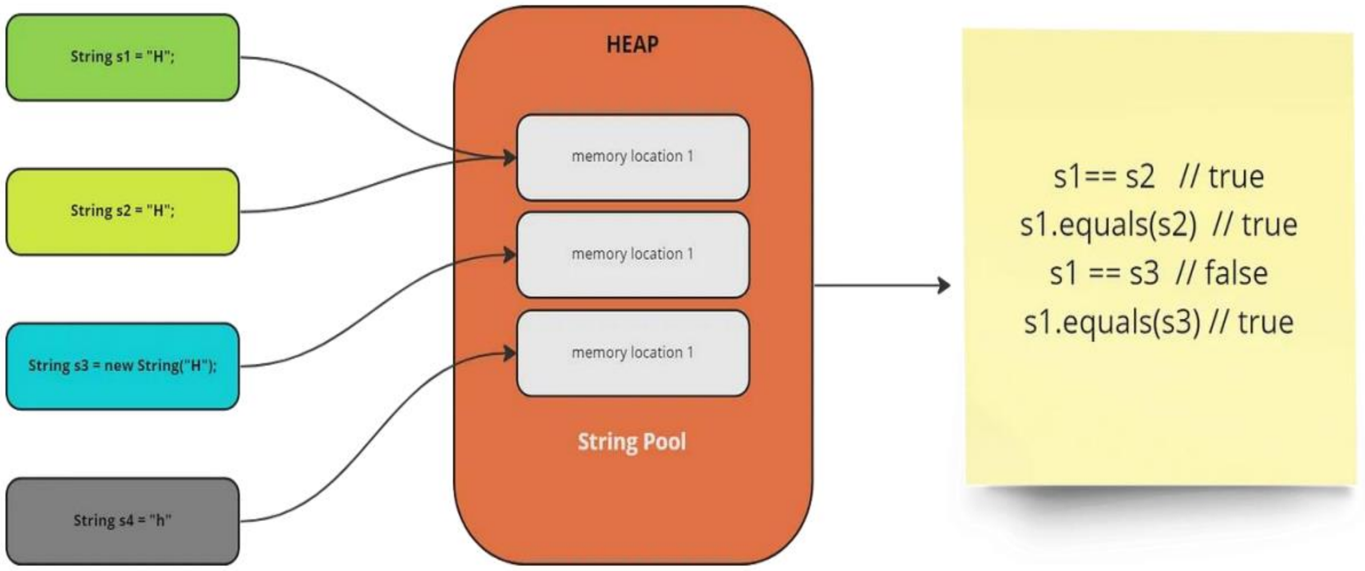


Stack Vs Heap


Demo D1 = new Demo ();
Heap memory
Object Of Demo
Here ,
D1 is a Reference variable which refer the object of Demo class



String advanced



```
s1== s2   // true
s1.equals(s2)  // true
s1 == s3  // false
s1.equals(s3) // true
```

Immutable strings

```
/*
This code is created to demonstrate "String pool" in the heap memory. In addition,it will also
demonstrate the Immutable strings
Before Executing This code:
You must be aware of the Heap and Stack memory concept
*/
public class Demo_String {
        public static void main(String args[])
        {
                String s1 = new String("Zainab");
                String s2 = new String("Zainab");
        System.out.println(s1==s2);//This statement will return False

        //Without the new keywords1 = "Zainab";
        s2= "Zainab";
        System.out.println(s1==s2);//This will return True

        //String are Immutable... Let's see
        System.out.println("Before append: "+ s1 + ": " + s1.hashCode());
        s1 += " Umair";
        System.out.println("After append: "+ s1 + ": " + s1.hashCode());
        //Hashes are different before and after append
        }
}
```

Mutable strings (String Buffer and String builder)

```
public class Demo_StringBuffer {
     public static void main(String args[])
     {
      StringBuffer sb = new StringBuffer("Zainab");
     //To append another string/text
     System.out.println("Before append:"+ sb + ": " + sb.hashCode());
     sb.append(" Umair");
     System.out.println("After append:"+ sb + ": " + sb.hashCode());
     System.out.println(sb.capacity());

     }
}
```

String Regular Expressions

```
import java.util.regex.*;
import java.util.*;
public class RegexExample {
        public static void main(String[] args)
        {
        Scanner sc = new Scanner(System.in);
        // Example 1: Checking if a String contains a specific pattern
        String text1 = "This is the sample text just to explore https://www.finance.gov.pk/";
        String pattern1 = "[a-zA-Z0-9._%+-]+gov.pk";
        boolean containsPattern = Pattern.compile(pattern1).matcher(text1).find();
        System.out.println("Contains 'xyz@gov.pk': " + containsPattern);

        // Example 2: To check whether the email is valid or not
        // Regex pattern for validating email addresses
        String emailPattern = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$";
        // Input email to be validated
        System.out.println("Enter your email");
        String email = sc.nextLine();
        // Replace with the email to be validated
        // Check if the email matches the pattern
        boolean isValidEmail = email.matches(emailPattern);
        System.out.println("The entered email is? :" + isValidEmail);

        // Example 3: Replacing parts of a String using regex
        String text3 = "The quick brown fox jumps over the lazy dog";
        String replacedText = text3.replaceAll("fox", "cat");
        System.out.println("Replaced text: " + replacedText);
        }
}
```

Input ways

JoptionPane

```
import javax.swing.JOptionPane;
public class JOptionPaneExample {
        public static void main(String[] args) {
        //String input(Default)
        String name = JOptionPane.showInputDialog("Enter your name:");
        JOptionPane.showMessageDialog(null, "Hello, " + name + "! Welcome.");

        //Integer Input
        String str = JOptionPane.showInputDialog("Enter Any Number: ");
        int num = Integer.parseInt(str);
        JOptionPane.showMessageDialog(null, "you entered" + num + "! Welcome.");

        //For float/double
        str = JOptionPane.showInputDialog("Enter Any Number: ");
        float num1 = Float.parseFloat(str);
        JOptionPane.showMessageDialog(null, "you entered" + num1 + "! Welcome.");
        }
}
```

Command Line Arguments

```
public class CMDArgumentsExample {
        public static void main(String[] args)
        {
        if(args.length > 0) {
            System.out.println("Arguments passed from command line:");
            for (String arg : args) {
                System.out.println(arg);
             }
        }
        else
             System.out.println("No arguments provided.");
        }
}
```

To run this code after compilation

```
java CMDArgumentsExample argument1 argument2 argument3
```

Buffered Reader

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class BufferedReaderExample {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Enter your name: ");
        String name = reader.readLine();
        System.out.println("Hello, " + name + "!");

        System.out.print("Enter your age: ");
        int age = Integer.parseInt(reader.readLine());
        System.out.println(“You are " + age + " years old.");

        reader.close();
    }
}
```

Output formatting

```
public class PrintfExample {
            public static void main(String[] args)
            {
            String name = "John";
            int age   =   30;
            double height = 6.1;

            System.out.printf("Name: %s, Age: %d, Height: %.1f\n", name, age, height);
            System.out.println();
            // Format strings to be placed in fixed width
            System.out.printf("Name: %-10s, Age: %-5d, Height: %-5.1f\n", name, age,height);

            double d = 4.5; System.out.println(d);
            // Shows 2 digits after the decimal point
            System.out.println(String.format("floatValue: %.2f", d));
            // Shows 2 digitsafter the decimal point
            }
}
```

Arrays in java

https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

https://www.hackerrank.com/challenges/java-1d-array-introduction/problem?isFullScreen=true

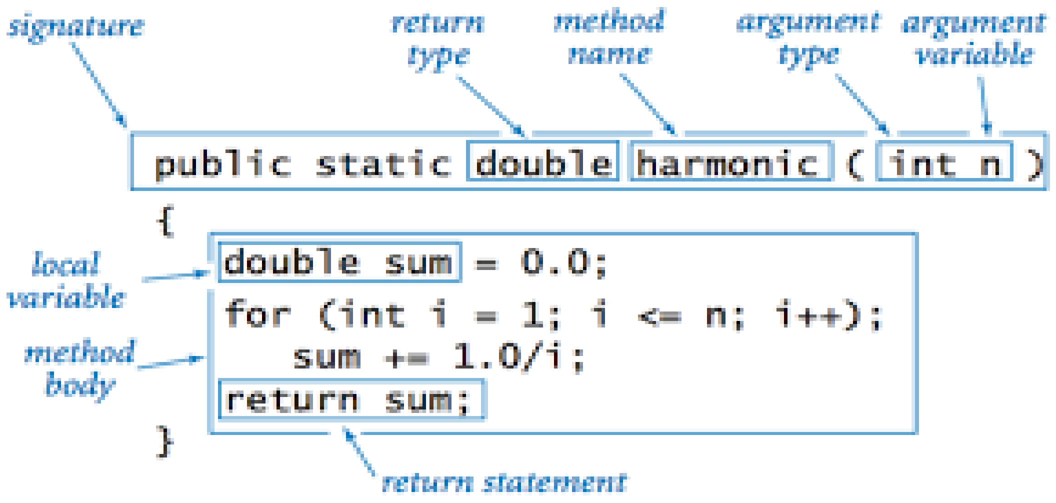
https://www.w3resource.com/java-exercises/array/index.php

Static Methods

A method (function) is a group of statements that is executed when it is called from some point in the program.

Static methods are the methods in Java that can be called without creating an object of class. They are referenced by the class name itself or reference to the Object of that class. Static method belongs to the class rather than the object of a class

The following is its format:



```
public static double harmonic ( int n )
{
    double sum = 0.0;
    for (int i = 1; i <= n; i++);
       sum += 1.0/i;
    return sum;
}
```

Where:

- ***return_type*** is the data type specifier of the data returned by the function.
- ***method_name*** is the identifier by which it will be possible to call the function.
- ***parameters*** (as many as needed): Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for example: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.
- ***statements*** are the function's body. It is a block of statements surrounded by {}

```
//Java Program to demonstrate the use of a static method.
class Student{
int rollno;
String name;
static String college = "ITS";
//static method to change the value of static variable
static void change(){
college = "BBDIT";
}
//constructor to initialize the variable
Student(int r, String n){
rollno = r;
name = n;
}
//method to display values
void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
public static void main(String args[]){
Student.change();//calling change method
//creating objects
Student s1 = new Student(111,"Engr");
Student s2 = new Student(222,"Zainab");
Student s3 = new Student(333,"Umair");
//calling display method
s1.display();
s2.display();
s3.display();
}
}
```

Restrictions

a. The static method cannot use non-static data member or call non-static method directly.
b. this and super cannot be used in static context.

Why is the Java main method static?

This is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first and then calls main( ) method which will lead to the problem of extra memory allocation

Types of static Methods

Methods without Return Type and No Parameter

```
public static void method_name ( )
```

What we will do in this case for similar functionality code, create a function named drawLine() and call that function at repeated code locations.

```
public class Lab05 {
   public static void main(String [] args){
   drawLine();
   System.out.println("Object Oriented Programming");
   drawLine();
   System.out.println("Lab 05");
   drawLine();
   System.out.println("Department of Computer Sciences");
   drawLine();
   }
  public static void drawLine(){;
    for(int i=1;i<30;i++)
      System.out.print("*");
      System.out.println();
   }
}
```

Methods without Return Type and with Parameters

```
public static void method_name ( par1, par2, par3 )
```

drawLine method with parameter to draw line according to the size provided in an argument.

```
public class Lab05 {
public static void main(String [] args){
     drawLine(30);
     System.out.println("Object Oriented Programming");
     drawLine(10);
     System.out.println("Lab 05");
     drawLine(10);
     System.out.println("Department of Computer Sciences");
     drawLine(30);
  }
public static void drawLine(int n){
  for(int i=1;i<n;i++)
     System.out.print("*");
     System.out.println();
  }
}
```

Methods With Return Type but no Parameter

```
public static return_type method_name ( )
```

```
public class GetNameExample {
 public static void main(String [] args) {
  System.out.println("The  University Name is "+getUniversityName());
 }//main ends
 public static String getUniversityName(){
  return "Sukkur IBA University";
 }
}//class ends
```

Methods With Return Type and Parameters

```
public static return_type method_name (par1, par2, par3)
```

```
public class ArraySum {
  public static void main(String [] args) {
    int [] a = {2,3,5,8,4,9,7,6,7,8};
         int sum = findSum(a);//invoking method with array argument
    System.out.println("The result is "+sum);
  }//main ends
  public static int findSum(int[] b){
    int total=0;
    for(int i=0; i<b.length;i++){
       total+=b[i]; return total;
    }
  }
}//class ends
```

There are two types of parameter passing:

1. **Pass by Value**

```
public static void sum(int a, int b)
```

- Call By value (copy of value) is when primitive data types are passed in the method call.

2. **Pass by Reference (value of memory address location)**

```
public static void displayCars(Car car)
```

- Objects and object variables are passed by reference or address.

Method Overloading

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different parameters.

OR

Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both.

Example:

```
// overloading in Java.
public class Sum {

    // Overloaded sum(). This sum takes two int parameters
    public int sum(int x, int y)
    {
        return (x + y);
    }

    // Overloaded sum(). This sum takes three int parameters
    public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }

    // Overloaded sum(). This sum takes two double parameters
    public double sum(double x, double y)
    {
        return (x + y);
    }

    // Driver code
    public static void main(String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}
```

Three ways to overload a method

In order to overload a method, the parameters of the methods must differ in either of these:

1. **Number of parameters.**

   For example: This is a valid case of overloading

   ```
   add(int, int)
   add(int, int, int)
   ```

2. **Data type of parameters.**

   For example:

   ```
   add(int, int)
   add(int, float)
   ```

3. **Sequence of Data type of parameters.**

   For example:

   ```
   add(int, float)
   add(float, int)
   ```

Invalid case of method overloading:

If two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

```
int add(int, int)
float add(int, int)
```

Recursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using recursive algorithm, certain problems can be solved quite easily. It does this by making problem smaller (simpler) at each call.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

Recursive functions have two important components:

1. **Base case** (i.e., when to stop), where the function directly computes an answer without calling itself. Usually, the base case deals with the simplest possible form of the problem you're trying to solve. The base case returns a value without making any subsequent recursive calls. It does this for one or more special input values for which the function can be evaluated without recursion.
2. **Recursive case** (i.e., call ourselves), where the function calls itself as part of the computation.

Perhaps the simplest example is calculating factorial: $n! = n \cdot (n-1) \cdot \cdots 2 \cdot 1$. However, we can also see that $n! = n \cdot (n-1)!$. Thus, factorial is defined in terms of itself. For example,

factorial(5) = 5 * factorial(4)
= 5 * (4 * factorial(3))
= 5 * (4 * (3 * factorial(2)))
= 5 * (4 * (3 * (2 * factorial(1))))
= 5 * (4 * (3 * (2 * (1 * factorial(0)))))
= 5 * (4 * (3 * (2 * (1 * 1))))
= 5 * 4 * 3 * 2 * 1 * 1 = 120

We can trace this computation in precisely the same way that we trace any sequence of function calls.

```
factorial(5)
  factorial(4)
    factorial(3)
      factorial(2)
        factorial(1)
          return 1
        return 2*1 = 2
      return 3*2 = 6
    return 4*6 = 24
  return 5*24 = 120
```

When a recursive call is made, new storage locations for variables are allocated on the stack

Exercises

Question 1: (Rectangle class)
Write a Java class called Rectangle with attributes **width and height**. Include methods to calculate the **area and perimeter** of the rectangle.

Create two instances/ reference variables of the Rectangle class:

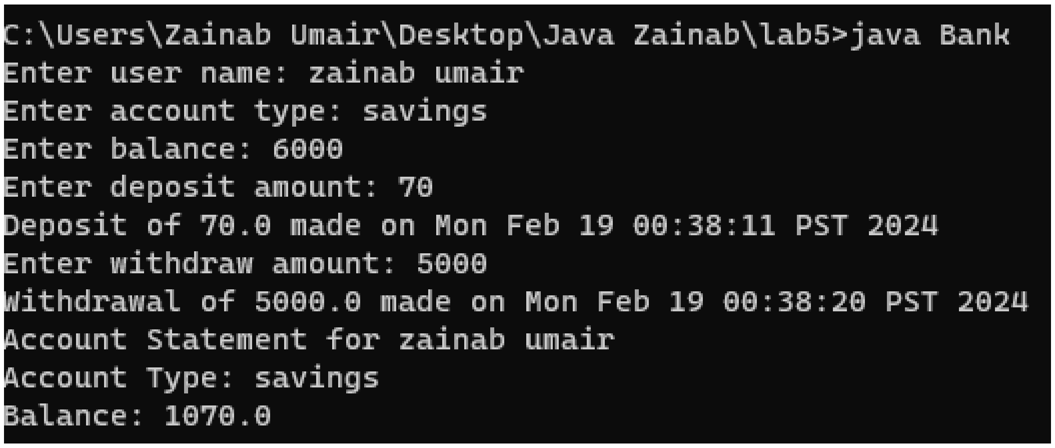
- **Set Object's instance variables using Scanner**: set the values of the instance variables using the Scanner class.
- **Set Object's instance variables using JOptionPane**: set the values instance variables using JOptionPane.
- **Call the Methods**: Call both methods of area and perimeter and display the output.

Question 2: (Generate random passwords)

Write a code that generates a random password of aspecified length, using a combination of letters, numbers, and symbols.

Question: 3
Write a program by creating Bank class having the following methods

- getInfo ( ) which takes user name, type of account, and balance.
- Cashdeposit ( ) which takes the deposit amount, and date of deposit.
- Withdraw( ) which takes the withdraw amount, and date of withdraw
- Statement ( ) that will display complete statement. Display final balance.



```
C:\Users\Zainab Umair\Desktop\Java Zainab\lab5>java Bank
Enter user name: zainab umair
Enter account type: savings
Enter balance: 6000
Enter deposit amount: 70
Deposit of 70.0 made on Mon Feb 19 00:38:11 PST 2024
Enter withdraw amount: 5000
Withdrawal of 5000.0 made on Mon Feb 19 00:38:20 PST 2024
Account Statement for zainab umair
Account Type: savings
Balance: 1070.0
```

Question: 4 (Static Method)

Write a static method to find the maximum element in an array of integers.

Question: 5 (FactorialDetector.java)

Write a program that takes input from the user. It tells the factorial number of the input.

Output

```
Enter any input: 24
24 is the factorial of: 4
Enter any input: 6
6 is the factorial of: 3
Enter any input: 10
10 has no any factorial!
```

Question: 6 (Recursion)

Create a recursive function to check if a given number is prime or not.

Question: 7 (Method Overloading)

Write a program by creating class MethodOverloading that contains following methods

- Computations (int length, int width). It takes length and width and displays Area of Rectangle.
- Computations (String name, int age) it takes name, and age. Display name, and age.
- Computations (double mass, double acceleration) It takes mass and acceleration and displays force.

Question: 8 (Recursion)

Write a recursive function to find the power of a number given its base and exponent.

Answer the following Questions:

1. Do you know about access modifiers? Write down different types of access modifiers with suitable examples.
2. What are non-access modifiers? Also, describe all non-access modifiers in Java.
3. Can constructors be static in Java? Try it out and justify it.
4. Why use iterations when we have recursion and vice versa?
5. Can we overload by return type?