Vidyavardhini's

College of Engineering & Technology

Vasai Road (W)

Department of Artificial Intelligence & Data Science

Laboratory ManualStudent Copy

Semester	III	Class	S.E.
Course Code	CSL304		
Course Name	Skill based Lab Course: Object Oriented Programming with Java		

Vidyavardhini's College of Engineering& Technology

Vision

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

Mission

- To provide technologically inspiring environmentfor learning.
- To promote creativity, innovation and professional activities.
- To inculcate ethical and moral values.
- To cater personal, professional and societal needs through quality education.

Department Vision:

To foster proficient artificial intelligence and data science professionals, making remarkable contributions to industry and society.

Department Mission:

- To encourage innovation and creativity with rational thinking for solvingthe challenges in emerging areas.
- To inculcate standard industrial practices and security norms whiledealing with Data.
- To develop sustainable Artificial Intelligence systems for the benefit ofvarious sectors.

Program Specific Outcomes (PSOs):

PSO1: Analyze the current trends in the field of Artificial Intelligence & Data Science and convey their findings by presenting / publishing at a national / international forum.

PSO2: Design and develop Artificial Intelligence & Data Science based solutions and applications for the problems in the different domains catering to industry and society.

Program Outcomes (POs):

Engineering Graduates will be able to:

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12. Life-long learning:** Recognize the need for and have the preparation and ability toengage in independent and life-long learning in the broadest context of technological change.

Course Objective

1	To learn the basic concept of object-oriented programming
2	To study JAVA Programming language
3	To study various concepts of JAVA programming like multithreading, exception handling, packages etc.
4	To explain components of GUI based application.

Course Outcomes

CO	At the end of course students will be able to:	**Action verbs**	**Bloom's Level**
CSL304.1	Apply the Object Oriented Programming and basic programming constructs for solving problems using JAVA.	Apply	Apply (level 3)
CSL304.2	Apply the concept of packages, classes , objects and accept the input using Scanner and Buffered Reader Class.	Apply	Apply (level 3)
CSL304.3	Apply the concept of strings, arrays, and vectors to perform various operations on sequential data.	Apply	Apply (level 3)
CSL304.4	Apply the concept of inheritance as method overriding and interfaces for multiple inheritance.	Apply	Apply (level 3)
CSL304.5	Apply the concept of exception handling using try, catch, finally, throw and throws and multithreading for thread management.	Apply	Apply (level 3)
CSL304.6	Develop GUI based application using applets and AWT Controls.	Develop	Create (level 6)

Mapping of Experiments with Course Outcomes

List of Experiments	Course Outcomes					
	CSL304.1	CSL304.2	CSL304.3	CSL304.4	CSL304.5	CSL304.6
Implement a program using Basic programming constructs like branching and looping	3	-	-	-	-	-
Implement a program to accept the input from user using Scanner and Buffered Reader.	3	-	-	-	-	-
Implement a program that demonstrates the concepts of class and objects	-	3	-	-	-	-
Implement a program on method and constructor overloading.	-	3	-	-	-	-
Implement a program on Packages.	-	-	3	-	-	-
Implement a program on 2D array & strings functions.	-	-	3	-	-	-
Implement a program on single inheritance.	-	-	-	3	-	-
Implement a program on Multiple Inheritance with Interface.	-	-	-	3	-	-
Implement a program on Exception handling.	-	-	-	-	3	-

Implement a program on Multithreading.	-	-	-	-	3	-
Implement a program on Applet or AWT Controls.	-	-	-	-	-	3
Mini Project based on the content of the syllabus (Group of 2-3 students)	-	-	-	-	-	3

INDEX

Sr. No.	Name of Experiment	D.O.P.	D.O.C.	Page No.	Remark
1	Implement a program using Basic programming constructs like branching and looping				
2	Implement a program to accept the input from user using Scanner and Buffered Reader.				
3	Implement a program that demonstrates the concepts of class and objects				
4	Implement a program on method and constructor overloading.				
5	Implement a program on Packages.				
6	Implement a program on 2D array & strings functions.				
7	Implement a program on single inheritance.				
8	Implement a program on Multiple Inheritance with Interface.				
9	Implement a program on Exception Handling.				
10	Implement a program on Multithreading.				
11	Implement a program on Applet or AWT Controls				
12	Mini Project based on the content of the syllabus (Group of 2-3 students)				

D.O.P: Date of performance

D.O.C : Date of correction

Experiment No.1
Basic programming constructs like branching and looping
Date of Performance:
Date of Submission:

Aim :- To apply programming constructs of decision making and looping.

Objective :- To apply basic programming constructs like Branching and Looping for solving arithmetic problems like calculating factorial of a no entered by user at command prompt .

Theory :-

Programming constructs are basic building blocks that can be used to control computer programs. Most programs are built out of a fairly standard set of programming constructs. For example, to write a useful program, we need to be able to store values in variables, test these values against a condition, or loop through a set of instructions a certain number of times. Some of the basic program constructs include decision making and looping.

Decision Making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled. A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

- if
- if-else
- nested-if
- if-else-if
- switch-case
- break, continue

These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

A loop is a programming structure that repeats a sequence of instructions until a specific condition is met. Programmers use loops to cycle through values, add sums of numbers, repeat functions, and many other things. ... Two of the most common types of loops are the while loop and the for loop. The different ways of looping in programming languages are

- while
- do-while

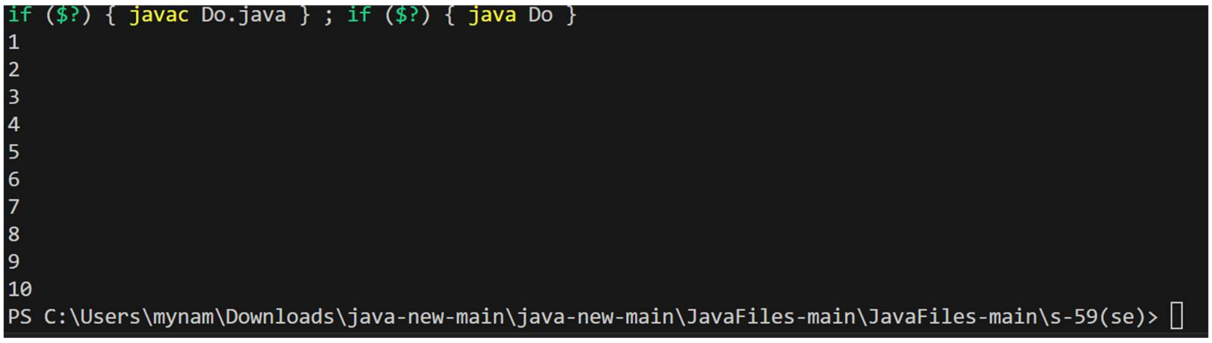
- for loop
- Some languages have modified for loops for more convenience eg :- Modified for loop in java.

For and while loop is entry-controlled loops. Do-while is an exit-controlled loop.

Code: - DO-WHILE

```
class Do {
    public static void main(String[] args) {
        int a = 0;
        do {
            a = a + 1;
            System.out.println(a);
        } while (a < 10);

    }
}
```

OUTPUT:

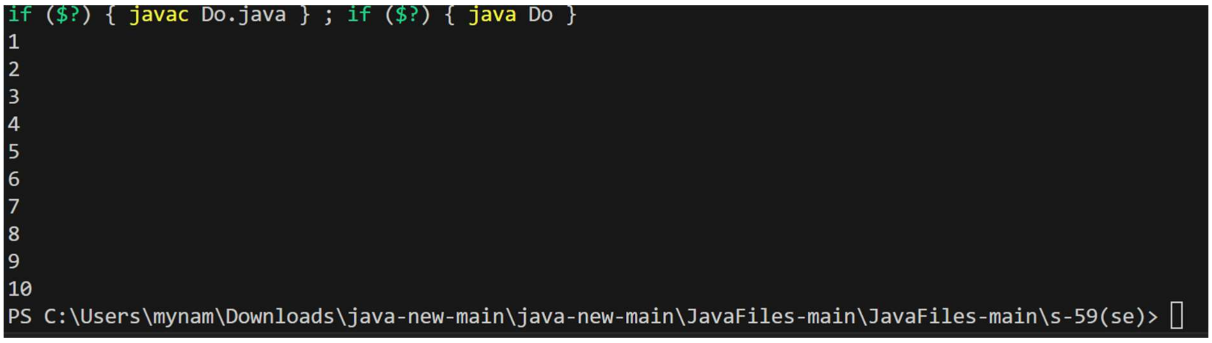


```
if ($?) { javac Do.java } ; if ($?) { java Do }
1
2
3
4
5
6
7
8
9
10
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

FOR

```
class For {
    public static void main(String[] args) {
        int a = 10;
        int i;
        for (i = 1; i <= a; i++) {
            System.out.println(i);
        }
    }
}
```

OUTPUT

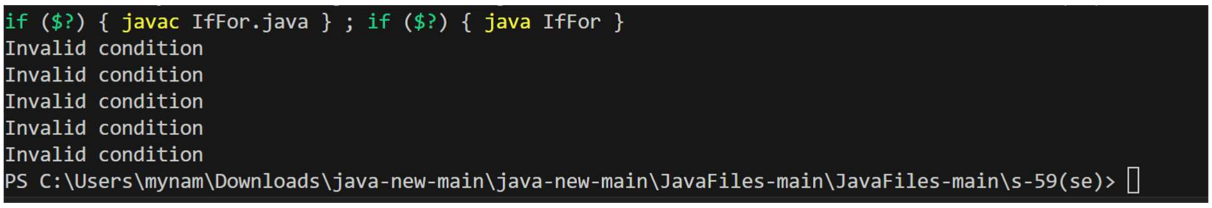


```
if ($?) { javac Do.java } ; if ($?) { java Do }
1
2
3
4
5
6
7
8
9
10
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

IF-FOR

```
class IfFor {
  public static void main(String args[]) {
    int a = 10;
    int b = 20;
    int i;
    for (i = 0; i < 5; i++) {
      if (a == b) {
        System.out.println("This is my first program");
      } else {
        System.out.println("Invalid condition");
      }
    }
  }
}
```

OUTPUT

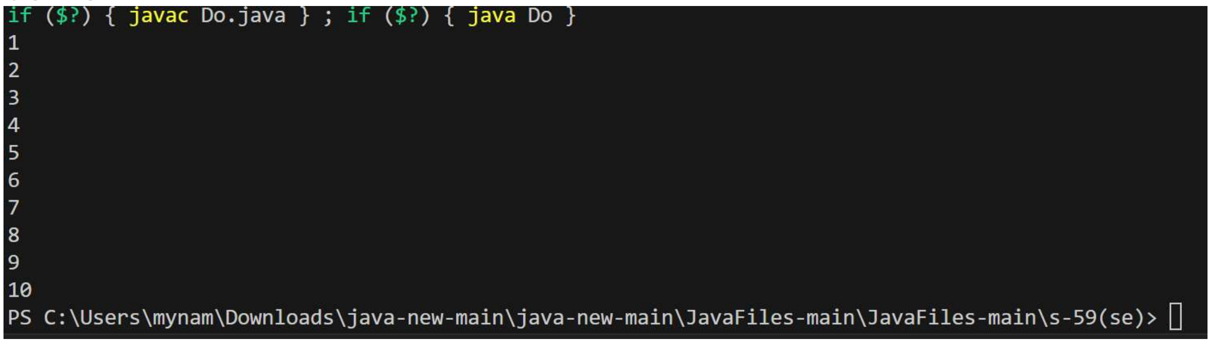


```
if ($?) { javac IfFor.java } ; if ($?) { java IfFor }
Invalid condition
Invalid condition
Invalid condition
Invalid condition
Invalid condition
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

WHILE

```
class While

{
    public static void main(String[] args) {
        int a = 1;
        while (a < 11) {
            System.out.println(a);
            a++;
        }
    }
}
```

OUTPUT

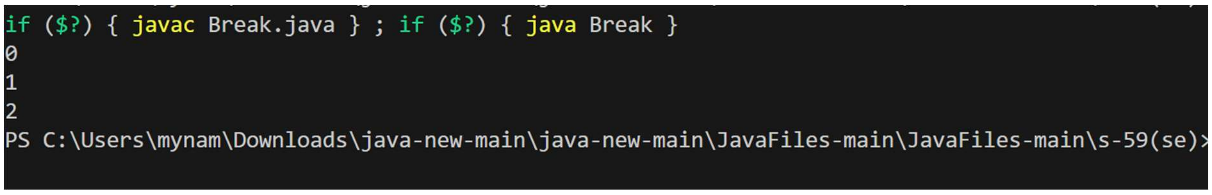


```
if ($?) { javac Do.java } ; if ($?) { java Do }
1
2
3
4
5
6
7
8
9
10
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

BREAK

```
class Break {
  public static void main(String args[]) {
    int i;
    for (i = 0; i < 5; i++) {
      if (i == 3)
        break;
      System.out.println(i);
    }
  }
}
```

OUTPUT

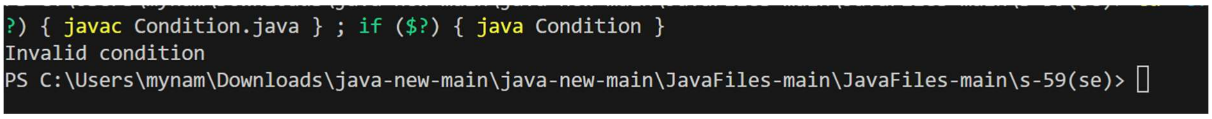


```
if ($?) { javac Break.java } ; if ($?) { java Break }
0
1
2
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

IF-ELSE

```
class Condition {
  public static void main(String args[]) {
    int a = 10;
    int b = 20;
    if (a == b) {
      System.out.println("This is my first program");
    } else {
      System.out.println("Invalid condition");
    }
  }
}
```

OUTPUT

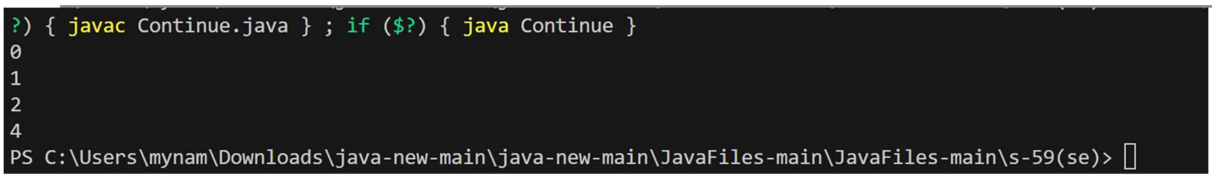


```
?) { javac Condition.java } ; if ($?) { java Condition }
Invalid condition
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

CONTINUE

```
class Continue {
  public static void main(String args[]) {
    int i;
    for (i = 0; i < 5; i++) {
      if (i == 3)
        continue;
      System.out.println(i);
    }
  }
}
```

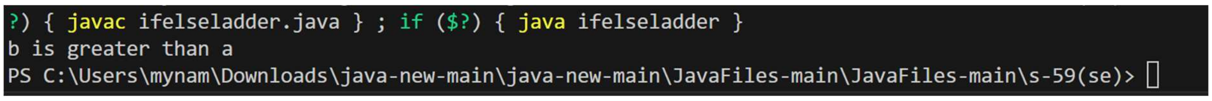
OUTPUT


```
?) { javac Continue.java } ; if ($?) { java Continue }
0
1
2
4
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

IF-ELSE LADDER

```
class ifelseladder {
  public static void main(String args[]) {
    int a = 10;
    int b = 20;

    if (a == b) {
      System.out.println("a is equal to b");
    } else if (a > b) {
      System.out.println("A is greater then b");
    } else {
      System.out.println("b is greater than a");
    }

  }

}
```

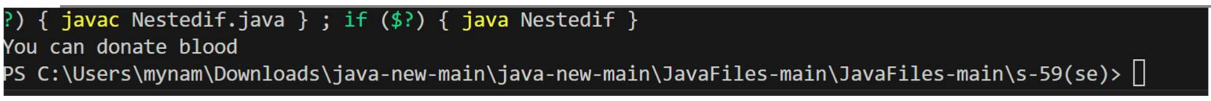
OUTPUT


```
?) { javac ifelseladder.java } ; if ($?) { java ifelseladder }
b is greater than a
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

NESTED-IF

```
class Nestedif {
    public static void main(String[] args) {
        int age = 21;
        int weight = 75;

        if (age >= 18) {
            if (weight > 50) {
                System.out.println("You can donate blood");
            }
        }
        else
          System.out.println("You cannot donate blood");

    }
}
```

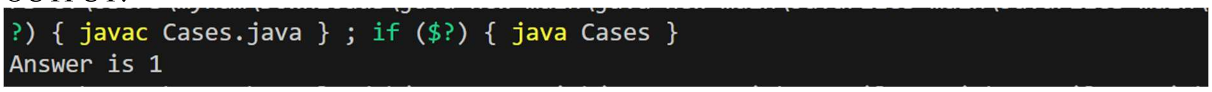
OUTPUT


```
?) { javac Nestedif.java } ; if ($?) { java Nestedif }
You can donate blood
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

SWITCH

```
class Cases {
    public static void main(String[] args) {
        int a = 2;
        int b = 3;
        int c;
        c = b - a;

        switch (c) {
            case 1:
                System.out.println("Answer is 1");
                break;

            case 2:
                System.out.println("Answer is 2");
                break;

            case 3:
                System.out.println("Answer is 3");
        }
    }
}
```

OUTPUT:


```
?) { javac Cases.java } ; if ($?) { java Cases }
Answer is 1
```

Conclusion:

Comment on how branching and looping useful in solving problems.

Branching (if-else and switch statements): Branching is pivotal for decision-making within a program. It enables the execution of different blocks of code based on certain conditions. In Java, the if-else and switch statements are commonly used for branching. They help in handling various scenarios, such as handling user inputs, managing exceptions, and implementing logic based on different conditions.

if-else: It allows the program to execute different blocks of code based on a condition's evaluation. This is crucial for creating adaptive code that responds differently to different inputs or situations.

switch: The switch statement provides an efficient way to select from many alternatives. It is particularly useful when dealing with a large number of potential execution paths based on the value of a single variable.

Looping (for, while, and do-while loops): Loops are vital for executing a block of code repeatedly. They allow programmers to automate repetitive tasks, process collections of data, and perform iterative operations. In Java, the commonly used loops are:

for loop: It is used when the number of iterations is known beforehand. It provides a concise way to write the loop's initialization, condition, and increment/decrement in a single line.

while loop: This loop continues to execute a block of code as long as a specified condition is true. It is particularly useful when the number of iterations is not predetermined.

do-while loop: This loop is similar to the while loop, but it guarantees that the block of code will be executed at least once, even if the condition is initially false.

Experiment No.2
Accepting Input Through Keyboard
Date of Performance:
Date of Submission:

Aim: To apply basic programing for accepting input through keyboard.

Objective: To use the facility of java to read data from the keyboard for any program

Theory:

Java brings various Streams with its I/O package that helps the user perform all the Java input-output operations. These streams support all types of objects, data types, characters, files, etc. to fully execute the I/O operations. Input in Java can be with certain methods mentioned below in the article.

Methods to Take Input in Java

There are two ways by which we can take Java input from the user or from a file

1. BufferedReader Class
2. Scanner Class

Using BufferedReader Class for String Input In Java

It is a simple class that is used to read a sequence of characters. It has a simple function that reads a character another read which reads, an array of characters, and a readLine() function which reads a line.

InputStreamReader() is a function that converts the input stream of bytes into a stream of characters so that it can be read as BufferedReader expects a stream of characters. BufferedReader can throw checked Exceptions.

Using Scanner Class for Taking Input in Java

It is an advanced version of BufferedReader which was added in later versions of Java. The scanner can read formatted input. It has different functions for different types of data types.

The scanner is much easier to read as we don't have to write throws as there is no exception thrown by it.

It was added in later versions of Java

It contains predefined functions to read an Integer, Character, and other data types as well.

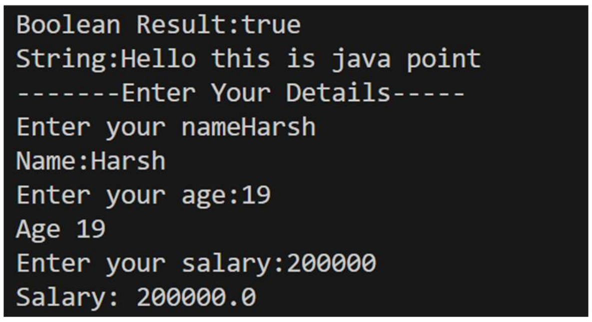
Syntax of Scanner class

Scanner scn = new Scanner(System.in);

Code:

```
import java.util.*;

public class ScannerClassExample1 {
   public s
tatic void main(String args[]) {
      String s = "Hello";
      Scanner scan = new Scanner(s);
      System.out.println("Boolean Result:" + scan.hasNext());
      System.out.println("String:" + scan.nextLine());
      System.out.println("-------Enter Your Details-----");
      Scanner in = new Scanner(System.in);
      System.out.print("Enter your name");
      String name = in.next();
      System.out.println("Name:" + name);
      System.out.print("Enter your age:");
      int i = in.nextInt();
      System.out.println("Age " + i);
      System.out.print("Enter your salary:");
      double d = in.nextDouble();
      System.out.println("Salary: " + d);

   }
}
```

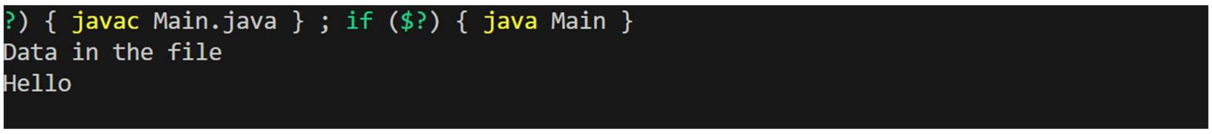
OUTPUT:



```
Boolean Result:true
String:Hello this is java point
-------Enter Your Details-----
Enter your nameHarsh
Name:Harsh
Enter your age:19
Age 19
Enter your salary:200000
Salary: 200000.0
```

```
import java.io.FileReader;
import java.io.BufferedReader;

class Main {
  public static void main(String[] args) {
    char[] array = new char[100];

    try {

      FileReader file = new FileReader("input.txt");

      BufferedReader input = new BufferedReader(file);

      input.read(array);
      System.out.println("Data in the file");
      System.out.println(array);

      input.close();
    }

    catch (Exception e) {
      e.getStackTrace();
    }

  }

}
```

OUTPUT:



```
?) { javac Main.java } ; if ($?) { java Main }
Data in the file
Hello
```

Conclusion:

Comment on how you have used BufferedReader and Scanner Class for accepting user input

Both BufferedReader and Scanner have their own use cases. While BufferedReader is more efficient for reading large amounts of data, Scanner is more convenient for simple input parsing. It's important to handle exceptions properly and close these resources after usage to prevent resource leaks. Choose the appropriate class based on the requirements and complexity of the input reading tasks in your Java application.

Experiment No. 3
Implement a program that demonstrates the concepts of class and objects
Date of Performance:
Date of Submission:

Aim: Implement a program that demonstrates the concepts of class and objects

Objective: To develop the ability of converting real time entity into objects and create their classes.

Theory:

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties i.e., members and methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. Modifiers: A class can be public or has default access.
2. class keyword: class keyword is used to create a class.
3. Class name: The name should begin with a initial letter (capitalized by convention).
4. Superclass (if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. Body: The class body surrounded by braces, {}.

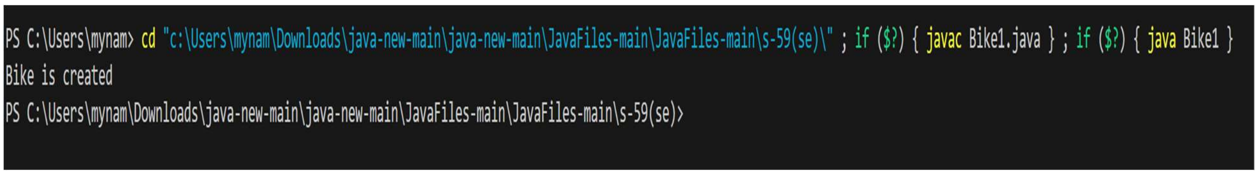
An OBJECT is a basic unit of Object-Oriented Programming and represents the real-life entities. A typical Java program creates many objects, which interact by invoking methods. An object consists of:

1. State: It is represented by attributes of an object. It also reflects the properties of an object.
2. Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. Identity: It gives a unique name to an object and enables one object to interact with other objects.

Code:

```
class Bike1 {

   Bike1()

   {

      System.out.println("Bike is created");

   }

   public static void main(String args[]) {

      Bike1 b = new Bike1();

   }

}
```

OUTPUT:



```
PS C:\Users\mynam> cd "c:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)\" ; if ($?) { javac Bike1.java } ; if ($?) { java Bike1 }
Bike is created
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
```

Conclusion:

Comment on how you create a class template and their objects.

In Java, you can create a class template, also known as a class blueprint, using the class keyword. A class template serves as a model for objects that you create based on its structure. Below is an example of how you can create a simple class template in Java:

```
public class MyClassTemplate {
   // Fields or data members
   private int myField;

   // Constructor
   public MyClassTemplate(int myField) {
      this.myField = myField;
   }

   // Methods
   public void setMyField(int myField) {
        this.myField = myField;
    }

    public int getMyField() {
        return myField;
    }
}
```

Here's a brief explanation of the components used in the above example:

public class MyClassTemplate: This line declares a class named MyClassTemplate.

private int myField;: This line declares a private integer variable myField, which is a field of the class.

public MyClassTemplate(int myField): This is the constructor, which initializes the myField variable.

public void setMyField(int myField): This is a setter method that allows you to modify the value of the myField variable.

public int getMyField(): This is a getter method that returns the value of the myField variable.

Experiment No. 4
Implement a program on method and constructor overloading.
Date of Performance:
Date of Submission:

Aim: Implement a program on method and constructor overloading.

Objective: To use concept of method overloading in a java program to create a class with same function name with different number of parameters.

Theory:
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 argument lists.

Example: This example to show how method overloading is done by having different number of parameters for the same method name.

```
Class DisplayOverloading
{
   public void disp(char c)
   {
       System.out.println(c);
   }
   public void disp(char c, int num)
   {
       System.out.println(c + " "+num);
   }
}
Class Sample
{
  Public static void main(String args[])
  {
     DisplayOverloading obj = new DisplayOverloading();
     Obj.disp('a');
     Obj.disp('a',10);
  }
}
```

Output:

A

A 10

Java supports Constructor Overloading in addition to overloading methods. In Java, overloaded constructor is called based on the parameters specified when a new is executed.

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading.

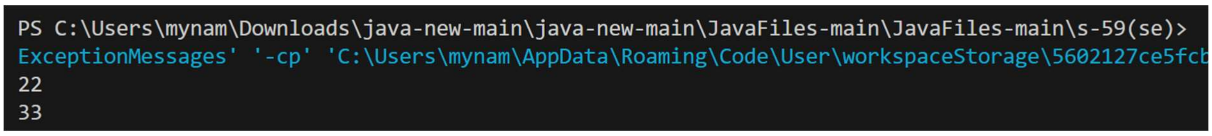
For example, the Thread class has 8 types of constructors. If we do not want to specify anything about a thread then we can simply use the default constructor of the Thread class, however, if we need to specify the thread name, then we may call the parameterized constructor of the Thread class with a String args like this:

Thread t= new Thread (" MyThread ");

Code:

```
class Testoverloading1 {
   public static void main(String[] args) {
      System.out.println(Adder.add(11, 11));
      System.out.println(Adder.add(11, 11, 11));
   }
}

class Adder {
   static int add(int a, int b) {
      return a + b;
   }

   static int add(int a, int b, int c) {
      return a + b + c;
   }
}
```

OUTPUT:



```
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
ExceptionMessages' '-cp' 'C:\Users\mynam\AppData\Roaming\Code\User\workspaceStorage\5602127ce5fcb
22
33
```

Conclusion:

Comment on how function and constructor overloading used using java

Function and constructor overloading in Java are used to create multiple methods or constructors with the same name but different parameters. This allows developers to create more flexible and intuitive code that can perform similar operations on different types of data or with different combinations of parameters.

Function overloading enables the creation of multiple functions with the same name but different parameter lists. Java determines which function to call based on the number and types of parameters provided

Constructor overloading allows the creation of multiple constructors within a class, each with a different parameter list. This enables the creation of objects in different ways, depending on the provided parameters.

Experiment No. 5
Implement a program on Packages.
Date of Performance:
Date of Submission:

Aim: To use packages in java.

Objective: To use packages in java to use readymade classes available in them using square root method in math class.

Theory:

A java package is a group of similar types of classes, interfaces and sub-packages. Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

There are two types of packages-

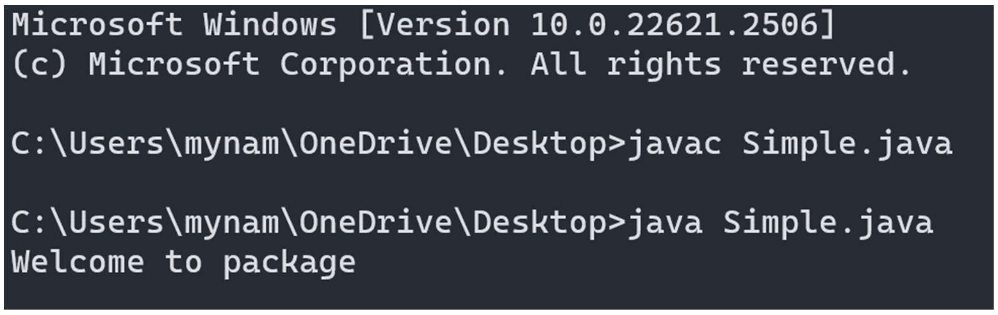
1. Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.
2. User defined package: The package we create for is called user-defined package.

Programmers can define their own packages to bundle group of classes/interfaces, etc. While creating a package, the user should choose a name for the package and include a package statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package. If a package statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.

Code:

```
package mypack;
public class Simple{
public static void main(String args[])
{
System.out.println("Welcome to package");
}
 }
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22621.2506]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mynam\OneDrive\Desktop>javac Simple.java

C:\Users\mynam\OneDrive\Desktop>java Simple.java
Welcome to package
```

Conclusion:

Comment on the autoencoder architecture and the Image compression results.

Autoencoders are a type of artificial neural network used for learning efficient representations of data, typically for dimensionality reduction, data denoising, or compression. The basic architecture of an autoencoder consists of an encoder and a decoder. The encoder compresses the input data into a lower-dimensional representation, and the decoder reconstructs the original input from this representation. Encoder: The encoder network maps the input data to a latent space representation, which is typically of lower dimensionality than the input. This compressed representation retains the most important features of the input data.

Decoder: The decoder network reconstructs the original input from the latent space representation generated by the encoder. It aims to produce an output that is as close as possible to the original input.

Experiment No. 6
Implement a program on 2D array & strings functions.
Date of Performance:
Date of Submission:

Aim: To use 2D arrays and Strings for solving given problem.

Objective: To use 2D array concept and strings in java to solve real world problem

Theory:

- An array is used to store a fixed-size sequential collection of data of the same type.
- An array can be init in two ways:
  1. Initializing at the time of declaration:
     dataType[] myArray = {value0, value1, ..., valuek};
  2. Dynamic declaration:
     dataType[] myArray = new dataType[arraySize];
     myArray[index] = value;
- Two – dimensional array is the simplest form of a multidimensional array. Data of only same data type can be stored in a 2D array.Data in a 2D Array is stored in a tabular manner which can be represented as a matrix.
- A 2D Array can be declared in 2 ways:
  1. Intializing at the time of declaration:
     dataType[][] myArray = { {valueR1C1, valueR1C2...}, {valueR2C1, valueR2C2...},..}
  2. Dynamic declaration:
     dataType[][] myArray = new dataType[x][y];
     myArray[row_index][column_index] = value;

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

1.String literal

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

Example:

String demoString = “GeeksforGeeks”;

2. Using new keyword

- String s = new String(“Welcome”);
- In such a case, JVM will create a new string object in normal (non-pool) heap memory and the literal “Welcome” will be placed in the string constant pool. The variable s will refer to the object in the heap (non-pool)

Example:

String demoString = new String (“GeeksforGeeks”);

Code:

```
class Multidimensional {

    public static void main(String args[]) {

        int arr[][] = { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                System.out.print(arr[i][j] + " ");

            System.out.println();
        }
    }
}
```

Conclusion:

Comment on how you have used the concept of string and 2D array.

Certainly! The concept of strings and 2D arrays is commonly used in various programming scenarios, including data manipulation, text processing, and algorithmic problem-solving.
In this example, the concept of strings is demonstrated by creating a string str and performing operations such as extracting substrings and converting the string to a character array. Additionally, the concept of a 2D array is

illustrated by creating a 2D array matrix, displaying its original values, and then modifying one of its elements.

Experiment No. 7
Implement a program on single inheritance.
Date of Performance:
Date of Submission:

Aim: To implement the concept of single inheritance.

Objective: Ability to design a base and child class relationship to increase reusability.

Theory:
Single inheritance can be defined as a derived class to inherit the basic methods (data members and variables) and behaviour from a superclass. It's a basic is-a relationship concept exists here. Basically, java only uses a single inheritance as a subclass cannot extend more superclass.
Inheritance is the basic properties of object-oriented programming. Inheritance tends to make use of the properties of a class object into another object. Java uses inheritance for the purpose of code-reusability to reduce time by then enhancing reliability and to achieve run time polymorphism. As the codes are reused it makes less development cost and maintenance. Java has different types of inheritance namely single inheritance, multilevel, multiple, hybrid. In this article, we shall go through on basic understanding of single inheritance concept briefly in java with a programming example. Here we shall have a complete implementation in java.

Syntax:

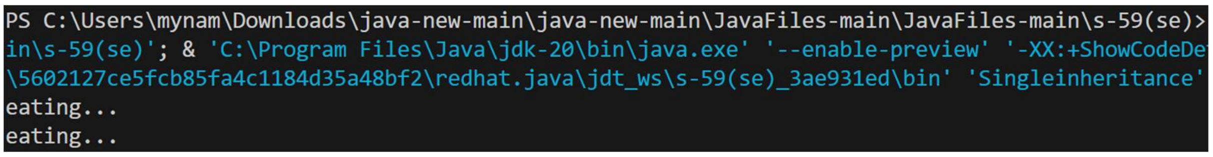
The general syntax for this is given below. The inheritance concepts use the keyword 'extend' to inherit a specific class. Here you will learn how to make use of extending keyword to derive a class. An extend keyword is declared after the class name followed by another class name. Syntax is,

```
class base class
{…. methods
}
class derived class name extends base class
{
methods … along with this additional feature
}
```

Java uses a keyword 'extends' to make a new class that is derived from the existing class. The inherited class is termed as a base class or superclass, and the newly created class is called derived or subclass. The class which gives data members and methods known as the base class and the class which takes the methods is known as child class.

Code:

```
class Singleinheritance {
   public static void main(String args[]) {
      Dog d = new Dog();
      d.bark();
      d.eat();
   }
}

class Animal {
   void eat() {
      System.out.println("eating");
   }
}

class Dog extends Animal {
   void bark() {
      System.out.println("barking");
   }
}
```



```
PS C:\Users\mynam\Downloads\java-new-main\java-new-main\JavaFiles-main\JavaFiles-main\s-59(se)>
in\s-59(se)'; & 'C:\Program Files\Java\jdk-20\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDe
\5602127ce5fcb85fa4c1184d35a48bf2\redhat.java\jdt_ws\s-59(se)_3ae931ed\bin' 'Singleinheritance'
eating...
eating...
```

Conclusion:

Comment on the Single inheritance.

In Java, single inheritance refers to the concept where a class can inherit properties and behavior from only one superclass. This means that a Java class can have only one direct superclass, also known as a single parent class. Single inheritance is a fundamental aspect of Java's object-oriented programming model, and it helps in maintaining a simple and straightforward class hierarchy. Here are some key points regarding single inheritance in Java:

Class Hierarchy: With single inheritance, a class can have only one immediate superclass. However, this superclass can have its own superclass, creating a hierarchical tree-like structure.

Superclass-Subclass Relationship: Single inheritance establishes a relationship where a subclass inherits the properties and behavior of its immediate superclass. This allows the subclass to extend the functionality of the superclass while maintaining a hierarchical relationship.

Java Interface Implementation: Java supports multiple interface implementation, allowing a class to implement multiple interfaces. This feature helps in achieving a form of multiple inheritance through interfaces, where a class can inherit the capabilities of multiple interfaces.

Experiment No. 8
Implement a program on multiple inheritance with interface.
Date of Performance:
Date of Submission:

Aim: Implement a program on multiple inheritance with interface.

Objective: Implement multiple inheritance in a program to perform addition, multiplication and transpose operations on a matrix. Create an interface to hold prototypes of these methods and create a class input to read input. Inherit a new class from this interface and class. In main class create object of this child class and invoke required methods.

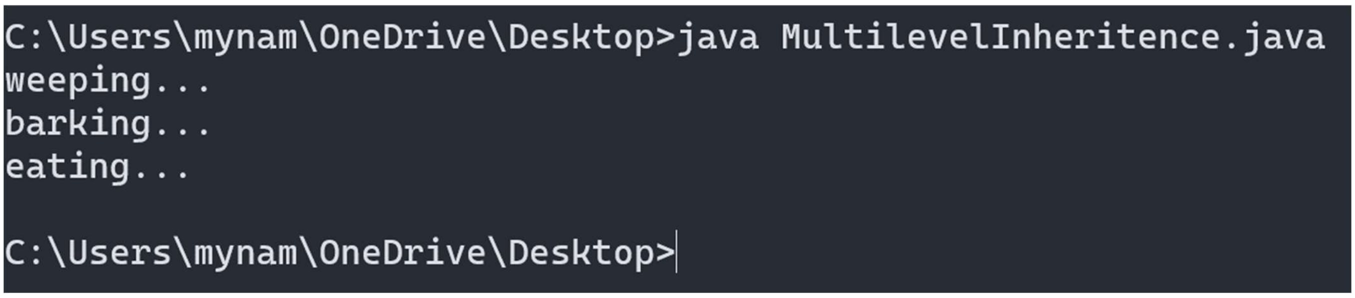
Theory:

- In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces.
- An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class. If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.
- However, Java supports multiple interface inheritance where an interface extends more than one super interfaces.
- A class implements an interface, but one interface extends another interface. Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.
- The following is the syntax used to extend multiple interfaces in Java:

```
access_specifier interface subinterfaceName extends superinterface1, superinterface2, …… {
// Body
}
```

Code:

```
class MutilevelInheritance
{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}
}
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
```



```
C:\Users\mynam\OneDrive\Desktop>java MultilevelInheritence.java
weeping...
barking...
eating...

C:\Users\mynam\OneDrive\Desktop>
```

Conclusion:

Comment on how interface are useful and implemented using java.

Interfaces in Java are essential for achieving abstraction and providing a way to achieve full abstraction in Java. They define a contract that specifies the capabilities a class must implement. Here are a few reasons why interfaces are useful and how they can be implemented in Java:

Abstraction: Interfaces allow you to create code that is more abstract and flexible. By defining a set of methods without specifying the implementation details, interfaces enable different classes to provide their own implementation. This promotes loose coupling between classes and allows for more modular and maintainable code.

Multiple Inheritance: Java does not support multiple inheritance of classes, but it does support the implementation of multiple interfaces. This means that a class can implement multiple interfaces, allowing it to inherit behaviors from multiple sources. This helps in achieving a higher level of flexibility and code reuse.

Standardization: Interfaces are often used to define a standard for a set of related classes. By implementing the same interface, different classes can adhere to the same contract, ensuring that they all provide the same set of functionalities. This makes it easier to work with different implementations interchangeably.

Experiment No. 9
Exception handling
Date of Performance:
Date of Submission:

Aim: Implement a program on Exception handling.

Objective: To able handle exceptions occurred and handle them using appropriate keyword

Theory:

The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.

throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

```
public class JavaExceptionExample{

  public static void main(String args[]){

   try{

      //code that may raise exception

      int data=100/0;



  }catch(ArithmeticException e){System.out.println(e);}

   //rest code of the program

   System.out.println("rest of the code...");

  }

}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Conclusion:

Comment on how exceptions are handled in JAVA.

Try-Catch Blocks: Java uses the try, catch, and finally blocks to handle exceptions. The try block contains the code that might throw an exception, and the catch block catches and handles the exception if it occurs. The finally block, if used, is executed regardless of whether an exception is thrown, and is often used for cleanup tasks.

Experiment No. 10
Implement program on Multithreading
Date of Performance:
Date of Submission:

Aim: Implement program on Multithreading

Objective:

Theory:

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:
Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

1) Java Thread Example by extending Thread class
FileName: Multi.java

```
class Multi extends Thread{
public void run(){
```

```
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output:

```
thread is running...
```

2) Java Thread Example by implementing Runnable interface
FileName: Multi3.java

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

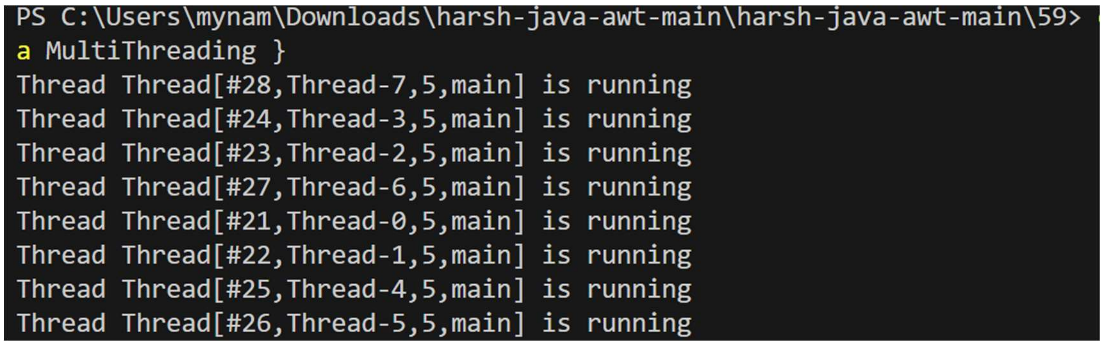
Output:

```
thread is running...
```

Code:

```
class MultithreadingDemo extends Thread {
        public void run()
        {
                try {

                        System.out.println(
                                "Thread " + Thread.currentThread()
                                + " is running");
                }
                catch (Exception e) {
                        // Throwing an exception
                        System.out.println("Exception is caught");
                }
        }
}


public class MultiThreading {
        public static void main(String[] args)
        {
                int n = 8; // Number of threads
                for (int i = 0; i < n; i++) {
                        MultithreadingDemo object
                                = new MultithreadingDemo();
                        object.start();
                }
        }
}
```

OUTPUT:



```
PS C:\Users\mynam\Downloads\harsh-java-awt-main\harsh-java-awt-main\59>
a MultiThreading }
Thread Thread[#28,Thread-7,5,main] is running
Thread Thread[#24,Thread-3,5,main] is running
Thread Thread[#23,Thread-2,5,main] is running
Thread Thread[#27,Thread-6,5,main] is running
Thread Thread[#21,Thread-0,5,main] is running
Thread Thread[#22,Thread-1,5,main] is running
Thread Thread[#25,Thread-4,5,main] is running
Thread Thread[#26,Thread-5,5,main] is running
```

Conclusion:

Comment on how multithreading is supported in JAVA.

Java provides built-in support for multithreading, allowing developers to create applications that can perform multiple tasks concurrently. Multithreading in Java is primarily achieved using the java.lang.Thread class and the java.lang.Runnable interface. Here are some key features and components of multithreading in Java:

Thread Class and Runnable Interface:
The Thread class in Java represents a thread of execution. Threads can be created by either extending the Thread class or implementing the Runnable interface. The Runnable interface is preferred when creating threads, as it allows for better separation of concerns.

Experiment No. 11
Implement a program on Applet or AWT Controls
Date of Performance:
Date of Submission:

Aim: Implement a program on Applet or AWT Controls

Objective:

To develop application like Calculator, Games, Animation using AWT Controls.

Theory:

Java AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

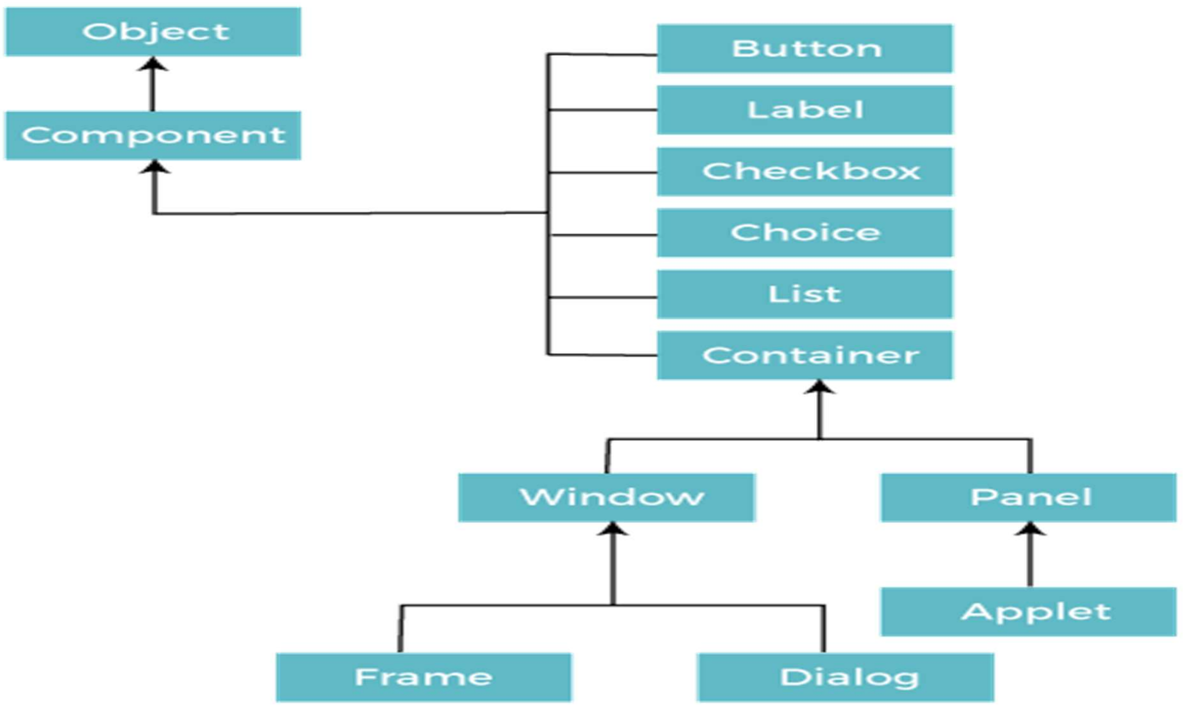
1. A general interface between Java and the native system, used for windowing, events and layout managers. This API is at the core of Java GUI programming and is also used by Swing and Java 2D. It contains the interface between the native windowing system and the Java application1.
2. A basic set of GUI widgets such as buttons, text boxes, and menus1. AWT also provides Graphics and imaging tools, such as shape, color, and font classes2. AWT also avails layout managers which helps in increasing the flexibility of the window layouts2

Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

Java AWT Hierarchy

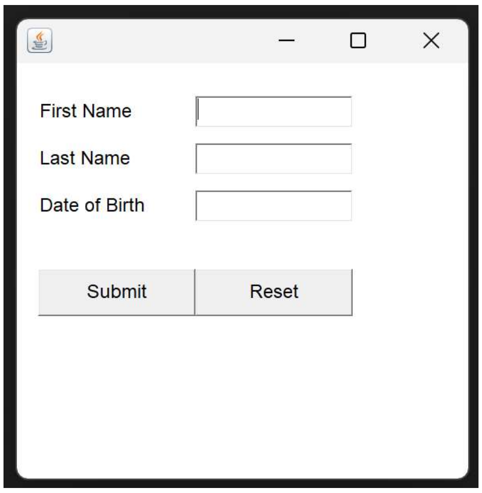


Code:

```
import java.awt.*;
public class AwtApp extends Frame {
AwtApp(){
Label firstName = new Label("First Name");
firstName.setBounds(20, 50, 80, 20);

Label lastName = new Label("Last Name");
```

```
lastName.setBounds(20, 80, 80, 20);

Label dob = new Label("Date of Birth");
dob.setBounds(20, 110, 80, 20);

TextField firstNameTF = new TextField();
firstNameTF.setBounds(120, 50, 100, 20);

TextField lastNameTF = new TextField();
lastNameTF.setBounds(120, 80, 100, 20);

TextField dobTF = new TextField();
dobTF.setBounds(120, 110, 100, 20);

Button sbmt = new Button("Submit");
sbmt.setBounds(20, 160, 100, 30);

Button reset = new Button("Reset");
reset.setBounds(120,160,100,30);

add(firstName);
add(lastName);
add(dob);
add(firstNameTF);
add(lastNameTF);
add(dobTF);
add(sbmt);
add(reset);

setSize(300,300);
setLayout(null);
```

```
setVisible(true);
}
public static void main(String[] args) {
AwtApp awt = new AwtApp();

}
}
```



Conclusion:

Comment on application development using AWT Controls.

AWT (Abstract Window Toolkit) is a set of application programming interfaces (APIs) used for creating graphical user interfaces (GUIs) in Java. AWT provides a collection of classes and methods for building and managing components such as windows, buttons, text fields, and other graphical elements. AWT Controls are the various GUI components provided by the AWT library for creating interactive user interfaces. Here's an overview of some commonly used AWT Controls:

Frame: AWT's Frame class is used to create the main application window. It serves as a container for other AWT components.

Button: The Button class represents a push button that can trigger an action when clicked.

Label: The Label class is used to display a single line of read-only text. It is often used to provide descriptions or captions for other components.

TextField: The TextField class provides a single-line input field for the user to enter text.

TextArea: The TextArea class is used to create a multi-line area for editing text.

Checkbox: The Checkbox class represents a check box that can be selected or deselected by the user.

Choice: The Choice class is used to create a drop-down list that allows the user to select one option from a list of available choices.

List: The List class is used to create a component that displays a list of items from which the user can make multiple selections.

Experiment No. 12
Course Project based on the content of the syllabus.
Date of Performance:
Date of Submission:

```
import java.awt.Dimension;
import java.awt.Color;
import java.awt.event.*;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Font;
import java.awt.Toolkit;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.ImageIcon;
import java.io.*;
import java.util.Scanner;

public class TicTacToe extends JPanel implements ActionListener {

   // core logic variables
   boolean playerX; // true if player X's turn, false if player O's turn
   boolean gameDone = false; // true if game is over
   int winner = -1; // 0 if X wins, 1 if O wins, -1 if no winner yet
   int player1wins = 0, player2wins = 0; // number of wins for each player
```

```
int[][] board = new int[3][3]; // 0 if empty, 1 if X, 2 if O

// paint variables
int lineWidth = 5; // width of the lines
int lineLength = 270; // length of the lines
int x = 15, y = 100; // location of first line
int offset = 95; // square width
int a = 0; // used for drawing the X's and O's
int b = 5; // used for drawing the X's and O's
int selX = 0; // selected square x
int selY = 0; // selected square y

// COLORS
Color turtle = new Color(152, 109, 142);
Color orange = new Color(255, 165, 0);
Color offwhite = new Color(0xf7f7f7);
Color darkgray = new Color(239, 227, 208);
Color pink = new Color(130, 92, 121);

// COMPONENTS
JButton jButton;

// CONSTRUCTOR
public TicTacToe() {
   Dimension size = new Dimension(420, 300); // size of the panel
   setPreferredSize(size);
   setMaximumSize(size);
   setMinimumSize(size);
   addMouseListener(new XOListener()); // add mouse listener
   jButton = new JButton("New Game");
   jButton.addActionListener(this); // add action listener
   jButton.setBounds(315, 210, 100, 30); // set button location
   add(jButton); // add button to panel
   resetGame();
}

public void resetGame() {
   playerX = true;
   winner = -1;
   gameDone = false;
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         board[i][j] = 0; // all spots are empty
      }
   }
   getJButton().setVisible(false); // hide the button
}

public void paintComponent(Graphics page) {
   super.paintComponent(page);
   drawBoard(page);
```

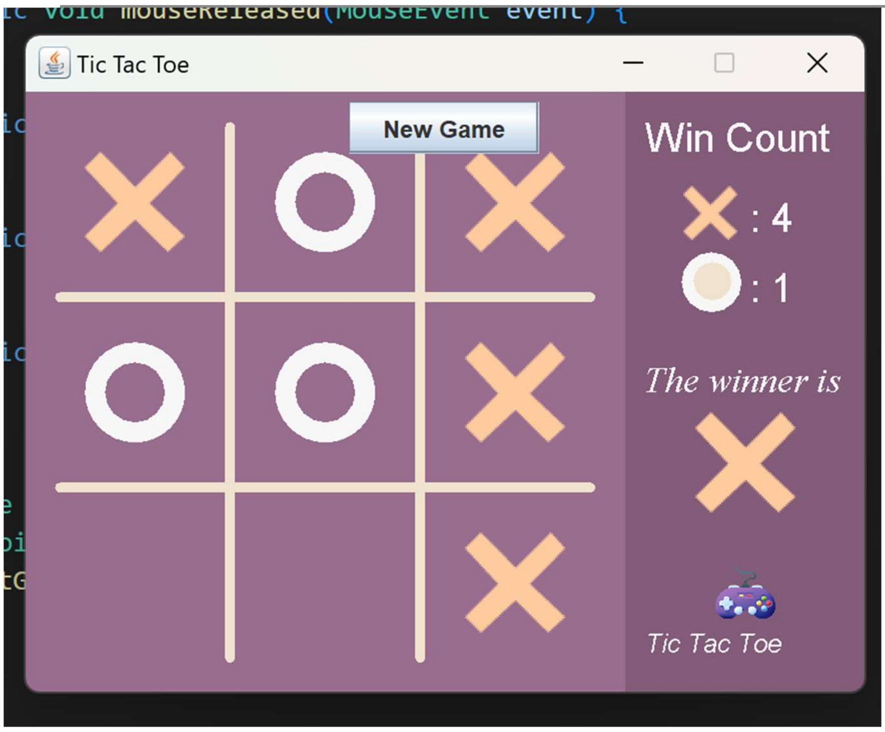
```
    drawUI(page);
    drawGame(page);
  }

  public void drawBoard(Graphics page) {
    setBackground(turtle);
    page.setColor(darkgray);
    page.fillRoundRect(x, y, lineLength, lineWidth, 5, 30);
    page.fillRoundRect(x, y + offset, lineLength, lineWidth, 5, 30);
    page.fillRoundRect(y, x, lineWidth, lineLength, 30, 5);
    page.fillRoundRect(y + offset, x, lineWidth, lineLength, 30, 5);
  }

  public void drawUI(Graphics page) {
    // SET COLOR AND FONT
    page.setColor(pink);
    page.fillRect(300, 0, 120, 300);
    Font font = new Font("Helvetica", Font.PLAIN, 20);
    page.setFont(font);

    // SET WIN COUNTER
    page.setColor(offwhite);
    page.drawString("Win Count", 310, 30);
    page.drawString(": " + player1wins, 362, 70);
    page.drawString(": " + player2wins, 362, 105);

    // DRAW score X
    ImageIcon xIcon = new ImageIcon("orangex.png");
    Image xImg = xIcon.getImage();
    Image newXImg = xImg.getScaledInstance(27, 27, java.awt.Image.SCALE_SMOOTH);
    ImageIcon newXIcon = new ImageIcon(newXImg);
    page.drawImage(newXIcon.getImage(), 44 + offset * 1 + 190, 47 + offset * 0, null);

    // DRAW score O
    page.setColor(offwhite);
    page.fillOval(43 + 190 + offset, 80, 30, 30);
    page.setColor(darkgray);
    page.fillOval(49 + 190 + offset, 85, 19, 19);

    // DRAW WHOS TURN or WINNER
    page.setColor(offwhite);
    Font font1 = new Font("Serif", Font.ITALIC, 18);
    page.setFont(font1);

    if (gameDone) {
      if (winner == 1) { // x
        page.drawString("The winner is", 310, 150);
        page.drawImage(xImg, 335, 160, null);
      } else if (winner == 2) { // o
        page.drawString("The winner is", 310, 150);
        page.setColor(offwhite);
```

```
            page.fillOval(332, 160, 50, 50);
            page.setColor(darkgray);
            page.fillOval(342, 170, 30, 30);
        } else if (winner == 3) { // tie
            page.drawString("It's a tie", 330, 178);
        }
    } else {
        Font font2 = new Font("Serif", Font.ITALIC, 20);
        page.setFont(font2);
        page.drawString("", 350, 160);
        if (playerX) {
            page.drawString("X 's Turn", 325, 180);
        } else {
            page.drawString("O 's Turn", 325, 180);
        }
    }
    // DRAW LOGO
    Image cookie = Toolkit.getDefaultToolkit().getImage("logo.png");
    page.drawImage(cookie, 345, 235, 30, 30, this);
    Font c = new Font("Courier", Font.BOLD + Font.CENTER_BASELINE, 13);
    page.setFont(c);
    page.drawString("Tic Tac Toe", 310, 280);
}

public void drawGame(Graphics page) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == 0) {

            } else if (board[i][j] == 1) {
                ImageIcon xIcon = new ImageIcon("orangex.png");
                Image xImg = xIcon.getImage();
                page.drawImage(xImg, 30 + offset * i, 30 + offset * j, null);
            } else if (board[i][j] == 2) {
                page.setColor(offwhite);
                page.fillOval(30 + offset * i, 30 + offset * j, 50, 50);
                page.setColor(turtle);
                page.fillOval(40 + offset * i, 40 + offset * j, 30, 30);
            }
        }
    }
    repaint();
}

public void checkWinner() {
    if (gameDone == true) {
        System.out.print("gameDone");
        return;
    }
    // vertical
    int temp = -1;
```

```
if ((board[0][0] == board[0][1])
        && (board[0][1] == board[0][2])
        && (board[0][0] != 0)) {
    temp = board[0][0];
} else if ((board[1][0] == board[1][1])
        && (board[1][1] == board[1][2])
        && (board[1][0] != 0)) {
    temp = board[1][1];
} else if ((board[2][0] == board[2][1])
        && (board[2][1] == board[2][2])
        && (board[2][0] != 0)) {
    temp = board[2][1];

    // horizontal
} else if ((board[0][0] == board[1][0])
        && (board[1][0] == board[2][0])
        && (board[0][0] != 0)) {
    temp = board[0][0];
} else if ((board[0][1] == board[1][1])
        && (board[1][1] == board[2][1])
        && (board[0][1] != 0)) {
    temp = board[0][1];
} else if ((board[0][2] == board[1][2])
        && (board[1][2] == board[2][2])
        && (board[0][2] != 0)) {
    temp = board[0][2];

    // diagonal
} else if ((board[0][0] == board[1][1])
        && (board[1][1] == board[2][2])
        && (board[0][0] != 0)) {
    temp = board[0][0];
} else if ((board[0][2] == board[1][1])
        && (board[1][1] == board[2][0])
        && (board[0][2] != 0)) {
    temp = board[0][2];
} else {

    // CHECK FOR A TIE
    boolean notDone = false;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == 0) {
                notDone = true;
                break;
            }
        }
    }
    if (notDone == false) {
        temp = 3;
    }
    }
    if (temp > 0) {
      winner = temp;
      if (winner == 1) {
        player1wins++;
        System.out.println("winner is X");
      } else if (winner == 2) {
        player2wins++;
        System.out.println("winner is O");
      } else if (winner == 3) {
        System.out.println("It's a tie");
      }
      gameDone = true;
      getJButton().setVisible(true);
    }
  }

  public JButton getJButton() {
    return jButton;
  }

  public void setPlayerXWins(int a) {
    player1wins = a;
  }

  public void setPlayerOWins(int a) {
    player2wins = a;
  }

  public static void main(String[] args) {
    JFrame frame = new JFrame("Tic Tac Toe");
    frame.getContentPane();

    TicTacToe gamePanel = new TicTacToe();
    frame.add(gamePanel);

    frame.addWindowListener(new WindowAdapter() {
      public void windowOpened(WindowEvent e) {
        try {
          File file = new File("score.txt");
          Scanner sc = new Scanner(file);
          gamePanel.setPlayerXWins(Integer.parseInt(sc.nextLine()));
          gamePanel.setPlayerOWins(Integer.parseInt(sc.nextLine()));
          sc.close();
        } catch (IOException io) {
          // file doesnt exist
          File file = new File("score.txt");
        }
      }

      public void windowClosing(WindowEvent e) {
            try {
                PrintWriter pw = new PrintWriter("score.txt");
                pw.write("");
                pw.write(gamePanel.player1wins + "\n");
                pw.write(gamePanel.player2wins + "\n");
                pw.close();
            } catch (FileNotFoundException e1) {
            }
        }
    });
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setResizable(false);
    frame.pack();
    frame.setVisible(true);
}

private class XOListener implements MouseListener {

    public void mouseClicked(MouseEvent event) {
        selX = -1;
        selY = -1;
        if (gameDone == false) {
            a = event.getX();
            b = event.getY();
            int selX = 0, selY = 0;
            if (a > 12 && a < 99) {
                selX = 0;
            } else if (a > 103 && a < 195) {
                selX = 1;
            } else if (a > 200 && a < 287) {
                selX = 2;
            } else {
                selX = -1;
            }

            if (b > 12 && b < 99) {
                selY = 0;
            } else if (b > 103 && b < 195) {
                selY = 1;
            } else if (b > 200 && b < 287) {
                selY = 2;
            } else {
                selY = -1;
            }
            if (selX != -1 && selY != -1) {

                if (board[selX][selY] == 0) {
                    if (playerX) {
                        board[selX][selY] = 1;
                        playerX = false;
                    } else {
```

```
                board[selX][selY] = 2;
                playerX = true;
              }
              checkWinner();
              System.out.println(" CLICK= x:" + a + ",y: " + b + "; selX,selY: " + selX + "," + selY);

            }
          } else {
            System.out.println("invalid click");
          }
        }
      }

      public void mouseReleased(MouseEvent event) {
      }

      public void mouseEntered(MouseEvent event) {
      }

      public void mouseExited(MouseEvent event) {
      }

      public void mousePressed(MouseEvent event) {
      }
    }

    @Override
    public void actionPerformed(ActionEvent e) {
      resetGame();
    }

}
```

OUTPUT:


Tic Tac Toe
New Game
Win Count
: 4
: 1
The winner is
Tic Tac Toe