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 to engage 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: -

```
import java.util.Scanner;
public class branch_loop {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();
        if (isPrime(number)) {
            System.out.println(number + " is a prime number.");
        } else {
            System.out.println(number + " is not a prime number.");
        }
    }
    private static boolean isPrime(int num) {
        if (num < 2) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX
:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb
6b54b9e23235e398037b07662649\redhat.java\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'branch_loop'
Enter a number: 01
1 is not a prime number.
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

In this simple Java program, we have demonstrated the use of basic programming constructs. The branching construct (if-else) is employed to determine whether a given number is even or odd. The looping construct (for loop) is used to print the message "Hello" a specified number of times.

These constructs are fundamental building blocks in programming, allowing developers to create more complex and versatile programs. By combining branching and looping, we can address various scenarios and efficiently perform repetitive tasks. Understanding and mastering these basic constructs are essential for any programmer as they form the foundation for more advanced programming concepts and problem-solving techniques.

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.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
public class bufferred_scanner {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter input using Scanner: ");
        String scannerInput = scanner.nextLine();
        System.out.println("You entered using Scanner: " + scannerInput);
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            System.out.print("Enter input using BufferedReader: ");
            String bufferedReaderInput = reader.readLine();
            System.out.println("You entered using BufferedReader: " +
bufferedReaderInput);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeD
etailsInExceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07
662649\redhat.java\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'bufferred_scanner'
Enter input using Scanner: DSE
You entered using Scanner: DSE
Enter input using BufferedReader: AIDS
You entered using BufferedReader: AIDS
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

In this program, we demonstrated two different ways to accept user input in Java, using **Scanner** and **BufferedReader**.

- **Scanner:** It is a simple and convenient class for parsing primitive types and strings. It provides various methods to read different types of input.
- **BufferedReader:** It is a more versatile class that can be used to read character streams from a variety of sources. It provides more control over the input, especially when dealing with complex scenarios.

The choice between **Scanner** and **BufferedReader** depends on the specific requirements of the program. **Scanner** is more user-friendly and suitable for simple input parsing, while **BufferedReader** offers more control and is often preferred for more complex input scenarios. Both classes are part of the Java I/O package and are widely used in Java programming for handling user input.

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 Car {
    String brand;
    String model;
    int year;
    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }
    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }
}
public class class_object {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", "Camry", 2022);
        myCar.displayInfo();
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  f:; cd 'f:\AIDS 3SEM\AIDS_ANKIT_BARI'; & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetails
InExceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\
bin' 'class_object'
Brand: Toyota
Model: Camry
Year: 2022
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

In this Java program, we've created a simple class named **Car** that represents a car object. The class has instance variables (**brand**, **model**, and **year**), a constructor to initialize these variables, and a method (**displayInfo**) to display information about the car.

In the **Main** class, we created an object (**myCar**) of the **Car** class and used it to access the **displayInfo** method, which prints information about the car to the console.

This program illustrates the fundamental concepts of object-oriented programming (OOP) in Java, including the definition of a class, instantiation of objects, and the use of methods to perform actions associated with the objects. OOP provides a way to model and structure code in a more organized and reusable manner, making it a powerful paradigm for software development.

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:

```
public class overloading {
    public overloading() {
        System.out.println("Default Constructor");
    }
    public overloading(int num) {
        System.out.println("Parameterized Constructor with an int: " + num);
    }
    public overloading(double num) {
        System.out.println("Parameterized Constructor with a double: " + num);
    }
    public void printMessage() {
        System.out.println("Default Method");
    }
    public void printMessage(String message) {
        System.out.println("Method with a String parameter: " + message);
    }
    public void printMessage(int number) {
        System.out.println("Method with an int parameter: " + number);
    }
    public static void main(String[] args) {
```

```
        overloading defaultConstructor = new overloading();
        overloading intConstructor = new overloading(42);
        overloading doubleConstructor = new overloading(3.14);
        overloading obj = new overloading();
        obj.printMessage();
        obj.printMessage("Hello, Method Overloading!");
        obj.printMessage(100);
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java
\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'overloading'
Default Constructor
Parameterized Constructor with an int: 42
Parameterized Constructor with a double: 3.14
Default Constructor
Default Method
Method with a String parameter: Hello, Method Overloading!
Method with an int parameter: 100
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

Method and constructor overloading in Java provide flexibility and versatility to the code by allowing the same method or constructor name to be used with different parameters. This enhances code readability and reduces redundancy, as multiple variations of the same functionality can be expressed using a single method or constructor name. Overloading simplifies code maintenance and makes it more intuitive for developers to use and understand the APIs.

However, it's important to use overloading judiciously, considering the context and ensuring that the overloaded methods or constructors serve distinct purposes to avoid confusion. Overall, method and constructor overloading are valuable features in Java that contribute to the language's expressiveness and developer-friendly design.

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 com.example.util;

public class StringUtils {
    public static String reverseString(String str) {
        return new StringBuilder(str).reverse().toString();
    }
}


package com.example.app;
```

```
import com.example.util.StringUtils;

public class MainApp {
    public static void main(String[] args) {
        String original = "Hello, World!";
        String reversed = StringUtils.reverseString(original);

        System.out.println("Original: " + original);
        System.out.println("Reversed: " + reversed);
    }
}


javac com/example/util/StringUtils.java
javac com/example/app/MainApp.java
java com.example.app.MainApp
```

Output :

```
javac com/example/util/StringUtils.java
javac com/example/app/MainApp.java
java com.example.app.MainApp
```

Conclusion:

Packages in Java provide a mechanism for organizing and structuring code in a modular way. They help in avoiding naming conflicts, improving code readability, and promoting code reusability. By grouping related classes and interfaces into packages, developers can create a logical organization for their codebase.

Key benefits of using packages include:

1. Namespace Management: Packages help prevent naming conflicts by encapsulating classes within a unique namespace. This is especially important in large projects with multiple developers, reducing the chances of naming collisions.
2. Code Organization: Packages allow developers to organize their code into meaningful units. Classes related to a specific functionality or feature can be grouped together, making it easier to locate and understand the code.
3. Access Control: Packages support access control mechanisms in Java, such as public, private, and protected. This helps in defining the visibility and accessibility of classes and members, controlling the exposure of implementation details.
4. Code Reusability: Packages facilitate code reusability by providing a structured way to import and use classes from one package into another. This promotes the creation of modular and reusable components.
5. Readability and Maintainability: A well-organized package structure contributes to code readability and maintainability. It allows developers to navigate the codebase more efficiently and makes it easier to maintain and extend the system.

In conclusion, packages play a crucial role in Java programming by promoting modularity, organization, and maintainability. They contribute to building scalable and well-structured applications, making it easier for developers to collaborate and maintain code over time.

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:

```
public class two_array {
    public static void main(String[] args) {
        String[][] stringArray = {
                {"Java", "is", "fun"},
                {"Programming", "with", "arrays"},
                {"String", "functions", "are", "useful"}
        };
        System.out.println("Original 2D Array:");
        printArray(stringArray);
        reverseStrings(stringArray);
        System.out.println("\nArray after reversing strings:");
        printArray(stringArray);
        String longestString = findLongestString(stringArray);
        System.out.println("\nLongest string in the array: " + longestString);
    }
    private static void reverseStrings(String[][] array) {
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array[i].length; j++) {
                array[i][j] = new StringBuilder(array[i][j]).reverse().toString();
            }
        }
    }
    private static String findLongestString(String[][] array) {
```

```
        String longestString = "";
        for (String[] row : array) {
            for (String str : row) {
                if (str.length() > longestString.length()) {
                    longestString = str;
                }
            }
        }
        return longestString;
    }
    private static void printArray(String[][] array) {
        for (String[] row : array) {
            for (String str : row) {
                System.out.print(str + " ");
            }
            System.out.println();
        }
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java
\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'two_array'
Original 2D Array:
Java is fun
Programming with arrays
String functions are useful

Array after reversing strings:
avaJ si nuf
gnimmargorP htiw syarra
gnirtS snoitcnuf era lufesu

Longest string in the array: gnimmargorP
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

In this Java program, we demonstrated the use of a 2D array containing strings and applied two functions. First, we reversed each string in the array using a nested loop and the **StringBuilder** class. Next, we found the longest string in the array by iterating through each element. The program showcases the versatility of arrays and the manipulation of strings in a programming context.

This exercise highlights the importance of understanding array manipulation and string operations, essential skills in Java programming. Such operations are common in real-world applications, including data processing, text analysis, and algorithmic problem-solving. Developing proficiency in working with arrays and strings contributes to the ability to write efficient and effective programs across various domains.

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 Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}
public class single_inheritance {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();
        myDog.bark();
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java
\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'single_inheritance'
This animal eats food.
The dog barks.
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

Single inheritance in Java allows a class to inherit from only one superclass. It helps in building a hierarchical structure in the code, promoting code reusability. In the example above, **Dog** is a specialized version of **Animal**, inheriting the common behavior (eating) from its parent class while adding its own unique behavior (barking).

However, it's essential to use inheritance judiciously, considering the "is-a" relationship between the classes. In situations where multiple inheritance is needed, interfaces can be used in Java to achieve a similar effect.

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:

```
interface Interface1 {
    void method1();
}
interface Interface2 {
```

```
    void method2();
}
class MyClass implements Interface1, Interface2 {
    @Override
    public void method1() {
        System.out.println("Implementing method1");
    }
    @Override
    public void method2() {
        System.out.println("Implementing method2");
    }
    void myClassMethod() {
        System.out.println("Additional method in MyClass");
    }
}
public class multiple_inheritance {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.method1();
        obj.method2();
        obj.myClassMethod();
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java
\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'multiple_inheritance'
Implementing method1
Implementing method2
Additional method in MyClass
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

Multiple inheritance in Java is achieved through interfaces, allowing a class to inherit from more than one interface. This is useful for achieving a higher level of abstraction and code reusability. Each interface can declare a set of methods, and a class implementing those interfaces must provide concrete implementations for all the methods declared in the interfaces.

While Java supports multiple inheritance through interfaces, it avoids the complexities and ambiguities associated with multiple inheritance in traditional class-based languages by excluding the possibility of inheriting the implementation of multiple classes. This helps in preventing the diamond problem and maintains a clear and predictable inheritance hierarchy.

In conclusion, Java's approach to multiple inheritance through interfaces provides a flexible and clean way to design and organize code while avoiding some of the challenges associated with multiple inheritance in other programming languages.

Experiment No. 9
Implement a program on 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...
```

Code:

```java
import java.util.Scanner;
public class exception_handling {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Enter numerator: ");
            int numerator = scanner.nextInt();
            System.out.print("Enter denominator: ");
            int denominator = scanner.nextInt();
            int result = numerator / denominator;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java
\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'exception_handling'
Enter numerator: 100
Enter denominator: 0
Error: Division by zero is not allowed.
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

Exception handling is a crucial aspect of robust software development. It allows programs to gracefully handle unexpected situations and prevent abrupt terminations. In Java, the try-catch mechanism provides a structured way to deal with exceptions. By handling exceptions appropriately, developers can enhance the reliability and resilience of their applications.

It's important to choose the appropriate level of granularity for exception handling and to provide meaningful error messages to aid in debugging. Additionally, the **finally** block is useful for cleaning up resources, ensuring they are released regardless of whether an exception occurs.

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 MyThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getId() + " Value " + i);
        }
    }
}
public class multithreading {
    public static void main(String args[]) {
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();
        t1.start();
        t2.start();
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>  & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java
\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'multithreading'
30 Value 1
30 Value 2
29 Value 1
30 Value 3
29 Value 2
30 Value 4
29 Value 3
30 Value 5
29 Value 4
29 Value 5
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI>
```

Conclusion:

1. **Concurrency:** Multithreading enables concurrent execution of tasks, making it possible to execute multiple threads simultaneously. This can lead to improved performance and responsiveness in applications.

2. **Resource Sharing:** Threads share the same process resources, which can be beneficial when multiple tasks need to access shared data. However, it also introduces challenges related to synchronization and data consistency.

3. **Responsiveness:** Multithreading is essential for creating responsive user interfaces and handling parallel tasks efficiently. For example, a user interface thread can remain responsive while a background thread performs time-consuming operations.

4. **Complexity:** While multithreading offers advantages, it also introduces complexity. Developers need to be careful about synchronization issues, race conditions, and deadlocks, which can be challenging to debug.

5. **Thread Lifecycle:** Understanding the lifecycle of a thread is crucial for effective multithreading. Threads go through states such as new, runnable, blocked, waiting, and terminated. Managing these states properly is essential for correct program behavior.

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.

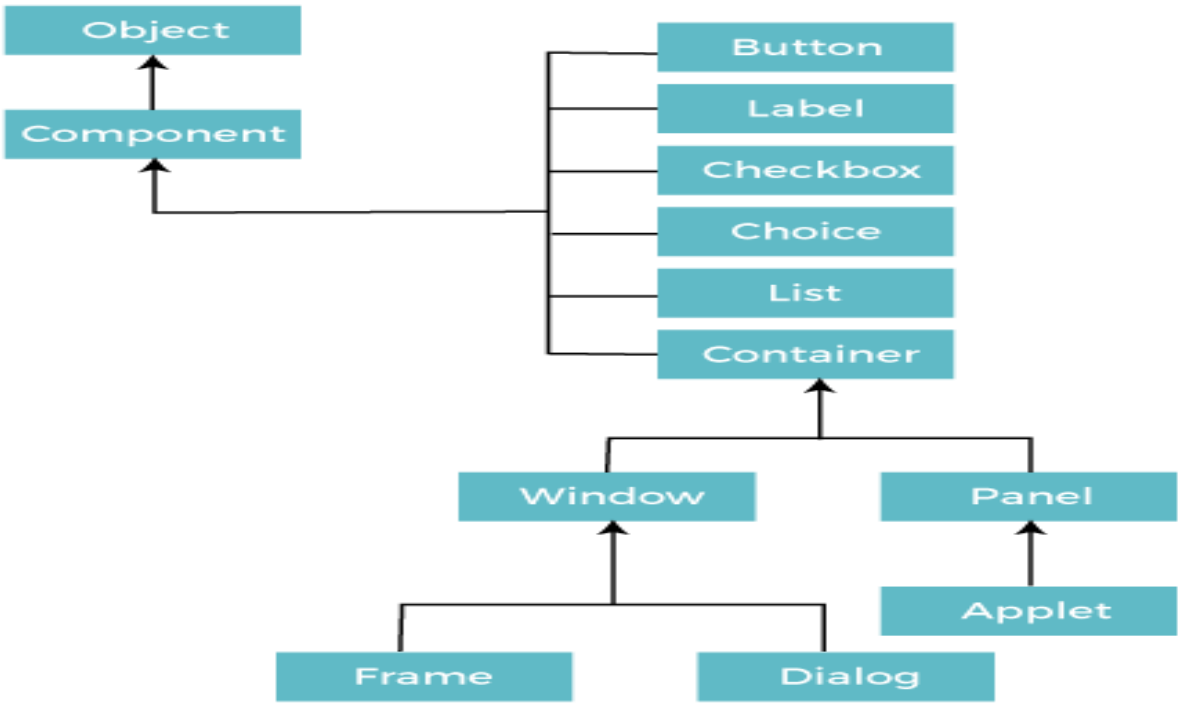
1. A general interface between Java and the native system, used for windowing, events and layout managers. This API is at the core of Java GUI programming and is also used by Swing and Java 2D. It contains the interface between the native windowing system and the Java application1.
2. A basic set of 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



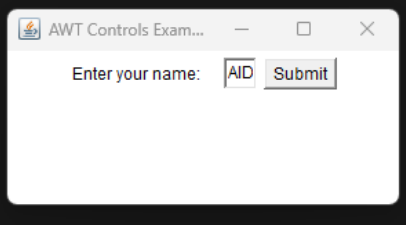
Code:

```
import java.awt.*;
import java.awt.event.*;

public class applet {
    private Frame frame;
    private Label label;
    private TextField textField;
    private Button button;

    public applet() {
        frame = new Frame("AWT Controls Example");
        label = new Label("Enter your name:");
        textField = new TextField();
        button = new Button("Submit");

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String name = textField.getText();
```

```
                System.out.println("Hello, " + name + "!");
            }
        });

        frame.setLayout(new FlowLayout());
        frame.add(label);
        frame.add(textField);
        frame.add(button);

        frame.setSize(300, 150);
        frame.setVisible(true);

        // Close the frame on window close
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        });
    }

    public static void main(String[] args) {
        new applet();
    }
}
```

Output :

```
PS F:\AIDS 3SEM\AIDS_ANKIT_BARI> & 'C:\Program Files\Java\jdk-20.0.2\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaceStorage\3ebb6b54b9e23235e398037b07662649\redhat.java
\jdt_ws\AIDS_ANKIT_BARI_9a7da79b\bin' 'applet'
Hello, AIDS!
```



Conclusion:

AWT (Abstract Window Toolkit) provides a set of basic building blocks for creating graphical user interfaces in Java. While it is an older technology and has been largely superseded by Swing and JavaFX for modern UI development, it still serves as a fundamental framework for GUI components in Java.

The example above showcases the simplicity of creating a basic UI with AWT controls such as Frame, Label, TextField, and Button. If you are developing a more complex and modern application, consider using Swing or JavaFX for a richer set of features and better performance.

Experiment No. 12
Course Project: Quiz Application Using AWT.
Date of Performance:
Date of Submission: