Vidyavardhini's
College of Engineering & Technology
Vasai Road (W)

Department of Artificial Intelligence & Data Science

Laboratory Manual
(Student Copy)

Semester	III	Class	S.E.(2024-25)
Course Code	CSL304		
Course Name	Skill based Lab Course: Object Oriented Programming with Java		
Student Name	Piyush Vitthal Kurwade		
Roll NO_Class	27_SE (AI&DS)		

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 a technologically inspiring environment for 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 solving the challenges in emerging areas.
- To inculcate standard industrial practices and security norms while dealing with Data.
- To develop sustainable Artificial Intelligence systems for the benefit of various 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 using super and final keyword.	-	-	-	3	-	-
Implement a program on Single inheritance.	-	-	-	3	-	-
Implement a program on Exception handling.	-	-	-	-	3	-

Implement a program on User Defined Exception.	-	-	-	-	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 using super and final keyword.				
8	Implement a program on Single Inheritance with Interface.				
9	Implement a program on Exception Handling.				
10	Implement a program on User Defined Exception.				
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

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

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 a 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 ControlFlowExample {

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter an integer: ");
    int num = scanner.nextInt();
    scanner.close();

    if (num > 0) {
      System.out.println(num + " is a positive number.");
    } else if (num < 0) {
      System.out.println(num + " is a negative number.");
    } else {
      System.out.println("You entered zero.");
    }

    if (num >= 0) {
      if (num % 2 == 0) {
        System.out.println(num + " is an even number.");
      } else {
        System.out.println(num + " is an odd number.");
      }
    }
```

```
System.out.print("Enter a grade (A/B/C/D/F): ");
char grade = scanner.next().charAt(0);
switch (grade) {
   case 'A':
      System.out.println("Excellent!");
      break;
   case 'B':
      System.out.println("Good job.");
      break;
   case 'C':
      System.out.println("Well done.");
      break;
   case 'D':
      System.out.println("You passed.");
      break;
   case 'F':
      System.out.println("Better try again.");
      break;
   default:
      System.out.println("Invalid grade.");
}

int sum = 0;
for (int i = 1; i <= 5; i++) {
   sum += i;
}
System.out.println("Sum of numbers from 1 to 5 is: " + sum);

System.out.print("Enter a number to calculate its factorial: ");
int n = scanner.nextInt();
long factorial = 1;
```

```
int i = 1;

while (i <= n) {
   factorial *= i;
   i++;
}
System.out.println("Factorial of " + n + " is: " + factorial);

int positiveNum;
do {
   System.out.print("Enter a positive number: ");
   positiveNum = scanner.nextInt();
} while (positiveNum <= 0);
System.out.println("You entered a positive number: " + positiveNum);

int[] numbers = {1, 2, 3, 4, 5};
System.out.print("Numbers in the array: ");
for (int number : numbers) {
   System.out.print(number + " ");
}

System.out.println("\nNumbers till break:");
for (int j = 1; j <= 10; j++) {
   if (j == 5) {
      break;
   }
   System.out.print(j + " ");
}

System.out.println("\nNumbers skipping 5:");
for (int j = 1; j <= 10; j++) {
```

```
            if (j == 5) {
                continue;
            }
            System.out.print(j + " ");
        }

        scanner.close();
    }
}
```

Output:

Enter an integer: 5

5 is a positive number.

5 is an odd number.

Enter a grade (A/B/C/D/F): A

Excellent!

Sum of numbers from 1 to 5 is: 15

Enter a number to calculate its factorial: 5

Factorial of 5 is: 120

Enter a positive number: -1

Enter a positive number: -5

Enter a positive number: 3

You entered a positive number: 3

Numbers in the array: 1 2 3 4 5

Numbers till break:

1 2 3 4

Numbers skipping 5:

1 2 3 4 6 7 8 9 10

Conclusion:

Branching and looping are essential constructs in programming that make problem-solving both flexible and efficient. In this program:

- **Branching (if-else)** allows the program to handle different scenarios based on user input. It ensures that only valid inputs (non-negative integers) are processed, preventing errors like calculating the factorial of a negative number. This decision-making ability is crucial in solving problems that require different outcomes based on specific conditions.
- **Looping (for loop)** is used to perform repetitive calculations, such as multiplying numbers to compute the factorial. Looping enables the program to efficiently handle repetitive tasks without having to manually write each step, which is particularly useful when the number of steps isn't known beforehand.

Together, these constructs simplify complex problem-solving by allowing programs to make decisions and automate repetitive tasks, making them highly effective in a wide range of scenarios.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

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 InputExample {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter a string using BufferedReader: ");
        String input1 = reader.readLine();
        System.out.println("You entered: " + input1);

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer using Scanner: ");
        int input2 = scanner.nextInt();
        System.out.println("You entered: " + input2);

        System.out.print("Enter a string using Scanner: ");
```

```
        scanner.nextLine();  // Consume newline
        String input3 = scanner.nextLine();
        System.out.println("You entered: " + input3);


        scanner.close();
    }
}
```

Output:

Enter a string using BufferedReader: Hello from BufferedReader

You entered: Hello from BufferedReader

Enter an integer using Scanner: 42

You entered: 42

Enter a string using Scanner: Hello from Scanner

You entered: Hello from Scanner

Conclusion:

The BufferedReader class is used to read a sequence of characters efficiently, capturing input with the readLine() method. This approach requires handling the IOException exception, which can make it slightly more complex. While it reads input as a string, any needed parsing (such as converting to integers) must be done manually.

The Scanner class provides a more modern and versatile way to read user input. It includes methods like nextInt() and nextLine() that directly read various data types, simplifying the process of capturing input. The absence of checked exceptions like IOException makes Scanner easier to use, especially for beginners.

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 color;
    String model;
    int year;

    Car(String color, String model, int year) {
        this.color = color;
        this.model = model;
        this.year = year;
    }

    void displayInfo() {
        System.out.println("Car Model: " + model);
        System.out.println("Car Color: " + color);
        System.out.println("Car Year: " + year);
    }
}

public class Main {
    public static void main(String[] args) {
        Car car1 = new Car("Red", "Toyota", 2020);
        Car car2 = new Car("Blue", "Honda", 2021);

        car1.displayInfo();
        System.out.println();
        car2.displayInfo();
    }
}
```

Output:

```
Car Model: Toyota
Car Color: Red
Car Year: 2020

Car Model: Honda
Car Color: Blue
Car Year: 2021
```

Conclusion:

Creating a class template involves defining attributes and methods that encapsulate the properties and behaviors of a particular type of object. Objects are instances of the class, created using the class constructor. This approach allows for organized, reusable code in object-oriented programming, making it easier to manage and interact with different entities in a program.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

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 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);

      Thread thread1 = new Thread();
      Thread thread2 = new Thread("MyThread");

      System.out.println("Thread 1 Name: " + thread1.getName());
```

```
        System.out.println("Thread 2 Name: " + thread2.getName());
    }
}
```

Output:
a
a 10
Thread 1 Name: Thread-0
Thread 2 Name: MyThread

Conclusion:

Both function and constructor overloading provide a way to define multiple methods or constructors with the same name but different parameter lists. This feature enhances the expressiveness and flexibility of the code, allowing developers to create more intuitive and reusable classes. By using overloading, Java enables better management of complex systems, making it easier to understand and maintain the code.

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 myPackage;

public class MyClass {
    public void display() {
        System.out.println("Hello from MyClass in myPackage.");
    }
}

class MainClass {
    public static void main(String[] args) {
        MyClass myClass = new MyClass();
        myClass.display();
    }
}
```

Output:

```
Hello from MyClass in myPackage.
```

Conclusion:

Autoencoders are a type of artificial neural network used primarily for unsupervised learning tasks, particularly in dimensionality reduction and image compression. The architecture typically consists of two main components: the encoder and the decoder. The encoder compresses the input data into a lower-dimensional representation, often referred to as the latent space or bottleneck. The decoder then reconstructs the original input from this compressed representation.

The autoencoder learns to capture the essential features of the input data while ignoring noise and less important information. By training the network to minimize the difference between the original input and its reconstruction,

autoencoders can effectively learn efficient representations of the data. In the context of image compression, this allows the model to reduce the size of image files while retaining significant visual information.

Image compression results using autoencoders can be quite promising. The network can achieve high compression ratios while preserving the quality of the images. The performance depends on the architecture of the autoencoder, such as the number of layers, the types of activation functions used, and the size of the latent space. When properly trained, autoencoders can reconstruct images that are visually similar to the original images, making them suitable for applications where storage and bandwidth are limited.

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 Main {
  public static void main(String[] args) {
    int[] oneDArray = {1, 2, 3, 4, 5};
    int[][] twoDArray = {{1, 2}, {3, 4}, {5, 6}};
```

```
        System.out.println("One-Dimensional Array:");
        for (int i = 0; i < oneDArray.length; i++) {
            System.out.print(oneDArray[i] + " ");
        }

        System.out.println("\nTwo-Dimensional Array:");
        for (int i = 0; i < twoDArray.length; i++) {
            for (int j = 0; j < twoDArray[i].length; j++) {
                System.out.print(twoDArray[i][j] + " ");
            }
            System.out.println();
        }

        String demoString = "GeeksforGeeks";
        String anotherString = new String("Welcome");

        System.out.println("String Literal: " + demoString);
        System.out.println("Using New Keyword: " + anotherString);
    }
}
```

Output:

One-Dimensional Array:

1 2 3 4 5

Two-Dimensional Array:

1 2

3 4

5 6

String Literal: GeeksforGeeks

Using New Keyword: Welcome

Conclusion:

The two-dimensional array is used to represent a matrix-like structure, allowing for organized storage of data in rows and columns. This structure is useful for a variety of applications, such as representing graphs, game boards, or any data that can be logically arranged in a grid. By iterating through the rows and columns, the code effectively displays the contents of the 2D array, showcasing its ability to store and manage related data.

Strings are utilized to demonstrate two different methods of initialization. The string literal approach showcases how Java efficiently manages memory by reusing existing string objects in the string constant pool. On the other hand, using the `new` keyword illustrates how a new string object is created in the heap, which can be useful when explicitly requiring a new instance. The output displays both strings clearly, emphasizing the flexibility of string handling in Java.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Experiment No. 7
Implement a program using super and final keyword.
Date of Performance:
Date of Submission:

Aim: To implement the concept of super and final keyword.

Objective: To understand the usage of superclass and final method , variables and class

Theory:

super and **final** keywords are two popular and useful keywords in Java. They also play a significant role in dealing with Java programs and their classes. In this chapter, you will learn about how to use super and final within a Java program.

Syntax: `super.<method-name>();`

- Super variables refer to the variable of a variable of the parent class.
- Super() invokes the constructor of immediate parent class.
- Super refers to the method of the parent class

Instance refers an instance variable of the current class by default, but when you have to refer parent class instance variable, you have to use super keyword to distinguish between parent class (here employee) instance variable and current class (here, clerk) instance variable.

What is final in Java?

Final is a keyword in Java that is used to restrict the user and can be used in many respects. Final can be used with:

- Class
- Methods
- Variables

A method declared as final cannot be overridden; this means even when a child class can call the final method of parent class without any issues, but the overriding will not be possible.

Once a variable is assigned with the keyword final, it always contains the same exact value. Again things may happen like this; if a final variable holds a reference to an object then the state of the

object can be altered if programmers perform certain operations on those objects, but the variable will always refer to the same object. A final variable that is not initialized at the time of declaration is known as a blank final variable. If you are declaring a final variable in a constructor, then you must initialize the blank final variable within the constructor of the class. Otherwise, the program might show a compilation error.

Code:

```
class Employee {
   String name;
   final int id;

   Employee(String name, int id) {
      this.name = name;
      this.id = id;
   }

   void display() {
      System.out.println("Employee Name: " + name);
      System.out.println("Employee ID: " + id);
   }
}

class Clerk extends Employee {
   String department;

   Clerk(String name, int id, String department) {
      super(name, id);
      this.department = department;
   }

   void display() {
      super.display();
      System.out.println("Department: " + department);
   }
}

public class Main {
   public static void main(String[] args) {
      Clerk clerk = new Clerk("John Doe", 101, "Sales");
      clerk.display();
   }
}
```

Output:
Employee Name: John Doe
Employee ID: 101
Department: Sales

Conclusion:

The final keyword is applied to the variable id in the employee class, which ensures that once assigned, this variable cannot be changed. This is crucial for maintaining the integrity of the employee's identification number throughout the lifecycle of the object. It showcases how final can be used to enforce immutability at the variable level, making it a fundamental concept for creating stable and predictable classes.

The super keyword is used in the clerk class to invoke the constructor and method of the parent Employee class. This allows the clerk class to inherit properties and behaviors from, Employee promoting code reusability and establishing a clear relationship between the two classes. By calling SUper (name, id) in the constructor, the code initializes the inherited properties of Employee, while the Super.display() call enables the Clerk class to leverage the functionality defined in the parent class.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Experiment No. 8
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 behavior 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 a 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 is 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 Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
        dog.bark();
    }
}
```

Output:

This animal eats food.
The dog barks.

Conclusion:

Single inheritance is a powerful feature in Java that facilitates the extension of existing classes, promoting code reuse and reducing redundancy. It helps in creating a well-organized code structure and allows for easier management of relationships between different classes. This principle is essential for building efficient, reliable, and maintainable software applications in object-oriented programming.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

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:

```
public class ExceptionHandlingExample {
   public static void main(String args[]) {
      try {
         int data = 100 / 0;
      } catch (ArithmeticException e) {
         System.out.println(e);
      } finally {
         System.out.println("This block is executed regardless of an exception.");
      }
      System.out.println("rest of the code...");
   }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
This block is executed regardless of an exception.
rest of the code...
```

Conclusion:

In Java, exceptions are handled using a structured approach that allows developers to manage runtime errors effectively. The primary mechanism for handling exceptions involves the use of five keywords: try, catch, finally, throw, and throws.

The try block is where code that may potentially throw an exception is placed. If an exception occurs, control is transferred to the corresponding catch block, which contains the code to handle the exception. This separation allows for a cleaner error management strategy, where the main logic of the program remains distinct from the error handling code. The finally block, when present, is always executed after the try and catch blocks, ensuring that necessary cleanup operations or resource releases occur regardless of whether an exception was thrown.

The throw keyword is used to explicitly throw an exception, while the throws keyword is included in method signatures to declare that a method may throw one or more exceptions, allowing callers to handle these exceptions appropriately.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Experiment No. 10
Implement program on User Defined Exception
Date of Performance:
Date of Submission:

Aim: Implement program on User Defined Exception.

Objective:

Theory:

An exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed.

Java provides us the facility to create our own exceptions which are basically derived classes of Exception. Creating our own Exception is known as a custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user needs. In simple words, we can say that a User-Defined Exception or custom exception is creating your own exception class and throwing that exception using the 'throw' keyword.

For example, MyException in the below code extends the Exception class.

Why use custom exceptions?

Java exceptions cover almost all the general types of exceptions that may occur in the programming. However, we sometimes need to create custom exceptions.

Following are a few of the reasons to use custom exceptions:

- To catch and provide specific treatment to a subset of existing Java exceptions.
- Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create a custom exception, we need to extend the Exception class that belongs to **java.lang package.**

Example: We pass the string to the constructor of the superclass- Exception which is obtained using the "getMessage()" function on the object created.

// A Class that represents use-defined exception

```
class MyException extends Exception {
        public MyException(String s)
        {
                // Call constructor of parent Exception
                super(s);
        }
}


// A Class that uses above MyException
public class Main {
        // Driver Program
        public static void main(String args[])
        {
                try {
                        // Throw an object of user defined exception
                        throw new MyException("UserDefined Exception");
                }
                catch (MyException ex) {
                        System.out.println("Caught");
```

```
                // Print the message from MyException object
                System.out.println(ex.getMessage());
            }
        }
}
```

Output:

```
Caught
UserDefined Exception
```

Code:

```
class MyException extends Exception {
    public MyException(String s) {
        super(s);
    }
}

public class Main {
    public static void main(String args[]) {
        try {
            throw new MyException("UserDefined Exception");
        } catch (MyException ex) {
            System.out.println("Caught");
            System.out.println(ex.getMessage());
        }
    }
}
```

Output:

Caught

UserDefined Exception

Conclusion:

User-defined exceptions in Java provide a powerful mechanism for developers to create custom error handling that aligns with the specific needs of their applications. By extending the Exception class, developers can create exceptions that convey meaningful information related to their business logic. This enhances the clarity and maintainability of the code, allowing for specific error handling strategies tailored to the application's requirements.

When a user-defined exception is thrown, it can be caught and managed separately from standard exceptions. This enables developers to provide precise feedback and solutions for errors unique to their application's context. For example, a custom exception can be used to handle validation errors, workflow issues, or business logic violations, allowing developers to pinpoint issues and respond accordingly.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

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.

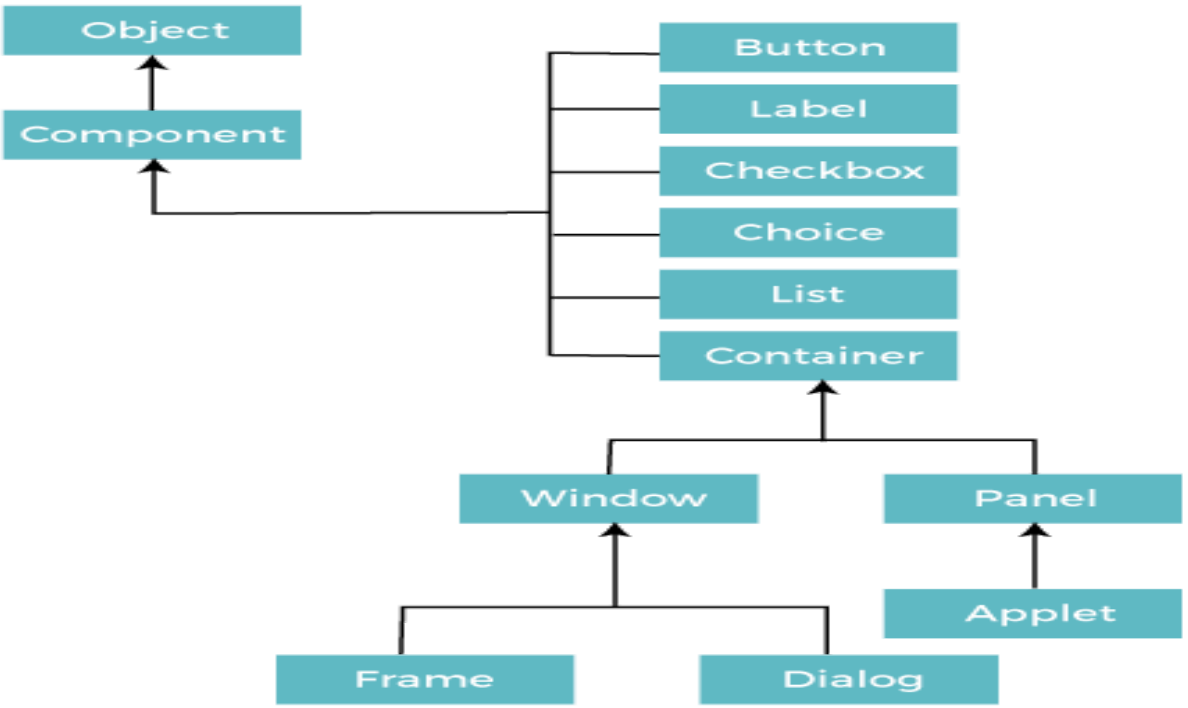
1. A general interface between Java and the native system, used for windowing, events and layout managers. This API is at the core of Java GUI programming and is also used by Swing and Java 2D. It contains the interface between the native windowing system and the Java application1.
2. A basic set of 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 AWTExample {
    public static void main(String[] args) {
        Frame frame = new Frame("AWT Example");
        Label label = new Label("Enter your name:");
        TextField textField = new TextField();
        Button button = new Button("Submit");

        label.setBounds(20, 50, 150, 20);
```

```
        textField.setBounds(20, 80, 150, 20);
        button.setBounds(20, 110, 80, 30);

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Name: " + textField.getText());
            }
        });

        frame.add(label);
        frame.add(textField);
        frame.add(button);
        frame.setSize(300, 200);
        frame.setLayout(null);
        frame.setVisible(true);

        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        });
    }
}
```

Output:

When the program is run, a window will appear with a label, a text field, and a button. You can enter a name in the text field and click the button. The entered name will be printed to the console. The appearance of the GUI will depend on the operating system you are using.

Conclusion:

Application development using AWT (Abstract Window Toolkit) controls allows developers to create graphical user interfaces (GUIs) for Java applications. AWT provides a variety of components, such as buttons, text fields, labels, checkboxes, and more, which enable the creation of interactive applications.

AWT components are heavyweight, meaning they rely on the underlying operating system for rendering and functionality. This can lead to a native look and feel, making AWT applications appear consistent with the host OS's interface. However, this dependence also means that the appearance and behavior of components can vary across different platforms, which may introduce challenges in achieving a uniform user experience.

The event-handling mechanism in AWT allows developers to define how the application should respond to user actions, such as button clicks or text entry. This capability is crucial for creating responsive applications. Developers can implement custom behavior by adding action listeners to components, which helps manage user interactions effectively.

While AWT is suitable for basic GUI applications, it has limitations compared to more advanced frameworks like Swing or JavaFX. These alternatives offer more sophisticated components and better design flexibility, making them preferable for complex applications.

Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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

This is my Infosys Springboard Certification.

1-3c2b300a-3110-42db-be83-9a0f1844a2ac.pdf