Vidyavardhini's

College of Engineering & Technology

Vasai Road (W)

Department of Artificial Intelligence & Data Science

Laboratory ManualStudent Copy

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

Vidyavardhini's College of Engineering& Technology

Vision

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

Mission

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

Department Vision:

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

Department Mission:

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

Program Specific Outcomes (PSOs):

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

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

Program Outcomes (POs):

Engineering Graduates will be able to:

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

Course Objective

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

Course Outcomes

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

Mapping of Experiments with Course Outcomes

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

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

INDEX

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

D.O.P: Date of performance

D.O.C : Date of correction

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

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

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

Theory :-

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

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

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

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

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

- while
- do-while

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

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

Code: -

Conclusion:

Branching and looping are two fundamental concepts in computer programming that are used to solve a wide variety of problems. Branching allows a program to make different decisions based on the value of a condition. This is useful for solving problems where the program needs to take different actions depending on the input or the current state of the program. For example, a program that calculates the shipping cost for an item might need to branch on the weight of the item to determine the shipping rate. Branching and looping can be used together to solve even more complex problems. For example, a program that sorts a list of numbers might use a loop to iterate over the list and compare each element to the next. If the elements are out of order, the program can use a branch to swap the elements.

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:

Conclusion:

The BufferedReader class is older and more basic, but it is still widely used. The Scanner class is newer and more advanced, and it provides a number of features that make it easier to use for accepting user input.

To use the BufferedReader class to accept user input, we would first create a BufferedReader object that wraps around the System.in stream. This stream represents the standard input stream, which is where the user's input is read from. we would then use the readLine() method to read the user's input as a line of text.

To use the Scanner class to accept user input, we would first create a Scanner object that wraps around the System.in stream. we would then use the appropriate next() method to read the user's input as a specific data type, such as a string, integer, or double.

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:

Conclusion:

In Java, creating a class involves defining a blueprint that specifies the structure and behavior of objects. The class acts as a template for creating objects. Objects are instances of classes that encapsulate data and methods to interact with that data. Classes and objects are fundamental concepts in Java, allowing for the creation of reusable and modular code by defining templates that can be instantiated to create multiple instances, each with its own state and behavior.

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:

Conclusion:

Function overloading in Java allows multiple methods to have the same name but with different parameters. This way, you can create methods that perform similar tasks but can handle different types or a different number of arguments. Overloading is based on the number and types of parameters in the methods. Constructor overloading follows a similar concept. In Java, constructors can also be overloaded by creating multiple constructors within a class, each having a different parameter list.

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:

Conclusion:

Autoencoders are a type of neural network commonly used for unsupervised learning and data compression. They consist of an encoder and a decoder, which work together to learn efficient representations of data, and they have applications in image compression. Autoencoder-based image compression has a number of advantages over traditional image compression algorithms. First, autoencoders are able to learn a more efficient representation of the image, which can lead to smaller compressed file sizes. Second, autoencoders are able to reconstruct the image with high fidelity, even at high compression ratios.

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:

Conclusion:

Java offers powerful support for strings and 2D arrays, allowing for a wide range of applications in various domains.
In strings:
Text Processing: Utilizing strings for handling and manipulating text inputs, including parsing, searching, modifying, and formatting data.
User Interface and Output: Generating messages, prompts, and text-based responses for user interfaces.
Data Processing: Tokenizing strings to extract specific information or break down data into smaller units.

In 2D Array:
Matrices and Grids: Representing structured data in the form of rows and columns, such as matrices for mathematical computations or grid-based games.
Image Processing: Storing pixel information in a 2D array format for image manipulation and processing.

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:

Conclusion:

Single inheritance is a fundamental concept in object-oriented programming that allows a class to inherit properties and behavior from a single superclass or parent class. Single inheritance makes it easier to trace the inheritance hierarchy of a class and to understand how the class's methods and properties are implemented. It can lead to redundant code, as the same code may need to be implemented in both the parent class and the child class.

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:

Conclusion:

Interfaces in Java serve as a key feature of the language, offering a way to achieve abstraction, multiple inheritance, and the definition of a contract for classes. Interfaces can help to reduce the amount of duplicate code in a system. By implementing interfaces, classes can share common functionality without having to duplicate code.

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:

Conclusion:

Exceptions in Java are a mechanism used to handle errors, exceptional conditions, and unexpected situations that may occur during the execution of a program. Java's exception handling provides a structured way to deal with errors, making code more robust and maintainable. It provides a structured and reliable approach to deal with errors and unexpected situations, promoting more robust and reliable software development. Proper handling of exceptions is vital for maintaining the stability and reliability of Java applications.

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:

Conclusion:

Multithreading is a powerful tool that can be used to improve the performance, responsiveness, and simplicity of Java applications. However, multithreading can be complex to use, so it is important to understand the basics before using it in your own code. Multithreading can improve the performance of applications by allowing multiple tasks to run simultaneously. This can be especially beneficial for applications that perform long-running tasks, such as I/O operations and database queries. Java's multithreading support offers a robust foundation for creating concurrent applications, with various tools and constructs to manage threads, synchronization, and coordination among threads.

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

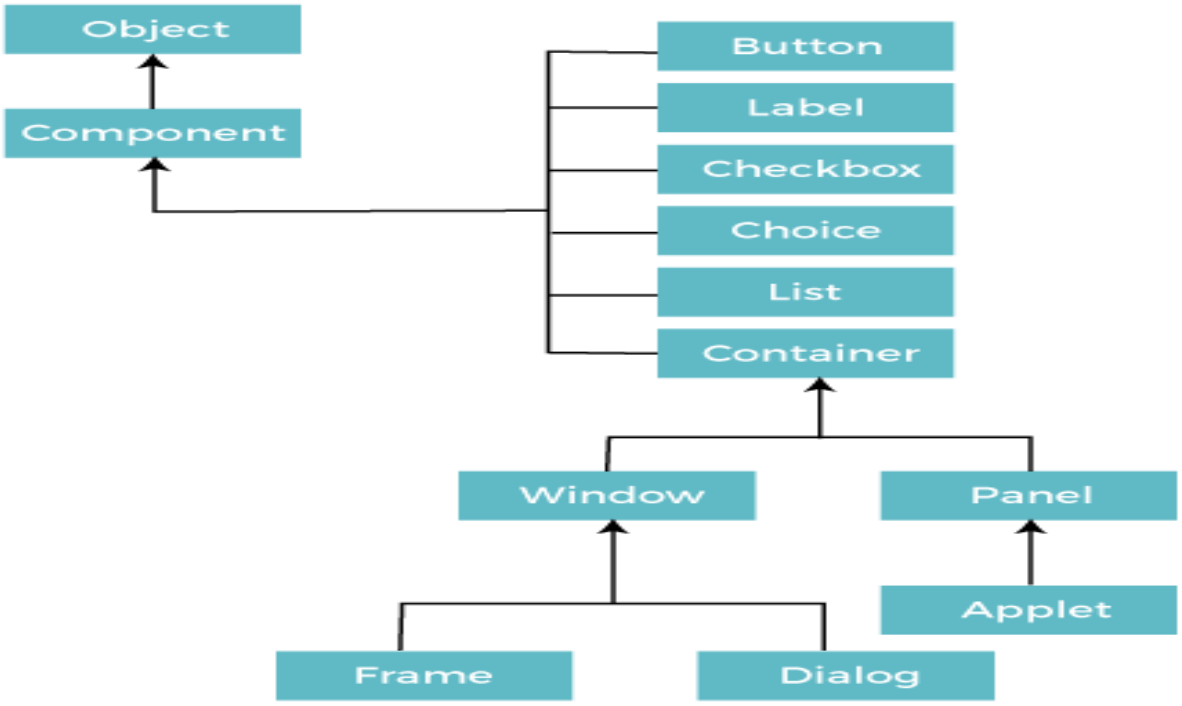
Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

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:

Conclusion:

AWT (Abstract Window Toolkit) in Java is one of the foundational GUI (Graphical User Interface) libraries used for creating applications with graphical interfaces. AWT controls provide the building blocks for creating windows, handling user input, and rendering basic UI elements. AWT Controls are heavyweight, meaning that they are directly mapped to the underlying operating system's widgets. This makes AWT Controls platform-dependent, but it also gives them a more native look and feel. AWT Controls can be used to develop desktop applications such as word processors, spreadsheets, and image editors. AWT Controls can be used to develop web applications such as online shopping carts and online banking applications.

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

CODE:

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.Timer;
import javax.swing.JPanel;
public class GamePlay extends JPanel implements KeyListener, ActionListener {
private boolean play = false;
private int score = 0;
private int totalBricks = 21;
private Timer timer;
private int delay = 8;
private int playerX = 310;
private int ballposX = 120;
private int ballposY = 350;
private int ballXdir = -1;
private int ballYdir = -2;
private MapGenerator map;
public GamePlay() {
map = new MapGenerator(3, 7);
addKeyListener(this);
setFocusable(true);
setFocusTraversalKeysEnabled(false);
timer = new Timer(delay, this);
timer.start();
}
public void paint(Graphics g) {
g.setColor(Color.white);
g.fillRect(1, 1, 692, 592);
map.draw((Graphics2D)g);
g.setColor(Color.yellow);
g.fillRect(0, 0, 3, 592);
g.fillRect(0, 0, 692, 3);
g.fillRect(691, 0, 3, 592);
g.setColor(Color.blue);
g.fillRect(playerX, 550, 100, 8);
g.setColor(Color.green);
g.fillOval(ballposX, ballposY, 20, 20);
g.setColor(Color.black);
```

```
g.setFont(new Font("serif", Font.BOLD, 25));
g.drawString("" + score, 590, 30);
if (totalBricks <= 0) {
play = false;
ballXdir = 0;
ballYdir = 0;
g.setColor(Color.green);
g.setFont(new Font("serif", Font.BOLD, 30));
g.drawString("You Won, Score: " + score, 190, 300);
g.setFont(new Font("serif", Font.BOLD, 20));
g.drawString("Press Enter to Restart.", 230, 350);
}
if(ballposY > 570) {
play = false;
ballXdir = 0;
ballYdir = 0;
g.setColor(Color.red);
g.setFont(new Font("serif", Font.BOLD, 30));
g.drawString("Game Over, Score: " + score, 190, 300);
g.setFont(new Font("serif", Font.BOLD, 20));
g.drawString("Press Enter to Restart.", 230, 350);
}
g.dispose();
}
@Override
public void actionPerformed(ActionEvent arg0) {
// TODO Auto-generated method stub
timer.start();
if(play) {
// Ball - Pedal interaction
if(new Rectangle(ballposX, ballposY, 20, 20).intersects(new
Rectangle(playerX, 550, 100, 8))) {
ballYdir = - ballYdir;
}
for( int i = 0; i<map.map.length; i++) {
for(int j = 0; j<map.map[0].length; j++) {
if(map.map[i][j] > 0) {
int brickX = j*map.brickWidth + 80;
int brickY = i*map.brickHeight + 50;
int brickWidth= map.brickWidth;
int brickHeight = map.brickHeight;
Rectangle rect = new Rectangle(brickX, brickY,
brickWidth, brickHeight);
Rectangle ballRect = new Rectangle(ballposX,
ballposY, 20,20);
Rectangle brickRect = rect;
```

```
if(ballRect.intersects(brickRect) ) {
map.setBrickValue(0, i, j);
totalBricks--;
score+=5;
if(ballposX + 19 <= brickRect.x || ballposX
+1 >= brickRect.x + brickRect.width)
ballXdir = -ballXdir;
else {
ballYdir = -ballYdir;
}
}
}
}
}
ballposX += ballXdir;
ballposY += ballYdir;
if(ballposX < 0) {
ballXdir = -ballXdir;
}
if(ballposY < 0) {
ballYdir = -ballYdir;
}
if(ballposX > 670) {
ballXdir = -ballXdir;
}
}
repaint();
}
@Override
public void keyTyped(KeyEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void keyPressed(KeyEvent arg0) {
// TODO Auto-generated method stub
if(arg0.getKeyCode() == KeyEvent.VK_RIGHT) {
if(playerX >= 600) {
playerX = 600;
} else {
moveRight();
}
}
if(arg0.getKeyCode() == KeyEvent.VK_LEFT) {
if(playerX < 10) {
playerX = 10;
} else {
```

```
moveLeft();
}
}
if(arg0.getKeyCode() == KeyEvent.VK_ENTER) {
if(!play) {
play = true;
ballposX = 120;
ballposY = 350;
ballXdir = -1;
ballYdir = -2;
score = 0;
totalBricks = 21;
map = new MapGenerator(3,7);
repaint();
}
}
}
public void moveRight() {
play = true;
playerX += 20;
}
public void moveLeft() {
play = true;
playerX -= 20;
}
@Override
public void keyReleased(KeyEvent arg0) {
// TODO Auto-generated method stub
}
}
import javax.swing.JFrame;
public class Main {
public static void main(String[] args) {
// TODO Auto-generated method stub
JFrame obj = new JFrame();
GamePlay gamePlay = new GamePlay();
obj.setBounds(10, 10, 700, 600);
obj.setTitle("Brick Breaker");
obj.setResizable(false);
obj.setVisible(true);
obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
obj.add(gamePlay);
}
}
import java.awt.BasicStroke;
import java.awt.Color;
```

```
import java.awt.Graphics2D;
public class MapGenerator {
public int map [][];
public int brickWidth;
public int brickHeight;
public MapGenerator(int row, int col) {
map = new int [row][col];
for (int i = 0; i < map.length; i++) {
for (int j=0; j< map[0].length;j++) {
map[i][j] = 1;
}
}
brickWidth = 540/col;
brickHeight = 150/row;
}
public void draw(Graphics2D g) {
for (int i = 0; i < map.length; i++) {
for (int j=0; j< map[0].length;j++) {
if(map[i][j] > 0) {
g.setColor(Color.black);
g.fillRect(j*brickWidth + 80, i*brickHeight + 50, brickWidth,
brickHeight);
g.setStroke(new BasicStroke(3));
g.setColor(Color.white);
g.drawRect(j*brickWidth + 80, i*brickHeight + 50,
brickWidth, brickHeight);
}
}
}
}
public void setBrickValue(int value, int row, int col) {
map[row][col] = value;
}
}
```

OUTPUT: