

**LAB MANUAL  
OF  
PROGRAMMING IN JAVA  
CIC-258**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Maharaja Agrasen Institute of Technology, PSP area,  
Sector – 22, Rohini, New Delhi – 110085  
(Affiliated to Guru Gobind Singh Indraprastha University,  
New Delhi)



# **MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

## **VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

## **MISSION**

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

### **Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and encouraging inquisitive nature.

### **Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

### **Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

### **Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

### **Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics, tutorials and related tools to create a platform for multi-disciplinary approach.

### **Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successful entrepreneurs and responsible citizens.



## **MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

### **COMPUTER SCIENCE & ENGINEERING DEPARTMENT**

#### **VISION**

“To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values.”

#### **MISSION**

“To foster an open, multidisciplinary and highly collaborative research environment to produce world-class engineers capable of providing innovative solutions to real life problems and fulfill societal needs.”

# **INDEX OF THE CONTENTS**

<b>1.</b>	<b>Introduction to the lab</b>	<b>1</b>
<b>2.</b>	<b>Lab Requirements (details of H/W &amp; S/W to be used)</b>	<b>2</b>
<b>3.</b>	<b>Marking scheme for the Practical Exam</b>	<b>15</b>
<b>4.</b>	<b>Format of the lab record to be prepared by the students.</b>	
<b>5.</b>	<b>List of Experiments as per GGSIPU</b>	<b>3</b>
<b>6.</b>	<b>List of experiments beyond the syllabus</b>	<b>9</b>
<b>7.</b>	<b>Instructions for each Lab Experiment</b>	<b>17</b>
<b>8.</b>	<b>Sample Viva – Questions</b>	<b>82</b>

# 1. INTRODUCTION TO THE LAB

## LAB OBJECTIVE

The goal of this course is to provide students with the ability to write programs in Java and apply concepts described in the Object-Oriented Programming course. Java as a class-based and pure OOP language is used to demonstrate and implement appropriate concepts and techniques. The students are exposed to the concepts, fundamental syntax, and the thought processes behind object-oriented programming. By the end of course students will acquire the basic knowledge and skills necessary to implement object-oriented programming techniques in software development in Java. Each practical class will culminate in an assessed exercise.

## Course Outcomes

On successful completion of this Course, students should be able to:

Course Outcomes (CO)												
CO 1	Ability to understand the compilation process of Java, role of JVM as an emulator and various types of instructions.											
CO 2	Ability to learn and apply concepts of Java programming, exceptional handling and inheritance.											
CO 3	Ability to understand the use of multi-threading, AWT components and event handling mechanism in Java.											
CO 4	Ability to understand the concepts of I/O streams, JDBC, object serialization, sockets, RMI, JNI, Collection API interfaces, Vector, Stack, Hash table classes, list etc.											
Course Outcomes (CO) to Programme Outcomes (PO) mapping (scale 1: low, 2: Medium, 3: High)												
	PO01	PO02	PO03	PO04	PO05	PO06	PO07	PO08	PO09	PO10	PO11	PO12
CO 1	3	2	2	2	3	-	-	-	3	2	2	3
CO 2	3	2	2	2	3	-	-	-	3	2	2	3
CO 3	3	2	2	2	3	-	-	-	3	2	2	3
CO 4	3	2	2	2	3	-	-	-	3	2	2	3

## **2. LAB REQUIREMENTS**

### **For Java Programming**

JDK, NetBeans

Java Compatible Web Browser

This Compiler has no special hardware requirements as such. Any System with a minimum 256 MB RAM and any normal processor can use for this lab.

### 3. MARKING SCHEME FOR PRACTICAL EXAMINATION

There will be two practical exams in each semester.

- i. Internal Practical Exam
- ii. External Practical Exam

#### INTERNAL PRACTICAL EXAM

Total Marks:40

Marking of Internal Practical depends on the below rubrics.

#### Rubrics for Lab Assessment

Rubrics		0	1	2	3
		Missing	Inadequate	Needs Improvement	Adequate
R1	Is able to identify the problem to be solved and define the objectives of the experiment.	No mention is made of the problem to be solved.	An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable.	The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors.	The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors.
R2	Is able to design a reliable experiment that solves the problem.	The experiment does not solve the problem.	The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution.	The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable solution.	The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution.
R3	Is able to communicate the details of an experimental procedure clearly and completely.	Diagrams are missing and/or experimental procedure is missing or extremely vague.	Diagrams are present but unclear and/or experimental procedure is present but important details are missing.	Diagrams and/or experimental procedure are present but with minor omissions or vague details.	Diagrams and/or experimental procedure are clear and complete.
R4	Is able to record and represent data in a meaningful way.	Data are either absent or incomprehensible.	Some important data are absent or incomprehensible.	All important data are present, but recorded in a way that requires some effort to comprehend.	All important data are present, organized and recorded clearly.
R5	Is able to make a judgment about the results of the experiment.	No discussion is presented about the results of the experiment.	A judgment is made about the results, but it is not reasonable or coherent.	An acceptable judgment is made about the result, but the reasoning is flawed or incomplete.	An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered.

#### EXTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch and by an external examiner. In this exam student needs to perform the experiment allotted at the time of the examination, a sheet will be given to the student in which some details asked by the examiner needs to be written and at the last viva will be taken by the external examiner.

#### **MARKING SCHEME FOR EXTERNAL EXAMINATION:**

Total Marks: 60

Division of 60 marks is as follows

- |                                 |    |
|---------------------------------|----|
| 1. Sheet filled by the student: | 20 |
| 2. Viva Voice:                  | 15 |

3. Experiment performance:	15
4. File submitted:	10

Note:-

- Internal marks + External marks = Total marks given to the students  
(40marks)                      (60marks)                      (100 marks)
- Experiments given to perform can be from any section of the lab.



## 4. FORMAT OF THE LAB RECORD TO BE PREPARED BY THE STUDENTS

1. The front page of the lab record prepared by the students should have a cover page as displayed below.

***NAME OF THE LAB***

***Paper Code***

Font should be (Size 20", italics bold, Times New Roman)

Faculty name :

Student name :

Roll No.:

Semester :

Font should be (12", Times Roman)



Maharaja Agrasen Institute of Technology, PSP  
Area, Sector – 22, Rohini, New Delhi – 110085

Font should be (18", Times Roman)

2. The second page in the record should be the index as displayed below.

## Programming In Java

### PRACTICAL RECORD

**Paper Code :- CIC-258**

**Name of Student :-**

**University Roll No. :-**

**Branch :-**

**Section/ Group :-**

## Index

S.No	Experiment	Marks					Total Marks	Date of Performance	Date of Checking	Signature
		R1	R2	R3	R4	R5				
1										
2										
3										
4										
5										

### PROJECT DETAILS

1. TITLE :
2. MEMBERS IN THE PROJECT GROUP :
3. PROJECT REPORT ATTACHED :
  - a) YES
  - b) NO
4. SOFT COPY SUBMITTED :
  - a) YES
  - b) NO

Signature of the lecturer

( )

Signature of the student

( )

3. Each practical which student is performing in the lab should have the following details :
  - a) Topic Detail
  - b) AIM
  - c) Algorithm
  - d) Source Code
  - e) Output
  - f) Viva questions
4. Project report should be added at last page.

## 5. LIST OF EXPERIMENTS (As prescribed by GGSIPU)

<b>Paper Code(s): CIC-258</b>	<b>L</b>	<b>P</b>	<b>C</b>
<b>Paper: Programming in Java Lab</b>	<b>-</b>	<b>2</b>	<b>1</b>

<b>Marking Scheme:</b> 1. Teachers Continuous Evaluation: 40 marks 2. Term end Theory Examinations: 60 marks
<b>Instructions:</b> 1. The course objectives and course outcomes are identical to that of (Programming in Java) as this is the practical component of the corresponding theory paper. 2. The practical list shall be notified by the teacher in the first week of the class commencement under intimation to the office of the Head of Department / Institution in which the paper is being offered from the list of practicals below. <u>Atleast 10 experiments must be performed by the students, they may be asked to do more. Atleast 5 experiments must be from the given list.</u>

	Page No.
1. Write a java program to implement stack and queue concept.(CO1,CO4)	16
2. Write a program to produce tokens from a given string.(CO1,CO2)	21
3. Write a java package to show dynamic polymorphism and interfaces.(CO1,CO2)	27
4. Write a Java program to show multithreaded producer and consumer application.(CO1,CO3)	36
5. Create a Customized Exception and also make use of all the 5 exception keywords.(CO1,CO3)	44
6. Convert the content of a given file into the uppercase content of the same file.(CO2,CO3)	49
7. Write a program in Java to sort the content of a given text file.(CO2,CO3,CO4)	52
8. Develop an Analog clock using Applet.(CO4)	55
9. Develop a Scientific calculator using Swings.(CO4)	60
10. Create an editor like MS-word using swings.(CO4)	65
11. Create a servlet that uses Cookies to store the number of times a user has visited your Servlet.(CO3,CO4)	68
12. Create a simple Java Bean having bound and constrained properties.(CO3,CO4)	71

## 6. Programming in Java (List of experiments beyond GGSIPU Syllabus)

1. Write a Program to accept a String as a command-line argument and print a Welcome message as given below:- "Welcome yourname". (CO1)
2. Program to find ASCII code of a character (CO1)
3. Write a Program to accept two integers as inputs and print their sum.(CO1)
4. Swapping two numbers using bitwise operator.(CO1)
5. Initialize two-character variables in a program and display the characters in alphabetical order.(CO1)
6. Write a program to receive a colour code from the user (an Alphabet).(CO1)
7. Write a program to print even numbers between 23 and 57. Each number should be printed in a separate row.(CO1)
8. Write a program to print \* in Floyd's format (using for and while loop).(CO1)
9. Write a Java program to find if the given number is palindrome or not.(CO1)
10. Initialize an integer array with ASCII values and print the corresponding character values in a single row.(CO1)
11. Write a program to reverse the elements of a given 2\*2 array. Four integer numbers need to be passed as Command-Line arguments.(CO1)
12. Using the concept of method overloading, write method for calculating the area of triangle, circle and rectangle.(CO2)
13. Create a class Box that uses a parameterized constructor to initialize the dimensions of a box. The dimensions of the Box are width, height, depth. The class should have a method that can return the volume of the box. Create an object of the Box class and test the functionalities.(CO2)
14. WAP to display the use of this keyword.(CO1,CO2)
15. Write a program that can count the number of instances created for the class.(CO1,CO2)
16. Java Program to get the cube of a given number using the static method.(CO1,CO2)
17. Write a program that can count the number of instances created for the class.(CO2)
18. Create a base class Fruit which has name ,taste and size as its attributes. A method called eat() is created which describes the name of the fruit and its taste. Inherit the same in 2 other class Apple and Orange and override the eat() method to represent each fruit taste. (Method overriding) (CO2)
19. Write a program to create a class named shape. It should contain 2 methods- draw() and erase() which should print "Drawing Shape" and "Erasing Shape" respectively. For this class we have three sub classes- Circle, Triangle and Square and each class override the parent class functions- draw () and erase (). The draw() method should print "Drawing Circle", "Drawing Triangle", "Drawing Square" respectively. The erase() method should print "Erasing Circle", "Erasing Triangle", "Erasing Square" respectively. Create objects of Circle, Triangle and Square in the following way and observe the polymorphic nature of the class by calling draw() and erase() method using each object. Shape c=new Circle(); Shape t=new Triangle(); Shape s=new Square(); (Polymorphism) (CO2)
20. Create a new class called Calculator with the following methods: A static method called powerInt(int num1,int num2) . This method should return num1 to the power num2.A static method called powerDouble(double num1,int num2). This method should return num1 to the power num2.Invoke both the methods and test the functionalities. (CO1,CO2)
21. Create a class named 'Animal' which includes methods like eat() and sleep().Create a child class of Animal named 'Bird' and override the parent class methods. Add a new method named fly().Create an instance of Animal class and invoke the eat and sleep methods using this object. Create an instance of Bird class and invoke the eat, sleep and fly methods using this object.(Inheritance) (CO2)
22. Write a program illustrating a super class variable a referencing as sub class object.(CO2)
23. Write a program illustrating all uses of super keywords. (CO2)

24. Create an abstract class shape. Let rectangle and triangle inherit this shape class. Add necessary functions. (CO2)
25. Create a package called test package; Define a class called foundation inside the test package; Inside the class, you need to define 4 integer variables; Var1 as private; Var2 as default; Var3 as protected; Var4 as public; Import this class and packages in another class. Try to access all 4 variables of the foundation class and see what variables are accessible and what are not accessible. (CO2)
26. Write an application that creates an 'interface' and implements it. (CO2)
27. Write a Program to take care of Number Format Exception if user enters values other than integer for calculating average marks of 2 students. The name of the students and marks in 3 subjects are taken from the user while executing the program. In the same Program write your own Exception classes to take care of Negative values and values out of range (i.e. other than in the range of 0-100) (CO2)
28. Write a program that accepts 2 integers a and b as input and finds the quotient of a/b. This program may generate an Arithmetic Exception. Use exception handling mechanisms to handle this exception. In the catch block, print the message as shown in the sample output. Also illustrate the use of finally block. Print the message "Inside finally block". Sample Input and Output 1: Enter the 2 numbers 5 2 The quotient of 5/2 = 2 Inside finally block Sample Input and Output 2: Enter the 2 numbers 5 0 DivideByZeroException caught Inside finally block. (CO2)
29. Write a program that takes as input the size of the array and the elements in the array. The program then asks the user to enter a particular index and prints the element at that index. Index starts from zero. This program may generate Array Index Out Of Bounds Exception or NumberFormatException. Use exception handling mechanisms to handle this exception. (CO2)
30. Write an application that executes two threads. One thread displays "A" every 1000 milliseconds and other displays "B" every 3000 milliseconds. Create the threads by extending the Thread class. (CO3)
31. Write an application that shows thread synchronization. (CO3)
32. Write an application that displays deadlock between threads. (CO3)
33. Write an application that shows thread priorities. (CO3)
34. Write an Applet that displays "Hello World" (Background color-black, text color-blue and your name in the status window.) (CO3)
35. Write a program that displays the life cycle of an Applet. (CO3)
36. Write an Applet displaying line, rectangle, rounded rectangle, filled rectangle, filled rounded rectangle, circle, ellipse, arc, filled arc and polygon, all in different colors. (CO3)
37. Write an Applet that displays a counter in the middle of applet. (CO3)
38. Write an Applet that displays a counter in the middle of applet. The counter starts from zero and keeps on incrementing after every second. (CO3)
39. Write an Applet that draws a dot at a random location in its display area every 200ms. Any existing dots are not erased. Therefore dots accumulate as the applet executes. (CO3)
40. Write an Applet that illustrates how to process mouse click, enter, exit, press and release events. The background color changes when the mouse is entered, clicked, pressed, released or exited. (CO3)
41. Write an Applet that displays your name whenever the mouse is clicked. (CO3)
42. Use adapter classes to write an Applet those changes to cyan while the mouse is being dragged. At all other times the applet should be white. (CO3)
43. Use inner classes to write an Applet those changes to cyan while the mouse is being dragged. At all other times the applet should be white. (CO3)
44. Use anonymous classes to write an Applet those changes to cyan while the mouse is being dragged. At all other times the applet should be white. (CO3)

45. Create a servlet that prints all the request headers it receives, along with their associated values.(CO3)
46. Write a servlet to show all the parameters sent to the servlet via either GET or POST.(CO3)
47. Create a servlet that recognizes visitor for the first time to a web application and responds by saying “Welcome, you are visiting for the first time”. When the page is visited for the second time ,it should say “Welcome Back”.(CO3)
48. Basic File handling program in java with reader/writer .(CO4)
49. Write a program that read from a file and write to file. (CO4)
50. Write RMI based client-server programs.(CO4)
51. Write programs of database connectivity using JDBC-ODBC drivers.(CO4)

## **PROJECTS TO BE ALLOTTED (Beyond the syllabus prescribed by G.G.S.I.P.U)**

Students will be divided into a group of four/five and projects are allotted to those groups. This project is to be submitted at the end of the semester along with a project report by the individual student.

List of projects given to the students is summarized as below:

- Dx Ball Game
- Moving ball with Java Script
- Checkerboard game with Java Script
- Auction Site (Bid Module)
- Flight Management System
- Global Defender Game
- Brick Game
- Rapid Roll Game
- Tic Tac Toe

Students can select project work of their own choice subject to the permission of concern faculty.

**NOTE:** The project is to be made in Java Language preferably.

## **7. INSTRUCTIONS FOR EACH LAB EXPERIMENT**

### **Java Programming**

#### **THIS SECTION COVERS:**

- 1. Console Based Programming**
- 2. OOP's Based Programming**
- 3. AWT and Event Handling**
- 4. Java DataBase Connectivity (JDBC)**
- 5. Socket Programming and RMI**
- 6. Java Native Interface (JNI)**
- 7. Collection Interface**

#### **JAVA PROGRAMMING ENVIRONMENT (AN INTRODUCTION)**

Anyone who is learning to program has to choose a programming environment that makes it possible to create and to run programs. Programming environments can be divided into two very different types: integrated development environments and command-line environments. All programming environments for Java require some text editing capability, a Java compiler, and a way to run applets and stand-alone applications. An integrated development environment, or IDE, is a graphical user interface program that integrates all these aspects of programming and probably others (such as a debugger, a visual interface builder, and project management). A command-line environment is just a collection of commands that can be typed in to edit files, compile source code, and run programs.

Command line environment is preferable for beginning programmers. IDEs can simplify the management of large numbers of files in a complex project, but they are themselves complex programs that add another level of complications to the already difficult task of learning the fundamentals of programming.

Java was developed at Sun Microsystems, Inc., and the primary source for information about Java is Sun's Java Web site, <http://java.sun.com/>. At this site, one can read documentation on-line and you can download documentation and software. The documentation includes the Java API reference and the Java tutorial.

The current version of Java on the Sun site is version 1.4. It is available for the Windows, Linux, and Solaris operating systems. One can download the "J2SE 1.4 SDK." This is the "Java 2 Platform Standard Edition Version 1.4 Software Development Kit." This package includes a Java compiler, a Java virtual machine that can be used to run Java programs, and all the standard Java packages. The JRE is the "Java Runtime Environment." It only includes the parts of the system that are needed to run Java programs. It does not have a compiler.

### **Integrated Development Environments**

There are sophisticated IDEs for Java programming that are available.

- Eclipse IDE -- An increasingly popular professional development environment that supports Java development, among other things. Eclipse is itself written in Java. It is available from <http://www.eclipse.org/>.
- NetBeans IDE -- A pure Java IDE that should run on any system with Java 1.7 or later. NetBeans is a free, "open source" program. It is essentially the open source version of the next IDE. It can be downloaded from [www.netbeans.org](http://www.netbeans.org).
- Sun ONE Studio 4 for Java, Community Edition, for Linux, Solaris, Windows 2000 to all versions till 2010, Windows NT, and Windows 98SE. This was formerly known as "Forte for Java", and it might be referred under that name. Again, it requires a lot of resources, with a 256 MB memory recommendation. Main site currently at <http://www.sun.com/software/sundev/jde/index.html>. It is available from



there and on the J2SE download page,

<http://java.sun.com/j2se/1.7/download.html>. The Community Edition is the free version.

- Borland JBuilder Personal Edition, for Linux, Solaris, MacOS X, Windows 2000, Windows XP, and Windows NT. Requires a lot of disk space & memory (256 MB memory recommended). Company Web page at <http://www.borland.com>. Jbuilder site at <http://www.borland.com/jbuilder/index.html>. The Personal Edition, which is free, has more than enough features for most programmers.
- BlueJ is a Java IDE written in Java that is meant particularly for educational use. It is available from <http://www.bluej.org/>.
- JCreator, for Windows. It looks like a nice lighter-weight IDE that works on top of Sun's SDK. There is a free version, as well as a shareware version. It is available at <http://www.jcreator.com>.

There are other products similar to JCreator, for Windows and for other operating systems.

## **Text Editors**

To use a command-line environment for programming good text editor is needed. A programmer's text editor is a very different thing from a word processor. Most important, it saves work in plain text files and it doesn't insert extra carriage returns beyond the ones you actually type. A good programmer's text editor will do a lot more than this. Here are some features to look for:

- Syntax coloring. Shows comments, strings, keywords, etc., in different colors to make the program easier to read and to help you find certain kinds of errors.
- Function menu. A pop-up menu that lists the functions in your source code. Selecting a function from this will take you directly to that function in the code.
- Auto-indentation. When you indent one line, the editor will indent following lines to match, since that's what you want more often than not when you are typing a program.

- Parenthesis matching. After typing a closing parenthesis the cursor jumps back to the matching parenthesis momentarily so one can see where it is. Alternatively, there might be a command that will highlight all the text between matching parentheses. The same thing works for brackets and braces.
- Indent Block and Unindent Block commands. These commands apply to a highlighted block of text. They will insert or remove spaces at the beginning of each line to increase or decrease the indentation level of that block of text. When you make changes in your program, these commands can help you keep the indentation in line with the structure of the program.
- Control of tabs. Don't use tab characters for indentation. A good editor can be configured to insert multiple space characters when tab key is pressed.

There are many free text editors that have some or all of these features. **Jedit**, a programmer's text editor written entirely in Java. It requires Java 1.3 or better. It has many features listed above, and there are plug-ins available to add additional features. Since it is written in pure Java, it can be used on any operating system that supports Java 1.3. In addition to being a nice text editor, it shows what can be done with the Swing GUI. Jedit is free and can be downloaded from <http://www.jedit.org>.

On Linux, use **nedit**. It has all the above features, except a function menu. Under Linux, it is likely that *nedit* is included in distribution, although it may not have been installed by default. It can be downloaded from <http://www.nedit.org/> and is available for many UNIX platforms in addition to Linux. Features such as syntax coloring and auto-indentation are not turned on by default. One can configure them in the Options menu. Use the "Save Options" command to make the configuration permanent. Of course, as alternatives to nedit, the Gnome and KDE desktops for Linux have their own text editors.

### Using the Java SDK

After installing Sun's Software Development Kit for Java, one can use the commands "javac", "java", and "appletviewer" for compiling and running Java programs and applets. These commands must be on the "path" where the operating system searches for commands.

Make a directory to hold Java programs. Create program with a text editor, or copy the program to be compiled into program directory

If program contains more than a few errors, most of them will scroll out of the window. In Linux and UNIX, a command window usually has a scroll bar that can be used to review the errors. In Windows 2000 to 2010/NT/XP (but **not** Windows 95/98), one can save the errors in a file which can be viewed later in a text editor.

The command in Windows is `javac SourceFile.java >& errors.txt`

The ">& errors.txt" redirects the output from the compiler to the file, instead of to the DOS window. It is possible to compile all the Java files in a directory at one time. Use the command `"javac *.java"`.

After compiled class files are made, run application or applet. For running a stand-alone application -- one that has a `main ()` routine -- use the "java" command from the SDK to run the application. If the class file that contains the `main ()` routine is named `Main. class`, then run the program with the command: `java Main`

## **SAMPLE CONSOLE BASED PROGRAM**

The following program, For Demo, uses the general form of the `for` statement to print the numbers 1 through 10 to standard output:

### **Steps to write JAVA Program**

1. Create JAVA file by text editor (eg vi editor).
2. Write the program as per JAVA syntax.
3. Save the file with `.java` extension.
4. Compile the file with JAVA compiler (`javac filename.java`) and create class file.
5. Run the class file with JAVA interpreter (`java classname.class`) and check the output.

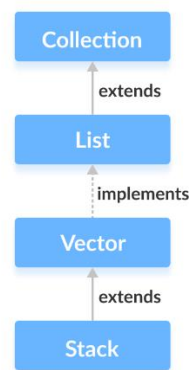
## EXPERIMENT-1

**Aim:-**Write a java program to implement stack and queue subject.

**Theory: -**

**Stacks** are linear data structures that follow the LIFO principle, that is, last in first out. This simply means that insertion of a new node and removal of a node takes place from the end. This is referred to as the top in stack terminology. The operations are referred to as push and pop for insertion and deletion respectively.

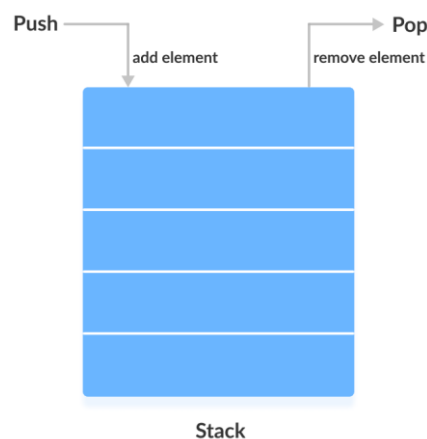
The Java collections framework has a class named Stack that provides the functionality of the stack data structure.



The Stack class extends the Vector class.

### Stack Implementation

In stack, elements are stored and accessed in Last In First Out manner. That is, elements are added to the top of the stack and removed from the top of the stack.



### Creating a Stack

In order to create a stack, we must import the java.util.Stack package first. Once we import the package, here is how we can create a stack in Java.

```
Stack<Type> stacks = new Stack<>();
```

Here, Type indicates the stack's type. For example-

```
// Create Integer type stack
Stack<Integer> stacks = new Stack<>();

// Create String type stack
Stack<String> stacks = new Stack<>();
```

## Stack Methods

Since Stack extends the Vector class, it inherits all the methods Vector. To learn about different Vector methods, visit [Java Vector Class](#).

Besides these methods, the Stack class includes 5 more methods that distinguish it from Vector.

**1.push() Method** - To add an element to the top of the stack, we use the push() method. For example,

```
import java.util.Stack;
class Main {
    public static void main(String[] args)
    {
        Stack<String> animals= new Stack<>();
        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
        animals.push("Cat");
        System.out.println("Stack: " + animals);
    }
}
```

**2. pop() Method** - To remove an element from the top of the stack, we use the pop() method. For example,

```
import java.util.Stack;
class Main {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();
        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
        animals.push("Cat");
        System.out.println("Initial Stack: " + animals);
        // Remove element stacks
        String element = animals.pop();
        System.out.println("Removed Element: " + element);
    }
}
```

**3. peek() Method**- The peek() method returns an object from the top of the stack. For example,

```
import java.util.Stack;
class Main {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();
        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
```

```

        animals.push("Cat");
        System.out.println("Stack: " + animals);
        // Access element from the top
        String element = animals.peek();
        System.out.println("Element at top: " + element);
    }
}

```

**4.search() Method-** To search an element in the stack, we use the search() method. It returns the position of the element from the top of the stack. For example,

```

import java.util.Stack;
class Main {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();
        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
        animals.push("Cat");
        System.out.println("Stack: " + animals);
        // Search an element
        int position = animals.search("Horse");
        System.out.println("Position of Horse: " + position);
    }
}

```

**5.empty() Method** -To check whether a stack is empty or not, we use the empty() method. For example,

```

import java.util.Stack;
class Main {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();
        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
        animals.push("Cat");
        System.out.println("Stack: " + animals);
        // Check if stack is empty
        boolean result = animals.empty();
        System.out.println("Is the stack empty? " + result);
    }
}

```

**Queues**, follow the FIFO principle, which is first in first out. They represent an actual queue in real-life where people who enter first are serviced first and so on. Insertion operation is known as enqueue and deletion is known as dequeue. Queues are also extremely popular in networking as well as system design. They are widely used for controlling access to resources.

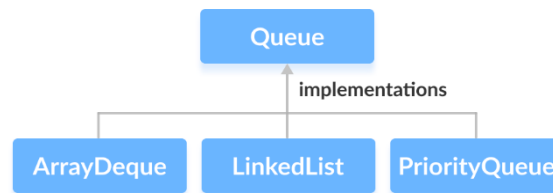
The Queue interface of the Java collections framework provides the functionality of the queue data structure. It extends the Collection interface.

**Classes that Implement Queue** :- Since the Queue is an interface, we cannot provide the direct implementation of it. In order to use the functionalities of Queue, we need to use classes that implement it:

ArrayDeque

LinkedList

PriorityQueue



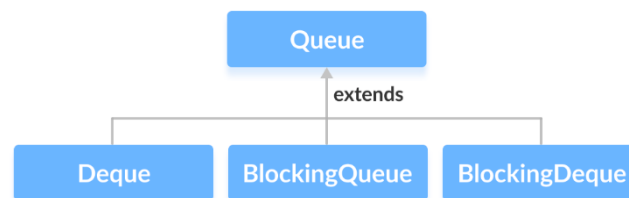
### Interfaces that extend Queue

The Queue interface is also extended by various subinterfaces:

Deque

BlockingQueue

BlockingDeque



### Working of Queue Data Structure

In queues, elements are stored and accessed in First In, First Out manner. That is, elements are added from the behind and removed from the front.



### How to use Queue?

In Java, we must import java.util.Queue package in order to use Queue.

// LinkedList implementation of Queue

Queue<String> animal1 = new LinkedList<>();

// Array implementation of Queue

Queue<String> animal2 = new ArrayDeque<>();

// Priority Queue implementation of Queue

Queue<String> animal 3 = new PriorityQueue<>();

Here, we have created objects animal1, animal2 and animal3 of classes LinkedList, ArrayDeque and PriorityQueue respectively. These objects can use the functionalities of the Queue interface.

### Methods of Queue :-

The Queue interface includes all the methods of the Collection interface. It is because Collection is the super interface of Queue.

Some of the commonly used methods of the Queue interface are:

1. `add()` - Inserts the specified element into the queue. If the task is successful, `add()` returns true, if not it throws an exception.
2. `offer()` - Inserts the specified element into the queue. If the task is successful, `offer()` returns true, if not it returns false.
3. `element()` - Returns the head of the queue. Throws an exception if the queue is empty.
4. `peek()` - Returns the head of the queue. Returns null if the queue is empty.
5. `remove()` - Returns and removes the head of the queue. Throws an exception if the queue is empty.
6. `poll()` - Returns and removes the head of the queue. Returns null if the queue is empty.

## **PRACTICE QUESTIONS**

- Q1. Design a stack that returns the minimum element in constant time.
- Q2. Reverse a string using a stack data structure.
- Q3. Design a stack that returns a minimum element without using an auxiliary stack.
- Q4. Reverse a stack using recursion.

## **Viva-Questions :-**

- Q1. Explain why Stack is a recursive data structure ?
- Q2. Why and when should Stack or Queue data structures should be chosen instead of Arrays/Lists?
- Q3. Compare Array-Based vs List-Based implementation of Queues.
- Q4. What are benefits of Circular Queue?
- Q5. How Iterative InOrder traversal and PostOrder traversal differ in implementation using Stack?



## EXPERIMENT- 2

**Aim :-** Write a program to produce tokens from a given string .

**Theory :-**

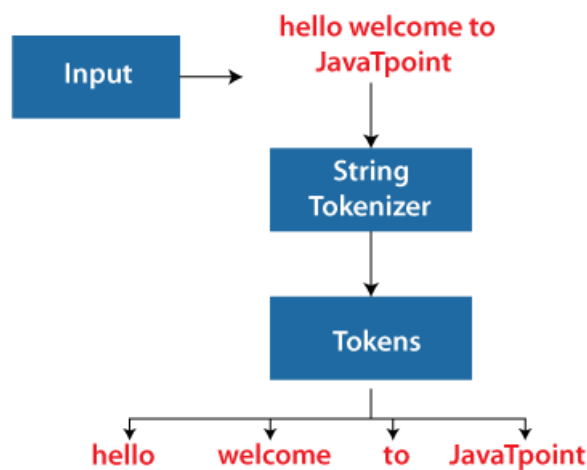
### **StringTokenizer:**

The StringTokenizer class helps us split Strings into multiple tokens.

StreamTokenizer provides similar functionality but the tokenization method of StringTokenizer is much simpler than the one used by the StreamTokenizer class. Methods of StringTokenizer do not distinguish among identifiers, numbers, and quoted strings, nor recognize and skip comments.

The java.util.StringTokenizer class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java. It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

The set of delimiters (the characters that separate tokens) may be specified either at the creation time or on a per-token basis.



### **Example of String Tokenizer class in Java**

#### **Constructors of the StringTokenizer Class :-**

There are 3 constructors defined in the StringTokenizer class.

Constructor	Description
StringTokenizer(String str)	It creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	It creates StringTokenizer with specified string and delimiter.
StringTokenizer(String str, String delim, boolean returnValue)	It creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

## Methods of the StringTokenizer Class

The six useful methods of the StringTokenizer class are as follows:

Methods	Description
boolean hasMoreTokens()	It checks if there is more tokens available.
String nextToken()	It returns the next token from the StringTokenizer object.
String nextToken(String delim)	It returns the next token based on the delimiter.
boolean hasMoreElements()	It is the same as hasMoreTokens() method.
Object nextElement()	It is the same as nextToken() but its return type is Object.
int countTokens()	It returns the total number of tokens.

### Example of StringTokenizer Class

Let's see an example of the StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

Simple.java

```
import java.util.StringTokenizer;

public class Simple{

    public static void main(String args[]){


        StringTokenizer st = new StringTokenizer("my name is khan"," ");

        while (st.hasMoreTokens()) {

            System.out.println(st.nextToken());

        } } }
```

### Output:



```
my
name
is
khan
```

The above Java code, demonstrates the use of StringTokenizer class and its methods hasMoreTokens() and nextToken().

### Example of nextToken(String delim) method of the StringTokenizer class

Test.java

```
import java.util.*;

public class Test {

    public static void main(String[] args)

    {
```

```
StringTokenizer st = new StringTokenizer("my,name,is,khan");

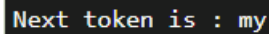
// printing next token

System.out.println("Next token is : " + st.nextToken(", "));

}

}
```

### Output:



```
Next token is : my
```

Note: The StringTokenizer class is deprecated now. It is recommended to use the split() method of the String class or the Pattern class that belongs to the java.util.regex package.

### Example of hasMoreTokens() method of the StringTokenizer class

This method returns true if more tokens are available in the tokenizer String otherwise returns false.

#### StringTokenizer1.java

```
import java.util.StringTokenizer;

public class StringTokenizer1
{
    /* Driver Code */

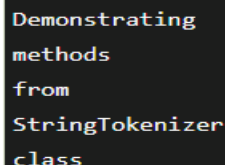
    public static void main(String args[])
    {
        /* StringTokenizer object */

        StringTokenizer st = new StringTokenizer("Demonstrating methods from StringTokenizer class", " ");

        /* Checks if the String has any more tokens */

        while (st.hasMoreTokens())
        {
            System.out.println(st.nextToken());
        }
    }
}
```

### Output:-



```
Demonstrating
methods
from
StringTokenizer
class
```

The above Java program shows the use of two methods hasMoreTokens() and nextToken() of StringTokenizer class.

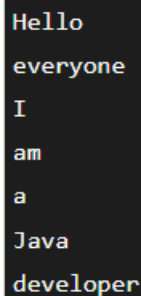
### Example of hasMoreElements() method of the StringTokenizer class

This method returns the same value as `hasMoreTokens()` method of `StringTokenizer` class. The only difference is this class can implement the `Enumeration` interface.

`StringTokenizer2.java`

```
import java.util.StringTokenizer;
public class StringTokenizer2
{
    public static void main(String args[])
    {
        StringTokenizer st = new StringTokenizer("Hello everyone I am a Java developer", " ");
        while (st.hasMoreElements())
        {
            System.out.println(st.nextToken());
        }
    }
}
```

Output:



```
Hello
everyone
I
am
a
Java
developer
```

The above code demonstrates the use of `hasMoreElements()` method.

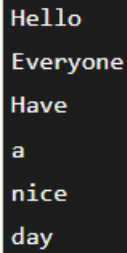
#### **Example of `nextElement()` method of the `StringTokenizer` class**

`nextElement()` returns the next token object in the tokenizer `String`. It can implement `Enumeration` interface.

`StringTokenizer3.java`

```
import java.util.StringTokenizer;
public class StringTokenizer3
{
    /* Driver Code */
    public static void main(String args[])
    {
        /* StringTokenizer object */
        StringTokenizer st = new StringTokenizer("Hello Everyone Have a nice day", " ");
        /* Checks if the String has any more tokens */
        while (st.hasMoreTokens())
        {
            /* Prints the elements from the String */
            System.out.println(st.nextElement());
        }
    }
}
```

## OUTPUT



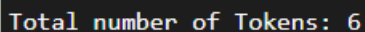
```
Hello
Everyone
Have
a
nice
day
```

The above code demonstrates the use of `nextElement()` method.  
Example of `countTokens()` method of the `StringTokenizer` class  
This method calculates the number of tokens present in the tokenizer String.

### **StringTokenizer4.java**

```
import java.util.StringTokenizer;
public class StringTokenizer3
{
    /* Driver Code */
    public static void main(String args[])
    {
        /* StringTokenizer object */
        StringTokenizer st = new StringTokenizer("Hello Everyone Have a nice day", " ");
        /* Prints the number of tokens present in the String */
        System.out.println("Total number of Tokens: "+st.countTokens());
    }
}
```

## OUTPUT



```
Total number of Tokens: 6
```

The above Java code demonstrates the `countTokens()` method of `StringTokenizer()` class.

## PRACTICE QUESTIONS

Q1. Java Program to check whether two Strings are anagram or not.

Q2. Java program to find the percentage of uppercase, lowercase, digits and special characters in a String

Q3. Write a java program reverse tOGGLE each word in string?

### **Viva Questions :-**

Q1. What is the difference between `equals()` method and `==` operator?

Q2. Is String class final?

Q3. What is the difference between String and StringBuffer in java?

Q4. What is the difference between StringBuffer and StringBuilder in java?

Q5. What does intern() method in java

## EXPERIMENT-3

**Aim:-**Write a java package to show dynamic polymorphism and interfaces.

**Theory:-** In Java, polymorphism is a concept of object-oriented programming that allows us to perform a single action in different forms. The word polymorphism is a combination of two words i.e. ploy and morphs. The word poly means many and morphs means different forms. In short, a mechanism by which we can perform a single action in different ways.

There are two types of polymorphism in Java:

1. Static Polymorphism (Compile Time Polymorphism)
2. Dynamic Polymorphism (Run Time Polymorphism)

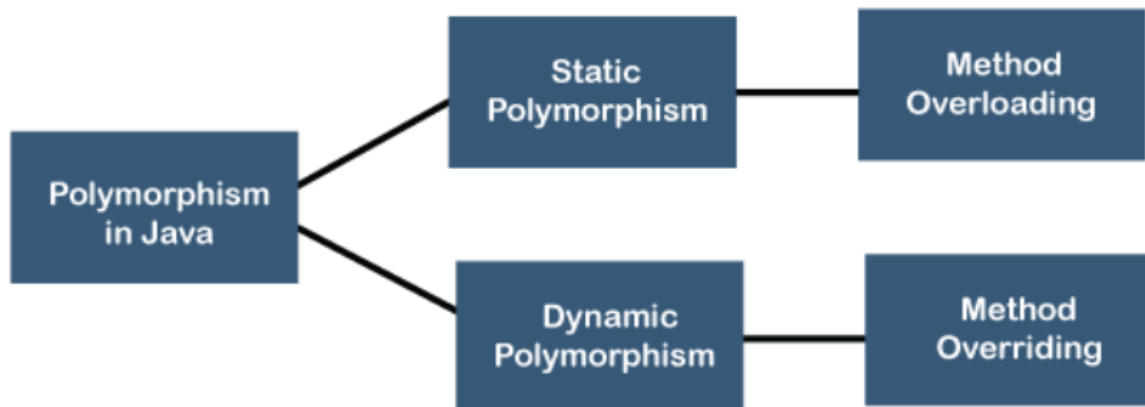


Fig 3.1 Polymorphism types in Java

**Dynamic Polymorphism** - Dynamic polymorphism is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time. It is also known as runtime polymorphism or dynamic method dispatch. Dynamic polymorphism can be achieved by using the method overriding.

In this process, an overridden method is called through a reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Method overriding is one of the ways in which Java supports Runtime Polymorphism.

### Properties of Dynamic Polymorphism

1. It decides which method is to execute at runtime.
2. It can be achieved through dynamic binding.
3. It happens between different classes.
4. It is required where a subclass object is assigned to a super-class object for dynamic polymorphism.
5. Inheritance involved in dynamic polymorphism.

### Method Overriding

It provides a specific implementation to a method that is already present in the parent class. It is used to achieve run-time polymorphism. Remember that, it is not possible to override the static method. Hence, we cannot override the main() method also because it is a static method.

## Rules for Method Overriding

The name of the method must be the same as the name of the parent class method.

The number of parameters and the types of parameters must be the same as in the parent class.

There must exist an IS-A relationship (inheritance).

We call an overridden method through a reference of the parent class. The type of object decides which method is to be executed and it is decided by the JVM at runtime.

**Dynamic Method Dispatch** is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.

At run-time, it depends on the type of object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

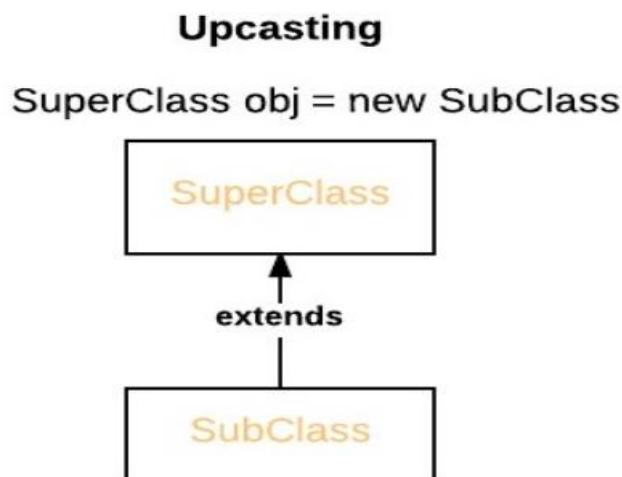


Fig.3.2- Upcasting in Java

Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

Here is an example that illustrates Dynamic Method Dispatch:

**// A Java program to illustrate Dynamic Method Dispatch using hierarchical inheritance**

```
class A
{
    void m1()
    {
        System.out.println("Inside A's m1 method");
    }
}
```



```

}
class B extends A
{ // overriding m1()
    void m1()
    {
        System.out.println("Inside B's m1 method");
    }
}
class C extends A
{
    // overriding m1()
    void m1()
    {
        System.out.println("Inside C's m1 method");
    }
}
// Driver class
class Dispatch
{
    public static void main(String args[])
    {
        // object of type A
        A a = new A();
        // object of type B
        B b = new B();
        // object of type C
        C c = new C();
        // obtain a reference of type A
        A ref;
        // ref refers to an A object
        ref = a;
        // calling A's version of m1()
        ref.m1();
    }
}

```

```

// now ref refers to a B object
ref = b;
// calling B's version of m1()
ref.m1();
// now ref refers to a C object
ref = c;
// calling C's version of m1()
ref.m1();
}
}

```

Explanation :

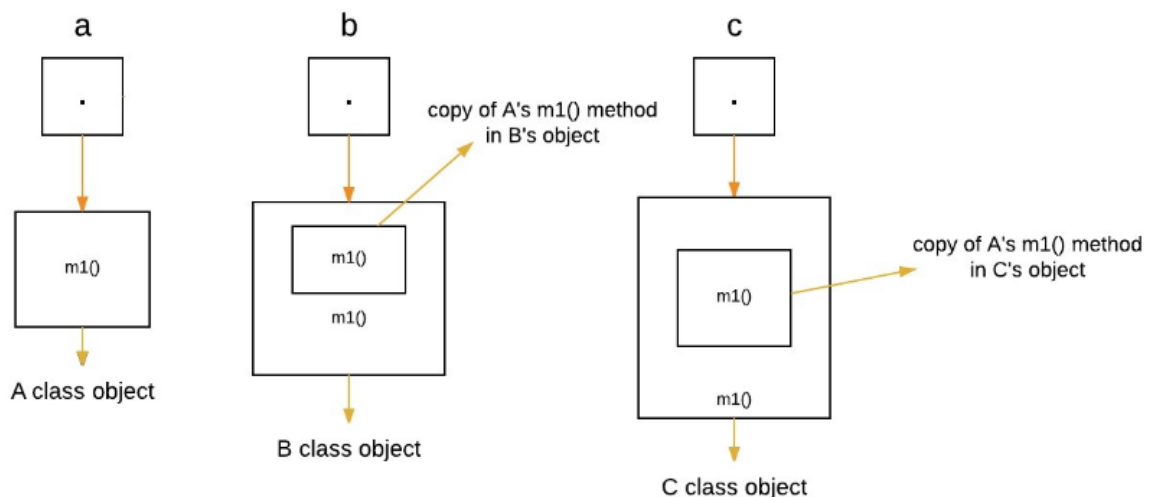
The above program creates one superclass called A and it's two subclasses B and C. These subclasses overrides m1() method.

Inside the main() method in Dispatch class, initially objects of type A, B, and C are declared.

```
A a = new A(); // object of type A
```

```
B b = new B(); // object of type B
```

```
C c = new C(); // object of type C
```



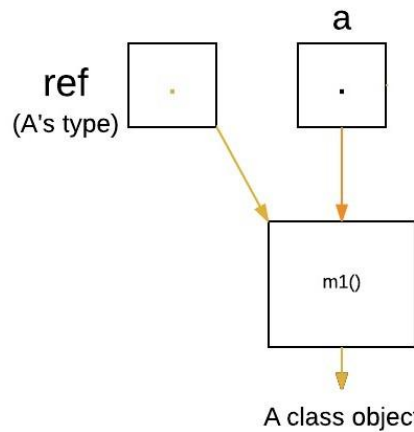
Now a reference of type A, called ref, is also declared, initially it will point to null.

```
A ref; // obtain a reference of type A
```

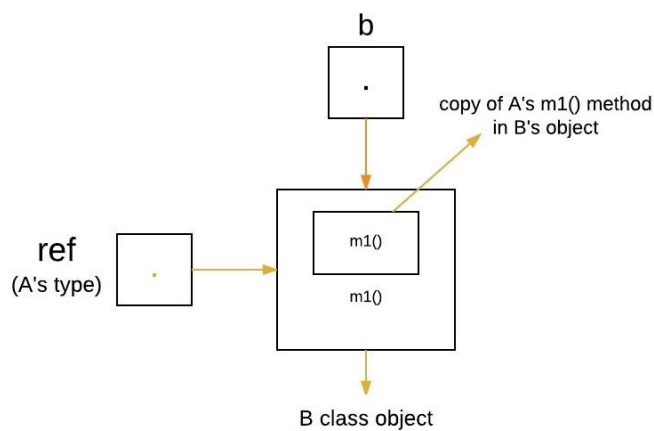
Now we are assigning a reference to each **type of object** (either A's or B's or C's) to *ref*, one-by-one, and uses that reference to invoke m1( ). As the output shows, the version of m1( ) executed is determined by the type of object being referred to at the time of the call.

```
ref = a; // r refers to an A object
```

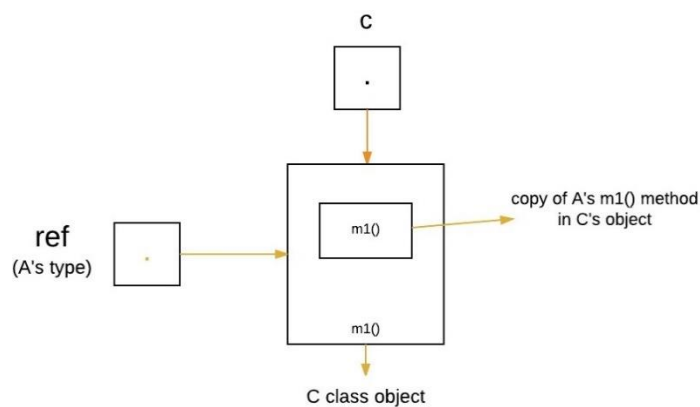
```
ref.m1(); // calling A's version of m1()
```



`ref = b; // now r refers to a B object`  
`ref.m1(); // calling B's version of m1()`



`ref = c; // now r refers to a C object`  
`ref.m1(); // calling C's version of m1()`



In the following example, we have created two classes named Sample and Demo. The Sample class is a parent class and the Demo class is a child or derived class. The child class is overriding the display() method of the parent class. We have assigned the child class object to the parent class reference. So, in order to determine which method would be called, the type of object would be decided by the JVM at run-time. It is the type of object that determines which version of the method would be called (not the type of reference).

**Demo.java**

```

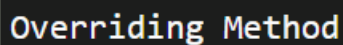
//parent class
class Sample
{
//method of the parent class
public void display()
{
System.out.println("Overridden Method");
}
}

//derived or child class
public class Demo extends Sample
{
//method of child class
public void display()
{
System.out.println("Overriding Method");
}

public static void main(String args[])
{
//assigning a child class object to parent class reference
Sample obj = new Demo();
//invoking display() method
obj.display();
}
}

```

### Output:



```
Overriding Method
```

### Example 3- Example of Method Overriding

```
public class DynamicPolymorphismExample
```

```

{
public static void main(String args[])
{
//assigning a child class object to a parent class reference
Fruits fruits = new Mango();
//invoking the method
fruits.color();
}
}
//parent class
class Fruits
{
public void color()
{
System.out.println("Parent class method is invoked.");
}
}
//derived or child class that extends the parent class
class Mango extends Fruits
{
//overrides the color() method of the parent class
@Override
public void color()
{
System.out.println("The child class method is invoked.");
}
}

```

### Output:

```
The child class method is invoked.
```

In the above example, we have used an annotation `@Override`. It indicates that the child class method is over-writing its base class method.

## INTERFACES

Interfaces are very similar to classes. They have variables and methods but the interfaces allow only abstract methods(that don't contain the body of the methods).

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the **IS-A relationship**.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have default and static methods in an interface.

Since Java 9, we can have private methods in an interface.

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>
{
    // declare constant fields

    // declare methods that abstract

    // by default.
}
```

## PRACTICE QUESTIONS-

Q1. Write a program to show interface inheritance .

Q2. Write a program to showcase the implementation of super keyword .

## VIVA QUESTIONS

Q1. Why use Java interface?

Q2. How static binding and dynamic binding differ ?

Q3. Can we override a super class method without throws clause as a method with throws clause in the sub class?

Q4. How do you refer super class version of overridden method in the sub class?

Q5. Can we override private methods?

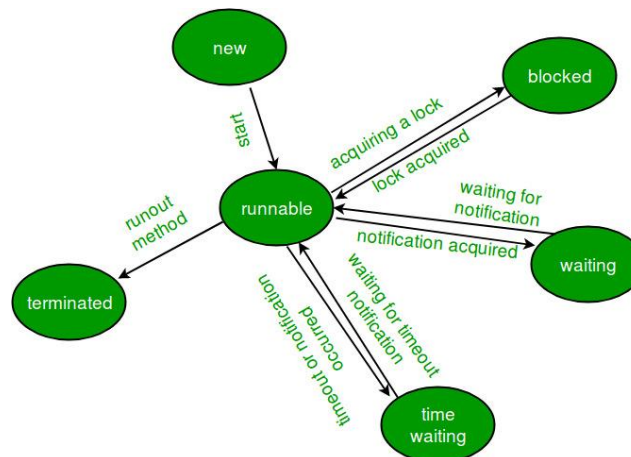
## EXPERIMENT-4

**Aim:-** Write a Java program to show multithreaded producer and consumer application.

### Theory :-

A thread in Java is the direction or path that is taken while a program is being executed. Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program's execution. At this point, when the main thread is provided, the main() method is invoked by the main thread. A thread in Java at any point of time exists in any one of the following states. A thread lies only in one of the states at any instant.

The diagram shown below represents various states of a thread at any instant in time.



### Life Cycle of a thread

**1.New Thread:** When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn't started to execute.

**Declaration:** `public static final Thread.State NEW`

**2.Runnable State:** A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run.

A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lie in a runnable state.

**Declaration:** `public static final Thread.State RUNNABLE`

**Description:** Thread state for a runnable thread. A thread in the runnable state is executing in the Java virtual machine but it may be waiting for other resources from the operating system such as a processor.

**3.Blocked/Waiting state:** When a thread is temporarily inactive, then it's in one of the following states:

- a. Blocked
- b. Waiting

**Declaration:** `public static final Thread.State BLOCKED`



Description: Thread state for a thread blocked waiting for a monitor lock. A thread in the blocked state is waiting for a monitor lock to enter a synchronized block/method or reenter a synchronized block/method after calling `Object.wait()`.

**Declaration: `public static final Thread.State WAITING`**

Description: Thread state for a waiting thread. A thread is in the waiting state due to calling one of the following methods:

`Object.wait` with no timeout

`Thread.join` with no timeout

`LockSupport.park`

**4.Timed Waiting:** A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls `sleep` or a conditional wait, it is moved to a timed waiting state.

**Declaration: `public static final Thread.State TIMED_WAITING`**

Description: Thread state for a waiting thread with a specified waiting time. A thread is in the timed waiting state due to calling one of the following methods with a specified positive waiting time:

`Thread.sleep`

`Object.wait` with timeout

`Thread.join` with timeout

`LockSupport.parkNanos`

`LockSupport.parkUntil`

**5.Terminated State:** A thread terminates because of either of the following reasons:

- a. Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
- b. Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

**Declaration: `public static final Thread.State TERMINATED`**

Description: Thread state for a terminated thread. The thread has completed execution.

**MULTITHREADING** - Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms:

1. Extending the Thread class
2. Implementing the Runnable Interface

**Thread creation by extending the Thread class**

We create a class that extends the `java.lang.Thread` class. This class overrides the `run()` method available in the Thread class. A thread begins its life inside `run()` method. We create an object of our new class

and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

// Java code for thread creation by extending

// the Thread class

```
class MultithreadingDemo extends Thread {  
    public void run()  
    {  
        try {  
            // Displaying the thread that is running  
            System.out.println(  
                "Thread " + Thread.currentThread().getId()  
                + " is running");  
        }  
        catch (Exception e) {  
            // Throwing an exception  
            System.out.println("Exception is caught");  
        }  
    }  
}  
  
// Main Class  
public class Multithread {  
    public static void main(String[] args)  
    {  
        int n = 8; // Number of threads  
        for (int i = 0; i < n; i++) {  
            MultithreadingDemo object  
                = new MultithreadingDemo();  
            object.start();  
        }  
    }  
}
```

## Thread creation by implementing the Runnable Interface

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

/ Java code for thread creation by implementing

// the Runnable Interface

```
class MultithreadingDemo implements Runnable {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}

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

## PRODUCER CONSUMER PROBLEM:

In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

The producer's job is to generate data, put it into the buffer, and start again.

At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

### Problem

To make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

### Solution

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

### Implementation of Producer Consumer Class

1. A LinkedList list – to store list of jobs in queue.
2. A Variable Capacity – to check for if the list is full or not
3. A mechanism to control the insertion and extraction from this list so that we do not insert into list if it is full or remove from it if it is empty.

### // Java program to implement solution of producer consumer problem.

```
import java.util.LinkedList;
public class Threadexample
{
    public static void main(String[] args)
        throws InterruptedException
    {
        // Object of a class that has both produce()
        // and consume() methods
        final PC pc = new PC();

        // Create producer thread
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                try {
                    pc.produce();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        // Create consumer thread
```

```

Thread t2 = new Thread(new Runnable() {
    @Override
    public void run()
    {
        try {
            pc.consume();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
});

// Start both threads
t1.start();
t2.start();

// t1 finishes before t2
t1.join();
t2.join();
}

// This class has a list, producer (adds items to list and consumer (removes items).

public static class PC {

    // Create a list shared by producer and consumer
    // Size of list is 2.
    LinkedList<Integer> list = new LinkedList<>();
    int capacity = 2;
    // Function called by producer thread
    public void produce() throws InterruptedException
    {
        int value = 0;
        while (true) {
            synchronized (this)
            {
                // producer thread waits while list is full

                while (list.size() == capacity)
                    wait();

                System.out.println("Producer produced-"
                                   + value);

                // to insert the jobs in the list
                list.add(value++);

                // notifies the consumer thread that now it can start consuming

                notify();

                // makes the working of program easier to understand

                Thread.sleep(1000);
            }
        }
    }
}

```

```

    }
}

// Function called by consumer thread
public void consume() throws InterruptedException
{
    while (true) {
        synchronized (this)
        {
            // consumer thread waits while list
            // is empty
            while (list.size() == 0)
                wait();

            // to retrieve the first job in the list
            int val = list.removeFirst();

            System.out.println("Consumer consumed-"
                               + val);

            // Wake up producer thread
            notify();

            // and sleep
            Thread.sleep(1000);
        }
    }
}
}
}

```

### Important Points

1. In **PC class** (A class that has both produce and consume methods), a linked list of jobs and a capacity of the list is added to check that producer does not produce if the list is full.
2. In **Producer class**, the value is initialized as 0.
  - Also, we have an infinite outer loop to insert values in the list. Inside this loop, we have a synchronized block so that only a producer or a consumer thread runs at a time.
  - An inner loop is there before adding the jobs to list that checks if the job list is full, the producer thread gives up the intrinsic lock on PC and goes on the waiting state.
  - If the list is empty, the control passes to below the loop and it adds a value in the list.
3. In the **Consumer class**, we again have an infinite loop to extract a value from the list.
  - Inside, we also have an inner loop which checks if the list is empty.
  - If it is empty then we make the consumer thread give up the lock on PC and passes the control to producer thread for producing more jobs.
  - If the list is not empty, we go round the loop and removes an item from the list.
4. In both the methods, we use notify at the end of all statements. The reason is simple, once you have something in list, you can have the consumer thread consume it, or if you have consumed something, you can have the producer produce something.
5. sleep() at the end of both methods just make the output of program run in step wise manner and not display everything all at once so that you can see what actually is happening in the program.

### **PRACTICE QUESTIONS-**

Q1. Program to creating multiple thread

Q2. Program to set priorities of thread

### **Viva Questions**

Q1. Differentiate between process and thread?

Q2. What is the purpose of wait() method in Java?

Q3. Why must wait() method be called from the synchronized block?

Q4. What is the difference between wait() and sleep() method?

Q5. Can we call the run() method instead of start()?

Q6. How Thread Class is different from Runnable Interface ?

Q7.What about the daemon threads?

Q8.Can we make the user thread as daemon thread if the thread is started?

## EXPERIMENT-5

**Aim: - Create a Customized Exception and also make use of all the 5 exception keywords.**

### 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. In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

### Types of Java Exceptions:

There are mainly two types of exceptions: Checked and Unchecked.

#### 1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

#### 2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 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.

**JAVA CUSTOMIZED EXCEPTION:** In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need. Using the custom exception, we can have your own exception and message. Here, we have



passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created.

### Why use custom exceptions?

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

Following are few of the reasons to use custom exceptions:

1. To catch and provide specific treatment to a subset of existing Java exceptions.
2. 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 custom exception, we need to extend Exception class that belongs to java.lang package.

Consider the following example, where we create a custom exception named WrongFileNameException:

```
public class WrongFileNameException extends Exception
{
    public WrongFileNameException(String errorMessage)
    {
        super(errorMessage);
    }
}
```

This **demo program** covers all **5 keywords** related to **Java Exception handling**

package in.bench.resources.exception.handling;

```
public class DemoOnTryCatchFinallyThrowThrows
{
    // main() method - start of JVM execution
    public static void main(String[] args)
    {
        try {
            // call division() method
            String welcomeMessage = welcomeMessage("SJ");
            // print to console
            System.out.println("The returned welcome message : "+ welcomeMessage);
        }
    }
}
```

```

        catch (NullPointerException npex){
            System.out.println("Exception handled : "+ npex.toString());
        }
        finally {
            System.out.println("Rest of the clean-up code here");
        }
    }
}

// division method returning quotient
public static String welcomeMessage(String name)
throws NullPointerException {
    if(name == null) {
        // explicitly throwing Null Pointer Error using throw keyword
        throw new NullPointerException(
            "Invoke method with VALID name");
    }

    // performing String concatenation
    String welcomeMsg = "Welcome " + name;

    // return concatenated string value
    return welcomeMsg;
}
}

```

**Java Custom Exception:** In the following code, constructor of InvalidAgeException takes a string as an argument. This string is passed to constructor of parent class Exception using the super() method. Also the constructor of Exception class can be called without using a parameter and calling super() method is not mandatory.

TestCustomException1.java

```

// class representing custom exception
class InvalidAgeException extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
    }
}

```

```

        super(str);
    }
}

// class that uses custom exception InvalidAgeException
public class TestCustomException1
{
    // method to check the age
    static void validate (int age) throws InvalidAgeException {
        if(age < 18){
            // throw an object of user defined exception
            throw new InvalidAgeException("age is not valid to vote");
        }
        else {
            System.out.println("welcome to vote");
        }
    }
    // main method
    public static void main(String args[])
    {
        try
        {
            // calling the method
            validate(13);
        }
        catch (InvalidAgeException ex)
        {
            System.out.println("Caught the exception");
            // printing the message from InvalidAgeException object
            System.out.println("Exception occurred: " + ex);
        }
        System.out.println("rest of the code...");
    } }

```

## PRACTICE QUESTIONS

Q1. Write a program that accepts 2 integers a and b as input and finds the quotient of a/b. This program may generate an Arithmetic Exception. Use exception handling mechanisms to handle this exception. In the catch block, print the message as shown in the sample output. Also illustrate the use of finally block. Print the message "Inside finally block". Sample Input and Output 1: Enter the 2 numbers 5 2  
The quotient of 5/2 = 2 Inside finally block Sample Input and Output 2: Enter the 2 numbers 5 0  
DivideByZeroException caught Inside finally block

Q2. Write a program that takes as input the size of the array and the elements in the array. The program then asks the user to enter a particular index and prints the element at that index. Index starts from zero. This program may generate Array Index Out Of Bounds Exception or NumberFormatException. Use exception handling mechanisms to handle this exception.

## Viva Questions:-

Q1. What is the difference between exception and error in Java?

Q2. Can we just use try instead of finally and catch blocks?

Q3. What is the difference between NoClassDefFoundError and ClassNotFoundException in Java?

Q4. Why should we clean up activities such as I/O resources in the finally block?

Q5. Describe OutOfMemoryError in exception handling.

Q6. What is the error of ClassCastException?

Q7. Is there any difference between throw and throws in exception handling in Java?

## EXPERIMENT-6

**Aim:- Convert the content of a given file into the uppercase content of the same file.**

### **Theory:-**

In this program , change all lower-case text to upper-case text in a file in java. Suppose the text file contains the following data –

“Merry and Jack are studying.”

Then after updating the text file with specific value the result will be –

“MERRY AND JACK ARE STUDYING.”

This modification is done with the help of **toUpperCase()** Method. It belongs to the String class in java.

### **Algorithm**

Step 1 – Take the file path as an input.

Step 2 – Read the content of the input text file using `readLine()` method.

Step 3 – Replace all lowercase text with uppercase text in the oldContent using `toUpperCase()` method.

Step 4 – Rewrite the input text file with newContent using `FileWriter ()` method.

Step 5 – Print the result.

### **Syntax**

`readLine()` – It is used to read a single line of text from the console and it belongs to console class in java.

`toUpperCase()` – It converts all the smaller Case letters to upper Case letters.

`write()` – It is used to write a specified string for a given file and belongs to the writer class in java.

`close()` – It is used to close the stream and release the resources that were busy in the stream if any stream is present. It belongs to the Reader class in java.

Java `BufferedReader` class - is used to read the text from a character-based input stream. It can be used to read data line by line by `readLine()` method. It makes the performance fast. It inherits [Reader class](#).

### **Multiple Approaches-**

1. By Using Static User Input
2. By Using User Defined Method

#### **1. By Using Static User Input**

In this approach, file path will be taken as an input. Then as per the algorithm change all lower-case text to upper-case text in a file.

```
import java.io.*;
public class Main{
    public static void main(String[] args){
        File fileToBeModified = new File("C:/Users/Kabir/Desktop/Work/file2.txt");
```

```

String oldContent = "";
BufferedReader reader = null;
FileWriter writer = null;
try{
    reader = new BufferedReader(new FileReader(fileToBeModified));
    String line = reader.readLine(); //Reading the content of input text file
    while (line != null) {
        oldContent = oldContent + line + System.lineSeparator();
        line = reader.readLine();
    }

    //printing the original content
    System.out.println("Original Content of the file: " + oldContent);
    //Replacing lowerCase text to upperCase text
    String newContent = oldContent.toUpperCase();
    //Rewriting the input text file with newContent
    writer = new FileWriter(fileToBeModified);
    writer.write(newContent); //Printing the content of modified file
    //printing the content of the modified file
    System.out.println("New content of the file: " + newContent);
}
catch (IOException e){
    e.printStackTrace();
}
finally{
    try{

        //Closing the resources
        reader.close();
        writer.close();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

## 2. By Using User Defined Method

In this approach, file path will be taken as an input. Then call a user defined method and as per the algorithm change all lower-case text to upper-case text in a file.

Example-

```

import java.io.*;
public class Main {
    public static void main(String[] args)
    {
        //calling user defined method
        modifyFile("C:/Users/Kabir/Desktop/Work/file3.txt");
    }
    //user defined method
    static void modifyFile(String filePath){
        File fileToBeModified = new File(filePath);
        String oldContent = "";
    }
}

```

```

BufferedReader reader = null;
FileWriter writer = null;
try{
    reader = new BufferedReader(new FileReader(fileToBeModified));
    //Reading the content of input text file
    String line = reader.readLine();
    while (line != null) {
        oldContent = oldContent + line + System.lineSeparator();
        line = reader.readLine();
    }
    //printing the original content
    System.out.println("Original Content of the file: " + oldContent);
    //Replacing the lowerCase text to upperCase text
    String newContent = oldContent.toUpperCase();
    //Rewriting the input text file with newContent
    writer = new FileWriter(fileToBeModified);
    //Printing the content of modified file
    writer.write(newContent);
    //printing the content of the modified file
    System.out.println("New content of the file: " + newContent);
}
catch (IOException e){
    e.printStackTrace();
}
finally{
    try{
        //Closing the resources
        reader.close();
        writer.close();
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
}
}

```

## PRACTICE QUESTIONS

- Q1. Write a program to remove the redundant elements from an ArrayList object in Java.
- Q2. Sort ArrayList in descending order using comparator with Java Collections.

## Viva Questions:-

- Q1. How do you handle directories in Java?
- Q2. How do you write to a file using FileWriter class?
- Q3. How do you read from a file using FileReader class?
- Q4. What is the use of BufferedWriter and BufferedReader classes in Java?

## EXPERIMENT-7

**Aim:-** Write a program in Java to sort the content of a given text file.

**Theory:-**

### ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want. The syntax is also slightly different:

**Syntax:-**

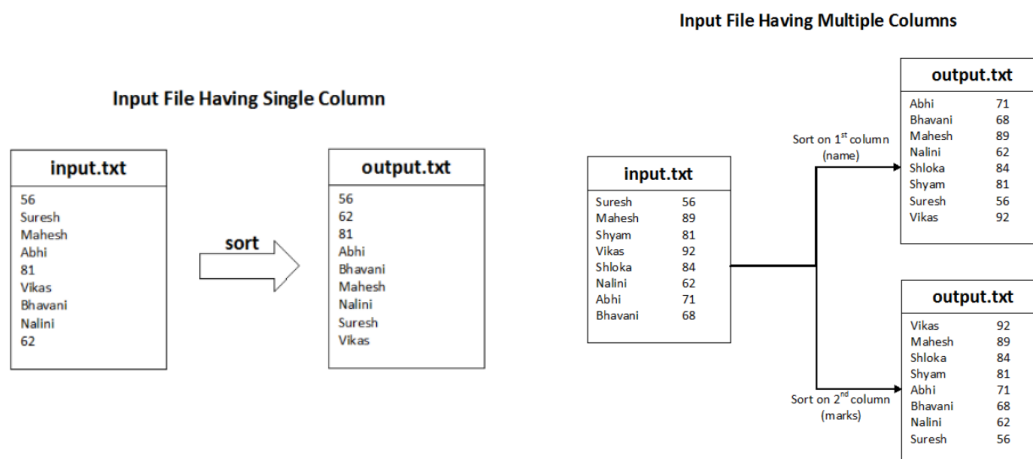
Create an ArrayList object called cars that will store strings:

```
import java.util.ArrayList; // import the ArrayList class
```

```
ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

Java Arrays.sort is function to sort a file with words or terms on each line.

**Collections.sort()** : It is used to sort the elements present in the specified list of Collection in ascending order. java.util.Collections.sort() method is present in java.util.Collections class. It works similar to java.util.Arrays.sort() method but it is better than as it can sort the elements of Array as well as linked list, queue and many more present in it.



**Step-1 :** Read contents of the text file:

Create a BufferedReader object and pass our file to it in the form of FileReader object.

```
FileReader fr = new FileReader("path to file");  
BufferedReader reader = new BufferedReader(fr);
```

**Step-2:** Store the contents in an ArrayList:

We don't know the length of content in the text file, so we need a dynamic size data structure and ArrayList fulfills this purpose.

We read the file line by line and store add it in our ArrayList.

```
ArrayList<String> str = new ArrayList<>();  
String line = "";  
while((line=reader.readLine())!=null){  
    str.add(line);  
}
```



**Step-3 :** Sort the arrayList:

Java Collections provide an easy way to sort the data of any Collection by using sort() method. By default sort() method is used to sort the elements present in the list in ascending order. sort() method is found in java.util.Collections.

```
Collections.sort(str);
```

**Step-4 :** Write the contents to another text file:

```
FileWriter writer = new FileWriter("newFile");
for(String s: str){
    writer.write(s);
    writer.write("\r\n");
}
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
public class Example{
    public static void main(String[] args) throws IOException{
        BufferedReader reader = new BufferedReader(new FileReader("filePath"));
        ArrayList<String> str = new ArrayList<>();
        String line = "";
        while((line=reader.readLine())!=null)
        {
            str.add(line);
        }
        reader.close();
        Collections.sort(str);
        FileWriter writer = new FileWriter("new file");
        for(String s: str){
            writer.write(s);
            writer.write("\r\n");
        }
        writer.close();
    }
}
```

**PRACTICE QUESTIONS:**

- Q1. Write a program to convert an ArrayList to Array and an Array to ArrayList.
- Q2. Write a program to implement ConcurrentHashMap in Java .

**Viva-Questions**

- Q1. What do you understand by Collection Framework in Java?
- Q2. What are primary interfaces provided by Java Collections Framework?
- Q3. Why Collection doesn't extend the Cloneable and Serializable interfaces?

Q4. How the Collection objects are sorted in Java?

Q5. What is the need for overriding equals() method in Java?

## EXPERIMENT-8

**Aim:- Develop an Analog clock using Applet.**

### Theory:-

Applet- An applet is a program written in the Java programming language that can be included in an HTML page, much in the same way an image is included in a page. When you use a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser's Java Virtual Machine (JVM). Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

### Advantage of Applet

There are many advantages of applet. They are as follows:

1. It works at client side so less response time.
2. Secured
3. It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

### Drawback of Applet

Plugin is required at client browser to execute applet.

### Applet Lifecycle :-



### Lifecycle methods for Applet:

1. For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

(i). **public void init():** is used to initialize the Applet. It is invoked only once.

(ii). **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.

(iii). **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

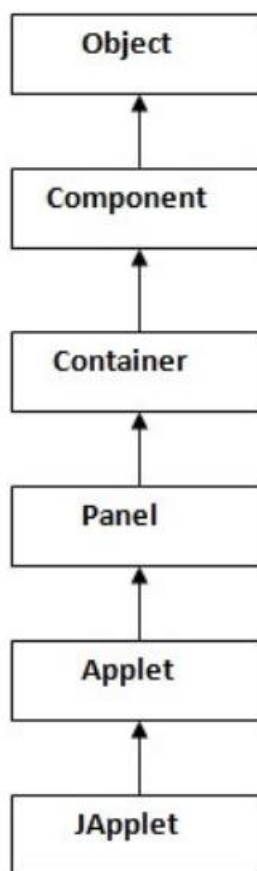
(iv). **public void destroy():** is used to destroy the Applet. It is invoked only once.

## 2. java.awt.Component class

The Component class provides 1 life cycle method of applet.

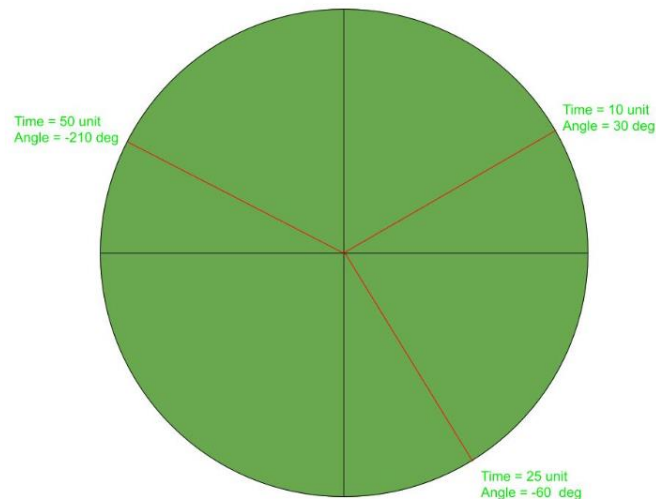
(i). **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

### HIERARCHY OF AN APPLLET :



Applet class extends Panel. Panel class extends Container which is the subclass of Component.

In this program, we shall be animating the applet window to show an **Analog Clock** with a 1-second delay. The idea is to display the system time of every instance.



#### Approach:

Each hand of the clock will be animating with 1-second delay keeping one end at the centre. The position of the other end can be derived by the system time. The angle formed by a hand of the clock in every second will be different throughout its journey. This is why various instances make a different angle to the horizontal line.

#### How to run:

1. Save the file as analogClock.java and run the following commands.
2. javac analogClock.java
3. appletviewer analogClock.java

Below is the implementation of the above approach:

```
// Java program to illustrate analog clock using Applets
import java.applet.Applet;
import java.awt.*;
import java.util.*;
public class analogClock extends Applet {
    @Override
    public void init()
    {
        // Applet window size & color
        this.setSize(new Dimension(800, 400));
        setBackground(new Color(50, 50, 50));
        new Thread() {
            @Override
            public void run()
            {
                while (true) {
                    repaint();
                    delayAnimation();
                }
            }
        }.start();
    }
}
// Animating the applet
```

```

private void delayAnimation()
{
    try {
        // Animation delay is 1000 milliseconds
        Thread.sleep(1000);
    }
    catch (InterruptedException e) {
        e.printStackTrace();
    }
}

// Paint the applet
@Override
public void paint(Graphics g)
{
    // Get the system time
    Calendar time = Calendar.getInstance();
    int hour = time.get(Calendar.HOUR_OF_DAY);
    int minute = time.get(Calendar.MINUTE);
    int second = time.get(Calendar.SECOND);
    // 12 hour format
    if (hour > 12) {
        hour -= 12;
    }
    // Draw clock body center at (400, 200)
    g.setColor(Color.white);
    g.fillOval(300, 100, 200, 200);
    // Labeling
    g.setColor(Color.black);
    g.drawString("12", 390, 120);
    g.drawString("9", 310, 200);
    g.drawString("6", 400, 290);
    g.drawString("3", 480, 200);
    // Declaring variables to be used
    double angle;
    int x, y;
    // Second hand's angle in Radian
    angle = Math.toRadians((15 - second) * 6);
    // Position of the second hand
    // with length 100 unit
    x = (int)(Math.cos(angle) * 100);
    y = (int)(Math.sin(angle) * 100);
    // Red color second hand
    g.setColor(Color.red);
    g.drawLine(400, 200, 400 + x, 200 - y);
    // Minute hand's angle in Radian
    angle = Math.toRadians((15 - minute) * 6);

    // Position of the minute hand with length 80 unit
    x = (int)(Math.cos(angle) * 80);
    y = (int)(Math.sin(angle) * 80);
    // blue color Minute hand
    g.setColor(Color.blue);
    g.drawLine(400, 200, 400 + x, 200 - y);
    // Hour hand's angle in Radian

```

```

        angle = Math.toRadians((15 - (hour * 5)) * 6);
        // Position of the hour hand with length 50 unit
        x = (int)(Math.cos(angle) * 50);
        y = (int)(Math.sin(angle) * 50);
        // Black color hour hand
        g.setColor(Color.black);
        g.drawLine(400, 200, 400 + x, 200 - y);
    }
}

```

### **PRACTICE QUESTIONS:**

- Q1. Write a program to show event handling in applet that prints a message by click on the button.
- Q2. Write a program to perform painting operation in applet by the mouseDragged() method of MouseMotionListener.

### **Viva-Questions**

- Q1. What is the difference between an Applet and a Java Application ?
- Q2. What are the restrictions imposed on Java applets ?
- Q3. What is the difference between applets loaded over the internet and applets loaded via the file system ?
- Q4. What is the applet class loader, and what does it provide ?
- Q5. What is the applet security manager, and what does it provide ?

## EXPERIMENT-9

**Aim :- Develop a Scientific calculator using Swings.**

**Theory :-** Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT]. Swing offers much-improved functionality over AWT, new components, expanded components features, and excellent event handling with drag-and-drop support. Swing in Java is a lightweight GUI toolkit which has a wide variety of widgets for building optimized window based applications.

### Container Class

Any class which has other components in it is called as a container class. For building GUI applications at least one container class is necessary.

Following are the three types of container classes:

1. Panel – It is used to organize components on to a window
2. Frame – A fully functioning window with icons and titles
3. Dialog – It is like a pop up window but not fully functional like the frame

### Java Swing Class Hierarchy

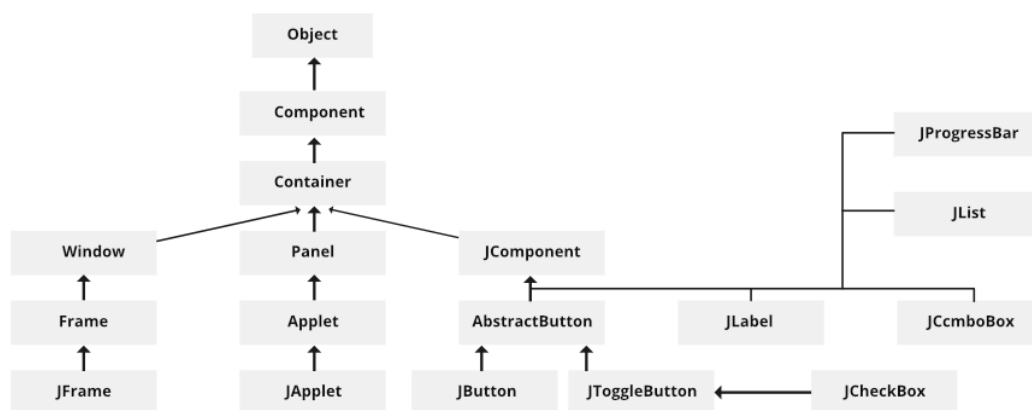


Fig.9.1 Java Swing Class Hierarchy

All the components in swing like **JButton**, **JComboBox**, **JList**, **JLabel** are inherited from the **JComponent** class which can be added to the container classes. Containers are the windows like frame and dialog boxes. Basic swing components are the building blocks of any gui application. Methods like **setLayout** override the default layout in each container. Containers like **JFrame** and **JDialog** can only add a component to itself.



Class	Description
Component	A Component is the Abstract base class for about the non menu user-interface controls of SWING. Components are represents an object with a graphical representation
Container	A Container is a component that can container SWING Components
JComponent	A JComponent is a base class for all swing UI Components In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container
JLabel	A JLabel is an object component for placing text in a container
JButton	This class creates a labeled button
JColorChooser	A JColorChooser provides a pane of controls designed to allow the user to manipulate and select a color
JCheckBox	A JCheckBox is a graphical(GUI) component that can be in either an on-(true) or off-(false) state
JRadioButton	The JRadioButton class is a graphical(GUI) component that can be in either an on-(true) or off-(false) state. in the group
JList	A JList component represents the user with the scrolling list of text items
JComboBox	A JComboBox component is Presents the User with a show up Menu of choices
JTextField	A JTextField object is a text component that will allow for the editing of a single line of text
JPasswordField	A JPasswordField object it is a text component specialized for password entry
JTextArea	A JTextArea object is a text component that allows for the editing of multiple lines of text
ImageIcon	A ImageIcon control is an implementation of the Icon interface that paints Icons from Images

Class	Description
JScrollbar	A JScrollbar control represents a scroll bar component in order to enable users to Select from range values
JOptionPane	JOptionPane provides set of standard dialog boxes that prompt users for a value or Something
JFileChooser	A JFileChooser it Controls represents a dialog window from which the user can select a file.
JProgressBar	As the task progresses towards completion, the progress bar displays the tasks percentage on its completion
JSlider	A JSlider this class is lets the user graphically(GUI) select by using a value by sliding a knob within a bounded interval.
JSpinner	A JSpinner this class is a single line input where the field that lets the user select by using a number or an object value from an ordered sequence

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

### Features Of Swing Class

1. Pluggable look and feel
2. Uses MVC architecture
3. Lightweight Components
4. Platform Independent
5. Advanced features such as JTable, JTabbedPane, JScrollPane, etc.
6. Java is a platform-independent language and runs on any client machine, the GUI look and feel, owned and delivered by a platform-specific O/S, simply does not affect an application's GUI constructed using Swing components
7. Lightweight Components: Starting with the JDK 1.1, its AWT-supported lightweight component development. For a component to qualify as lightweight, it must not depend on any

non-Java [O/s based) system classes. Swing components have their own view supported by Java's look and feel classes.

8. Pluggable Look and Feel: This feature enables the user to switch the look and feel of Swing components without restarting an application. The Swing library supports components' look and feels that remain the same across all platforms wherever the program runs. The Swing library provides an API that gives real flexibility in determining the look and feel of the GUI of an application
9. Highly customizable – Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.
10. Rich controls– Swing provides a rich set of advanced controls like Tree TabbedPane, slider, colorpicker, and table controls.

### Approach To Develop Scientific Calculator Using Swing

1. Math class in Java
2. Handling Classes and Objects creations
3. Functions, Loops, Conditionals, and variables
4. Java Swing and Java AWT for creating a user-friendly GUI.

#### Features:

Addition, Subtraction, Multiplication, and Division

Finding Sin, Cos, Tan, Log, Factorial, Pi, Square, and Square root of a number



### PRACTICE QUESTIONS

- Q1. Write a program to create three buttons with caption OK , SUBMIT, CANCEL.
- Q2. Develop a program using label (swing) to display message “GFG WEB Site Click”.

**Viva Questions –**

Q1. What is EDT thread in Swing?

Q2. What is difference between Swing and AWT in Java?

Q3. Is Swing Thread safe in Java ?

Q4. Which method of Swing are thread-safe?

Q5. What is difference between Container and Component ?

## EXPERIMENT-10

**Aim:- Create an editor like MS-word using swings.**

**Theory:-**

Two Swing classes support styled text: JEditorPane and its subclass JTextPane. The JEditorPane class is the foundation for Swing's styled text components and provides a mechanism through which you can add support for custom text formats. If you want unstyled text, use a text area instead.

To create a simple text editor in Java Swing we will use a JTextArea, a JMenuBar and add JMenu to it and we will add JMenuItem's. All the menu items will have ActionListener to detect any action. There will be a menu bar and it will contain two menus and a button:

### 1. File menu

- **open**: this menuitem is used to open a file
- **save**: this menuitem is used to save a file
- **print** : this menuitem is used to print the components of the text area
- **new** : this menuitem is used to create a new blank file

### 2. Edit menu

- **cut**: this menuitem is to cut the selected area and copy it to clipboard
- **copy**: this menuitem is to copy the selected area to the clipboard
- **paste** : this menuitem is to paste the text from the clipboard to the text area

### 3. Close : this button closes the frame

JTextArea is a part of java Swing package . It represents a multi line area that displays text. It is used to edit the text .

JTextArea inherits JComponent class. The text in JTextArea can be set to different available fonts and can be appended to new text. A text area can be customized to the need of user .

**Constructors of JTextArea are:**

1. **JTextArea ()** : constructs a new blank text area .
2. **JTextArea(String s)** : constructs a new text area with a given initial text.
3. **JTextArea(int row, int column)** : constructs a new text area with a given number of rows and columns.
4. **JTextArea(String s, int row, int column)** : constructs a new text area with a given number of rows and columns and a given initial text.

**Commonly used methods :**

1. **append(String s)** : appends the given string to the text of the text area.
2. **getLineCount()** : get number of lines in the text of text area.
3. **setFont(Font f)** : sets the font of text area to the given font.
4. **setColumns(int c)** : sets the number of columns of the text area to given integer.
5. **setRows(int r)** : sets the number of rows of the text area to given integer.
6. **getColumns()** : get the number of columns of text area.
7. **getRows()** : get the number of rows of text area.

**JMenuBar, JMenu and JMenuItem** are a part of Java Swing package.

JMenuBar is an implementation of menu bar . the JMenuBar contains one or more JMenu objects, when the JMenu objects are selected they display a popup showing one or more JMenuItem's . JMenu basically represents a menu . It contains several JMenuItem Object . It may also contain JMenu Objects (or submenu).

**Constructors :**

1. **JMenuBar()** : Creates a new MenuBar.
2. **JMenu()** : Creates a new Menu with no text.
3. **JMenu(String name)** : Creates a new Menu with a specified name.

4. **JMenu (String name, boolean b) :** Creates a new Menu with a specified name and boolean value specifies it as a tear-off menu or not. A tear-off menu can be opened and dragged away from its parent menu bar or menu.

**Commonly used methods:**

1. **add(JMenu c) :** Adds menu to the menu bar. Adds JMenu object to the Menu bar.
2. **add(Component c) :** Add component to the end of JMenu
3. **add(Component c, int index) :** Add component to the specified index of JMenu
4. **add(JMenuItem menuItem) :** Adds menu item to the end of the menu.
5. **add(String s) :** Creates a menu item with specified string and appends it to the end of menu.
6. **getItem(int index) :** Returns the specified menuitem at the given index

**JEditorPane** class in java is used to display text components that can handle different types of text with style. By default, it can handle only HTML, plain and Rich Text Format ( RTF ). JEditorPane is being primarily used to display HTML content with limited basic HTML tags.

By Default this class is preconfigured to the following three types of content:

text/plain: Plain Text, which is the default type when the content is not recognized. The kit used over here is an extension of DefaultEditorKit that will produce a wrapped plain text view.

text/HTML: HTML Text. The kit used over here is class javax.swing.text.html.HTMLEditorkit which will provide support till HTML (ver. 3.2).

text/RTF: RTF Text. The kit used over here is class javax.swing.text.rtf.RTFEditorkit which will provide limited support Rich Text Format.

### **Constructors of JEditorPane**

Below are the constructors of JEditorPane:

**JEditorPane ():** This type of constructor will simply create a new JEditorPane.

**JEditorPane (String URL):** This type of constructor will create a JEditorPane based on the string in the parameter containing the URL specifications.

**JEditorPane (URL initial page):** This constructor will create the JEditorPane based on the specified URL in the input parameter.

**JEditorPane(String type, String text ):** This constructor will create a JEditorPane that has been initialized to the text given within the parameter.

### **Some Useful Methods of JEditorPane Class**

Following are the method follows:

**void setText (String text):** This method will set the text of the component with the specified text given in the input, which is expected to be in the same content type as of the editor.

**void getText():** This method will return the text of the component within the specified content type of the editor.

**void setPage (URL page):** This method will trigger the JEditorPane to show the specified URL as the current page.

**void setContentType (String type):** This method is used to set the type of content that the editor can handle.

**void addHyperlinkListener (HyperlinkListener listener):** This method will add a hyperlink listener to the component which will help to notify whenever a link or hyperlink is being clicked or selected.

**JTextPane :** A JTextPane is an extension of JEditorPane which provides word processing features like fonts, text styles, colors and etc. If we need to do heavy-duty text processing we can use this class whereas a JEditorPane supports display/editing of HTML and RTF content and can be extended by creating our own EditorKit.

The JTextPane class is a subclass of the JEditorPane class and is a specialized component to handle the styled document with embedded images and components. To display an HTML, RTF, or plain document, use the JEditorPane. To edit or display styled text, use the JTextPane. JTextPane is a mini word processor. A JTextPane uses a styled document, which is an instance of the StyledDocument interface, as its model. The StyledDocument interface inherits the Document interface. DefaultStyledDocument is an implementation class for the StyledDocument interface. A JTextPane uses a DefaultStyledDocument as its default model. A document in a Swing text component contains elements organized in a tree-like structure. An element in a document is an instance of the javax.swing.text.Element interface.

### **PRACTICE QUESTIONS:**

Q1. Write a program to implement JPanel with FLOWLayout.

Q2. Create a chat Frame using swing in java.

### **Viva-questions**

Q1. What is the difference between invokeAndWait() and invokeLater() ?

Q2. Why should any call back implementation execute quickly?

Q3. Which method is used by applet to recognize the height and width ?

Q4. Why does JComponent have add() and remove() methods but Component does not ?

Q5. What is the relationship between clipping and repainting ?

## EXPERIMENT-11

**Aim:-** Create a servlet that uses Cookies to store the number of times a user has visited your Servlet.

**Theory:-** A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

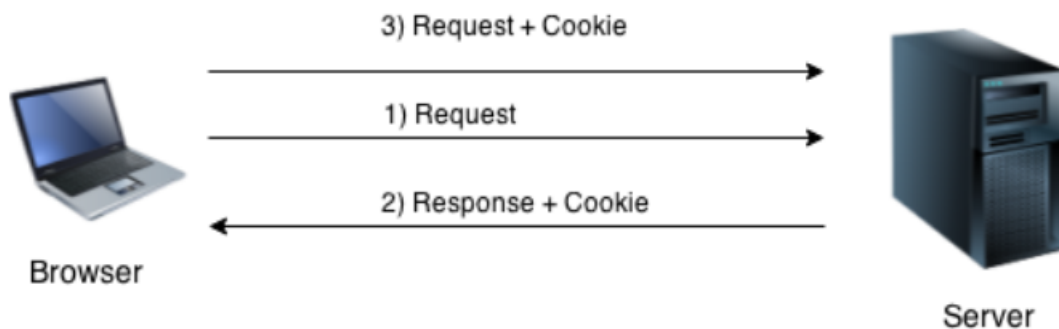
The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

A cookie is a small piece of information that is persisted between multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

### How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



### Types of Cookie

There are 2 types of cookies in servlets.

- 1).Non-persistent cookie
- 2).Persistent cookie

#### Non-persistent cookie

It is valid for single session only. It is removed each time when user closes the browser.

#### Persistent cookie

It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

#### Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

#### Constructor of Cookie class

Cookie() :constructs a cookie.



Cookie(String name, String value) : constructs a cookie with a specified name and value.

### Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

**public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.

**public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

### How to create Cookie?

Let's see the simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object  
response.addCookie(ck);//adding cookie in the response
```

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie  
ck.setMaxAge(0);//changing the maximum age to 0 seconds  
response.addCookie(ck);//adding cookie in the response
```

### VisitServlet.java

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
public class VisitServlet extends HttpServlet  
{  
    static int i=1;  
    public void doGet(HttpServletRequest request,HttpServletResponse response)  
        throws IOException,ServletException  
    {  
        response.setContentType("text/html");  
        PrintWriter out=response.getWriter();  
        String k=String.valueOf(i);  
        Cookie c=new Cookie("visit",k);  
        response.addCookie(c);  
    }  
}
```

```

    int j=Integer.parseInt(c.getValue());
    if(j==1)
    {
        out.println("Welcome to web page ");
    }
    else    {
        out.println("You are visited at "+i+" times");
    }
    i++;
}
}

```

#### **Web.xml**

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
<servlet>
<servlet-name>VisitServlet</servlet-name>
<servlet-class>VisitServlet</servlet-class>
</servlet>
    36<servlet-mapping>
<servlet-name>VisitServlet</servlet-name>
<url-pattern>/VS</url-pattern>
</servlet-mapping>
</web-app>

```

#### **PRACTICE QUESTIONS :**

- Q1. Create a servlet that prints all the request headers it receives, along with their associated values.
- Q2. Write a servlet to show all the parameters sent to the servlet via either GET or POST .

#### **Viva-Questions**

- Q1. How is retrieving information different in Servlets as compared to CGI?
- Q2. How does a servlet examine all its init parameters?
- Q3. What do you mean by Servlet chaining?
- Q4. What is the life cycle contract that a servlet engine must conform to?
- Q5. How can a servlet get the name of the server and the port number for a particular request?

## EXPERIMENT-12

**Aim: - Create a simple Java Bean having bound and constrained properties.**

**Theory: -** JavaBeans are classes that encapsulate many objects into a single object (the bean). It is a java class that should follow following conventions:

1. Must implement Serializable.
2. It should have a public no-arg constructor.
3. All properties in java bean must be private with public getters and setter methods.

// Java program to illustrate the structure of JavaBean class

```
public class TestBean {  
    private String name;  
    public void setName(String name)  
    {  
        this.name = name;  
    }  
    public String getName()  
    {  
        return name;  
    }  
}
```

### **Syntactic rules for setter methods:**

It should be public in nature.

The return-type should be void.

The setter method should be prefixed with set.

It should take some argument i.e. it should not be no-arg method.

### **Syntactic rules for getter methods:**

It should be public in nature.

The return-type should not be void i.e. according to our requirement we have to give return-type.

The getter method should be prefixed with get.

It should not take any argument.

For Boolean properties getter method name can be prefixed with either “get” or “is”. But recommended to use “is”.

**// Java program to illustrate the getName() method on boolean type attribute**

```
public class Test {  
    private boolean empty;  
    public boolean getName()  
    {  
        return empty;  
    }  
    public boolean isempty()  
    {  
        return empty;  
    }  
}
```

// Java Program of JavaBean class

```
package geeks;  
public class Student implements java.io.Serializable  
{  
    private int id;  
    private String name;  
    public Student()  
    {  
    }  
}
```

```

public void setId(int id)
{
    this.id = id;
}
public int getId()
{
    return id;
}
public void setName(String name)
{
    this.name = name;
}
public String getName()
{
    return name;
}
}
// Java program to access JavaBean class
package geeks;
public class Test {
public static void main(String args[])
{
    Student s = new Student(); // object is created
    s.setName("GFG"); // setting value to the object
    System.out.println(s.getName());
}
}

```

**Properties :** To define a property in a bean class, supply public getter and setter methods. For example, the following methods define an int property called mouthWidth:

```

public class FaceBean
{
private int mMouthWidth = 90;
public int getMouthWidth()
{
    return mMouthWidth;
}
public void setMouthWidth(int mw)
{
    mMouthWidth = mw;
}
}

```

A builder tool like NetBeans recognizes the method names and shows the mouthWidth property in its list of properties. It also recognizes the type, int, and provides an appropriate editor so the property can be manipulated at design time.

**1.Indexed Properties :-** An indexed property is an array instead of a single value. In this case, the bean class provides a method for getting and setting the entire array. Here is an example for an int[] property called testGrades:

```

public int[] getTestGrades()
{
    return mTestGrades;
}

```

```

}
public void setTestGrades(int[] tg) {
    mTestGrades = tg;
}

```

For indexed properties, the bean class also provides methods for getting and setting a specific element of the array.

```

public int getTestGrades(int index)
{
    return mTestGrades[index];
}
public void setTestGrades(int index, int grade) {
    mTestGrades[index] = grade;
}

```

**2.Bound Properties :-** A bound property notifies listeners when its value changes. This has two implications: The bean class includes `addPropertyChangeListener()` and `removePropertyChangeListener()` methods for managing the bean's listeners. When a bound property is changed, the bean sends a `PropertyChangeEvent` to its registered listeners.

`PropertyChangeEvent` and `PropertyChangeListener` live in the `java.beans` package.

The `java.beans` package also includes a class, `PropertyChangeSupport`, that takes care of most of the work of bound properties. This handy class keeps track of property listeners and includes a convenience method that fires property change events to all registered listeners.

The following example shows how you could make the `mouthWidth` property a bound property using `PropertyChangeSupport`. The necessary additions for the bound property are shown below.

```

import java.beans.*;
public class FaceBean
{
    private int mMouthWidth = 90;
    private PropertyChangeSupport mPcs =
        new PropertyChangeSupport(this);
    public int getMouthWidth()
    {
        return mMouthWidth;
    }
    public void setMouthWidth(int mw) {
        int oldMouthWidth = mMouthWidth;
        mMouthWidth = mw;
        mPcs.firePropertyChange("mouthWidth",oldMouthWidth, mw);
    }
    public void addPropertyChangeListener(PropertyChangeListener listener)
    {
        mPcs.addPropertyChangeListener(listener);
    }
    public void
        removePropertyChangeListener(PropertyChangeListener listener) {
        mPcs.removePropertyChangeListener(listener);
    }
}

```

Bound properties can be tied directly to other bean properties using a builder tool like NetBeans. You could, for example, take the value property of a slider component and bind it to the mouthWidth property shown in the example. NetBeans allows you to do this without writing any code.

## Constrained Properties

A constrained property is a special kind of bound property. For a constrained property, the bean keeps track of a set of veto listeners. When a constrained property is about to change, the listeners are consulted about the change. Any one of the listeners has a chance to veto the change, in which case the property remains unchanged.

The veto listeners are separate from the property change listeners. Fortunately, the java.beans package includes a VetoableChangeSupport class that greatly simplifies constrained properties.

Changes to the mouthWidth example are shown in bold:

```
import java.beans.*;
public class FaceBean {
    private int mMouthWidth = 90;
    private PropertyChangeSupport mPcs = new PropertyChangeSupport(this);
    private VetoableChangeSupport mVcs = new VetoableChangeSupport(this);
    public int getMouthWidth()
    {
        return mMouthWidth;
    }
    public void
    setMouthWidth(int mw) throws PropertyVetoException
    {
        int oldMouthWidth = mMouthWidth;
        mVcs.fireVetoableChange("mouthWidth", oldMouthWidth, mw);
        mMouthWidth = mw;
        mPcs.firePropertyChange("mouthWidth", oldMouthWidth, mw);
    }
    public void addPropertyChangeListener(PropertyChangeListener listener) {
        mPcs.addPropertyChangeListener(listener);
    }
    public void removePropertyChangeListener(PropertyChangeListener listener) {
        mPcs.removePropertyChangeListener(listener);
    }
    public void addVetoableChangeListener(VetoableChangeListener listener) {
        mVcs.addVetoableChangeListener(listener);
    }
    public void removeVetoableChangeListener(VetoableChangeListener listener) {
        mVcs.removeVetoableChangeListener(listener);
    }
}
```

## **PRACTICE QUESTIONS :**

**Q1.** Create a Simple Bean class Test.java and index.jsp page which loads the bean and sets/gets a simple String Parameter.

**Q2.** Write a Java program to illustrate the getName() method on boolean type attribute.

## **Viva Questions**

**Q1.** What are the various components of Message-driven beans?

**Q2.** What are the important elements of EJB?

**Q3.** What Is a Deployment Descriptor?

**Q4.** What does ACID mean in the context of transaction management?

**Q5.** What is Callback in EJB?

**Q6.** Is it possible to have threading in an EJB application?

**Q7.** What are the callbacks annotations for entity beans?

**Q8.** Is Decorator a Design Pattern in Ejb?

## 8.SOME ADDITIONAL VIVA QUESTIONS

Viva Voce:

1. What is the most important feature of Java?

Java is a platform independent language.

2. What do you mean by platform independence?

Platform independence means that we can write and compile the java code in one platform (eg Windows) and can

execute the class in any other supported platform eg (Linux,Solaris,etc).

3. What is a JVM?

JVM is Java Virtual Machine which is a run time environment for the compiled java class files.

4. Are JVM's platform independent?

111

JVM's are not platform independent. JVM's are platform specific run time implementation provided by the vendor.

5. What is the difference between a JDK and a JVM?

JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM

is purely a run time environment and hence you will not be able to compile your source files using a JVM.

6. What is a pointer and does Java support pointers?

Pointer is a reference handle to a memory location. Improper handling of pointers leads to memory leaks and

reliability issues hence Java doesn't support the usage of pointers.

7. What is the base class of all classes?

java.lang.Object

8. Does Java support multiple inheritance?

Java doesn't support multiple inheritance.

9. Is Java a pure object oriented language?

Java uses primitive data types and hence is not a pure object oriented language.

10. Are arrays primitive data types?

In Java, Arrays are objects.

11. What is difference between Path and Classpath?

Path and Classpath are operating system level environment variables. Path is used to define where the system can find the



executables(.exe) files and classpath is used to specify the location .class files.

12. What are local variables?

Local variables are those which are declared within a block of code like methods. Local variables should be

initialised before accessing them

25. Can a class declared as private be accessed outside its package?

Not possible.

26. Can a class be declared as protected?

A class can't be declared as protected. only methods can be declared as protected.

27. What is the access scope of a protected method?

112

A protected method can be accessed by the classes within the same package or by the subclasses of the class in any

package.

28. What is the purpose of declaring a variable as final?

A final variable's value can't be changed. final variables should be initialized before using them.

29. What is the impact of declaring a method as final?

A method declared as final can't be overridden. A sub-class can't have the same method signature with a different

implementation.

30. I don't want my class to be inherited by any other class. What should I do?

You should declare your class as final. But you can't define your class as final, if it is an abstract class. A class

declared as final can't be extended by any other class.

31. Can you give few examples of final classes defined in Java API?

java.lang.String, java.lang.Math are final classes.

32. How is final different from finally and finalize()?

final is a modifier which can be applied to a class or a method or a variable. final class can't be inherited, final method

can't be overridden and final variable can't be changed.

finally is an exception handling code section which gets executed whether an exception is raised or not by the try

block code segment.

finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give

a final chance for resource releasing activity.

33. Can a class be declared as static?

We can not declare top level class as static, but only inner class can be declared static.

```
public class Test
{
    static class InnerClass
    {
        public static void InnerMethod()
        { System.out.println("Static Inner Class!"); }
    }
    public static void main(String args[])
    {
        Test.InnerClass.InnerMethod();
    }
}
```

113

//output: Static Inner Class!

34. When will you define a method as static?

When a method needs to be accessed even before the creation of the object of the class then we should declare the

method as static.

35. What are the restriction imposed on a static method or a static block of code?

A static method should not refer to instance variables without creating an instance and cannot use "this" operator to

refer the instance.

36. I want to print "Hello" even before main() is executed. How will you achieve that?

Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the

memory and even before the creation of an object. Hence it will be executed before the main() method. And it will be executed only once.

37.How-will-you-communicate-between-two-Applets

The simplest method is to use the static variables of a shared class since there's only one instance of the class and

hence only one copy of its static variables. A slightly more reliable method relies on the fact that all the applets on a

given page share the same AppletContext. We obtain this applet context as follows:

```
AppletContext ac = getAppletContext();
```

AppletContext provides applets with methods such as `getApplet(name)`, `getApplets()`, `getAudioClip`, `getImage`,

`showDocument` and `showStatus()`.

38. Which classes can an applet extend?

An applet can extend the `java.applet.Applet` class or the `java.swing.JApplet` class. The `java.applet.Applet` class

extends the `java.awt.Panel` class and enables you to use the GUI tools in the AWT package. The `java.swing.JApplet`

class is a subclass of `java.applet.Applet` that also enables you to use the Swing GUI tools.

38. How do you cause the applet GUI in the browser to be refreshed when data in it may have changed?

You invoke the `repaint()` method. This method causes the `paint` method, which is required in any applet, to be invoked

again, updating the display.

39. For what do you use the `start()` method?

You use the `start()` method when the applet must perform a task after it is initialized, before receiving user input. The

`start()` method either performs the applet's work or (more likely) starts up one or more threads to perform the work.

40. True or false: An applet can make network connections to any host on the Internet.

False: An applet can only connect to the host that it came from.

114

41. How do you get the value of a parameter specified in the `APPLET` tag from within the applet's code?

You use the `getParameter("Parameter name")` method, which returns the `String` value of the parameter.

42. True or false: An applet can run multiple threads.

True. The `paint` and `update` methods are always called from the AWT drawing and event handling thread. You can

have your applet create additional threads, which is recommended for performing time-consuming tasks.

## **APPENDIX**

### **Course Exit Survey**

#### **Programming in Java Lab (CIC-258)**

Range from 1(lowest) to 5 (highest):

**CIC-258.1** Are you able to understand the compilation process of Java, role of JVM as an emulator and various types of instructions.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

**CIC-258.2** Are you able to learn and apply concepts of Java programming, exceptional handling, and inheritance.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

**CIC-258.3** Are you able to understand the use of multi-threading, AWT components and event handling mechanism in Java.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

**CIC-258.4** Are you able to understand the concepts of I/O streams, IDBC, object serialization, sockets, RMI, JNI, Collection API interfaces, Vector, Stack, Hash table classes, list etc.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

**CIC-258.5** Does the foundation of good programming knowledge of Java language open the door to the latest innovations in programming

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Suggest the way the course could be improved?

What marks range do you expect in this course?

- ☐ Below 50%
- ☐ 50% - 60%
- ☐ 60% - 75%
- ☐ 75% - 90%
- ☐ Above 90%