LAB MANUAL

OF

ADVANCED JAVA PROGRAMMING

CIE-306P/FSD-318P

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

उद्यमेन हि सिध्यन्ति
कार्याणि न मनोरथैः

VISION

To attain global excellence through education, innovation, research, and work ethics with the commitment to serve humanity.

MISSION

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

- To promote **diversification** by adopting advancement in science, technology, management, and allied discipline through continuous learning
- To foster **moral values** in students and equip them for developing sustainable solutions to serve **both national and global needs in society and industry**.
- To **digitize educational resources and process** for enhanced teaching and effective learning..
- To cultivate an environment supporting **incubation, product development, technology transfer, capacity building and entrepreneurship.**
- To encourage **faculty-student networking with alumni, industry, institutions,** and other **stakeholders** for collective engagement.

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY
M A I T
उद्यमेन हि सिध्यन्ति कार्याणि न मनोरथैः

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION

"To attain global excellence through education, innovation, research, and work ethics in the field of Computer Science and engineering with the commitment to serve humanity."

MISSION

- **M1** To lead in the advancement of computer science and engineering through internationally recognized research and education.
- **M2** To prepare students for full and ethical participation in a diverse society and encourage lifelong learning.
- **M3** To foster development of problem solving and communication skills as an integral component of the profession.
- **M4** To impart knowledge, skills and cultivate an environment supporting incubation, product development, technology transfer, capacity building and entrepreneurship in the field of computer science and engineering.
- **M5** To encourage faculty, student's networking with alumni, industry, institutions, and other stakeholders for collective engagement.

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

उद्यमेन हि सिध्यन्ति
कार्याणि न मनोरथैः

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Program Outcomes (POs)

Engineering Graduates will be able to:

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

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

MAIT

उद्यमेन हि सिध्यन्ति कार्याणि न मनोरथैः

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Program Specific Outcomes (PSOs)

PSO1: Able to explore and apply emerging technologies in computer science and engineering such as Artificial Intelligence, Machine Learning, Data Science, etc.

PSO2: Able to independently and collaboratively design, develop and evaluate innovative solutions to existing problems, addressing the needs of industry and society.

PSO3: Able to pursue advanced studies, conduct research and development, and cultivate entrepreneurship skills in the modern computing environment.

Program Educational Objectives (PEOs)

PEO1: Graduates will work with the top institutions and researchers, dedicating themselves to lifelong learning and social responsibility. (M1, M2)

PEO2: Graduates will exhibit outstanding communication skills and the capacity to collaborate effectively within diverse teams. (M3)

PEO3: Graduates cultivating skills in computer science and engineering contribute to driving innovation, entrepreneurship, and economic growth. (M4)

PEO4: Graduates network with stakeholders to contribute to the growth of the department. (M5)

INDEX

1. INTRODUCTION TO LAB

LAB OBJECTIVE-

In the realm of coding, creativity, and state-of-the-art technology have a pivotal role in the domain of software creation. Java is known for its platform independence, robustness, and extensive libraries. Advanced Java concepts are helpful in accomplishing complicated programs, it encompasses an array of technologies, libraries, and various frameworks that are beyond the fundamental principles of Core Java.

Following objectives are targeted to be achieved after studying advanced java programming-

1- To learn the ability to design console based, GUI based and web-based applications
2- To learn how to create dynamic web pages, using Servlets and JSP.
3- To learn Designing applications using pre-built framework.
4- To learn how to do distributed programming in Java using RMI, CORBA.

At the end of the course, a student will be able to:
CIE-306/FSD-318P.CO1 Able to Understand advanced programming concepts.
CIE-306/FSD-318P.CO2 Able to Develop server-side programs using JSP and Servlets
CIE-306/FSD-318P.CO3 Able to Develop component-based java software using java beans.
CIE-306/FSD-318P.CO4 Able to develop advanced projects based on java.

CO-PO MAPPING

Course Outcomes to -Programme Outcomes Mapping (scale 1: low,2: Medium,3:High)												
	PO01	PO02	PO03	PO04	PO05	PO06	PO07	PO08	PO09	PO10	PO11	PO12
CO1	3	2	2	3	2	-	-	2	3	2	2	3
CO2	3	2	2	3	2	-	-	2	3	2	2	3
CO3	3	2	2	3	2	-	-	2	3	2	2	3
CO4	3	2	2	3	2	-	-	2	3	2	2	3

CO-PSO MAPPING

CO	PSO1	PSO2	PSO3
CO1	3	2	-
CO2	3	3	2
CO3.	3	3	2
CO4	3	3	3

2. HARDWARE /SOFTWARE REQUIREMENTS

For Java Programming

JDK Enterprise Edition, NetBeans, Tomcat Server

Java Compatible Web Browser

This Compiler has no special hardware requirements as such. Any System with a minimum 4GB RAM and i3 processor can be used for this lab.

3. LIST OF EXPERIMENTS (PRESCRIBED BY GGSIPU)

Advanced Java Programming Lab	L	P	C
		2	1

Discipline(s) / EAE / OAE	Semester	Group	Sub-group	Paper Code
CSE/IT/CST/ITE	6	PCE	PCE-1	CIE-306P
EAE	6	FSD-EAE	FSD-EAE-1	FSD-318P
CSE-in-EA	7	OAE-CSE-EA	OAE-2	OSD-453P
OAE	7	SD-OAE	SD-OAE-5A	OSD-453P

List of Experiments

S.No	Experiment	CO
1	Write a Java program to demonstrate the concept of socket programming.	CO2
2	Write a Java program to demonstrate the concept of applet programming.	CO1
3	Write a Java program to demonstrate the concept of multi-threading.	CO1
4	Write a Java program to demonstrate the concept of applet	CO1
5	Write a Java program to demonstrate the use of Java Beans.	CO3
6	Write a Java program to insert data into a table using JSP.	CO2
7	Write JSP program to implement form data validation.	CO2
8	Write a Java program to show user validation using Servlet.	CO2
9	Write a program to set cookie information using Servlet.	CO2
10	Develop a small web program using Servlets, JSPs with Database connectivity.	CO4

4. Advanced Java Programming

(Additional/ Advanced Experiments)

S.No.	Program	CO
Week 1		
1.	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.(Inheritance)	CO1
2.	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)	CO1
3.	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)	CO1
4.	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) (Exception Handling)	CO1
Week 2		
5.	Write a Java program to demonstrate the concept of multi-threading	CO1
6.	Implement Producer-Consumer Problem using multithreading.	CO1
7.	Illustrate Deadlock in multithreading.	CO1
Week 3		
8.	Write a Java program to demonstrate the concept of applet programming.	CO1
9.	Write an applet for event handling which prints a message when clicked on the button	CO1
10.	Implement Painting using mouseDragged() method of MouseMotionListener in Applet	CO1
Week 4		
11.	Write a Java program to demonstrate the concept of socket programming.	CO2
12.	Implement Datagram UDP socket programming in java.	CO2
Week 5		
13.	Write a Java program to demonstrate the use of Java Beans.	CO3
14.	Write a program in java to demonstrate encapsulation in java beans.	CO3
Week 6		
15.	Create a servlet that recognizes a 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".	CO2
16.	Write a program to set cookie information using Servlet.	CO2
17.	Write a Java program to show user validation using Servlet.	CO2

Week 7		
18.	Write a Java program to insert data into a table using JSP.	CO2
19.	Write JSP program to implement form data validation.	CO2
Week 8		
20.	Create a database in MySQL using JSP and perform insertion and retrieval operations	CO2
21.	Create a Java JSP login and Sign Up form with Session using MySQL.	CO2
Week 9		
22.	Create Employee Registration Form using a combination of JSP, Servlet, JDBC and MySQL database.(CO4)	CO4
23.	Develop a small web program using Servlets, JSPs with Database connectivity.	CO4
Week 10		
24.	Create an RMI application where the server hosts a calculator service, and the client can perform basic arithmetic operations remotely.	CO4
25.	Create a Hibernate application to perform CRUD (Create, Read, Update, Delete) operations on a Student entity.	CO4
26.	Projects	CO4

5. 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
MAIT
उद्यमेन हि सिध्यन्ति कार्याणि न मनोरथैः

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.

Advanced Java Programming

PRACTICAL RECORD

Paper Code: - CIE-306P/FSD-318P
Name of Student: -
University Roll No.: -
Branch: -
Section/ Group: -

LAB INDEX

S.No	Experiment	Date	R1	R2	R3	R4	R5	Total Marks (10)	Remarks	Faculty Signature
			Is able to identify and define the objective of	Is proposed design /procedure /algorithm	Has the understanding of the	Are the result(s) verified using sufficient	Individuality of submission?			
			2 Marks	2 Marks	2 Marks	2 Marks	2 Marks			
1										
2										
3										

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 Faculty ( ) 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

5. Project report should be added at last page.

6. 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		10 Marks			POs and PSOs Covered	
		0 Marks	1 Marks	2 Marks	PO	PSO
R1	Is able to identify and define the objective of the given problem?	No	Partially	Completely	PO1, PO2	PSO1, PSO2
R2	Is proposed design/procedure/algorithm solves the problem?	No	Partially	Completely	PO1,PO2, PO3	PSO1, PSO2
R3	Has the understanding of the tool/programming language to implement the proposed solution?	No	Partially	Completely	PO1,PO3, PO5	PSO1, PSO2
R4	Are the result(s) verified using sufficient test data to support the conclusions?	No	Partially	Completely	PO2,PO4, PO5	PSO2
R5	Individuality of submission?	No	Partially	Completely	PO8, PO12	PSO1, PSO3

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.

7. INSTRUCTIONS FOR EACH LAB EXPERIMENT

PROGRAM 1

AIM- Write a Java program to demonstrate the concept of socket programming.

Overview-

Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.

Socket class - A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class - The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

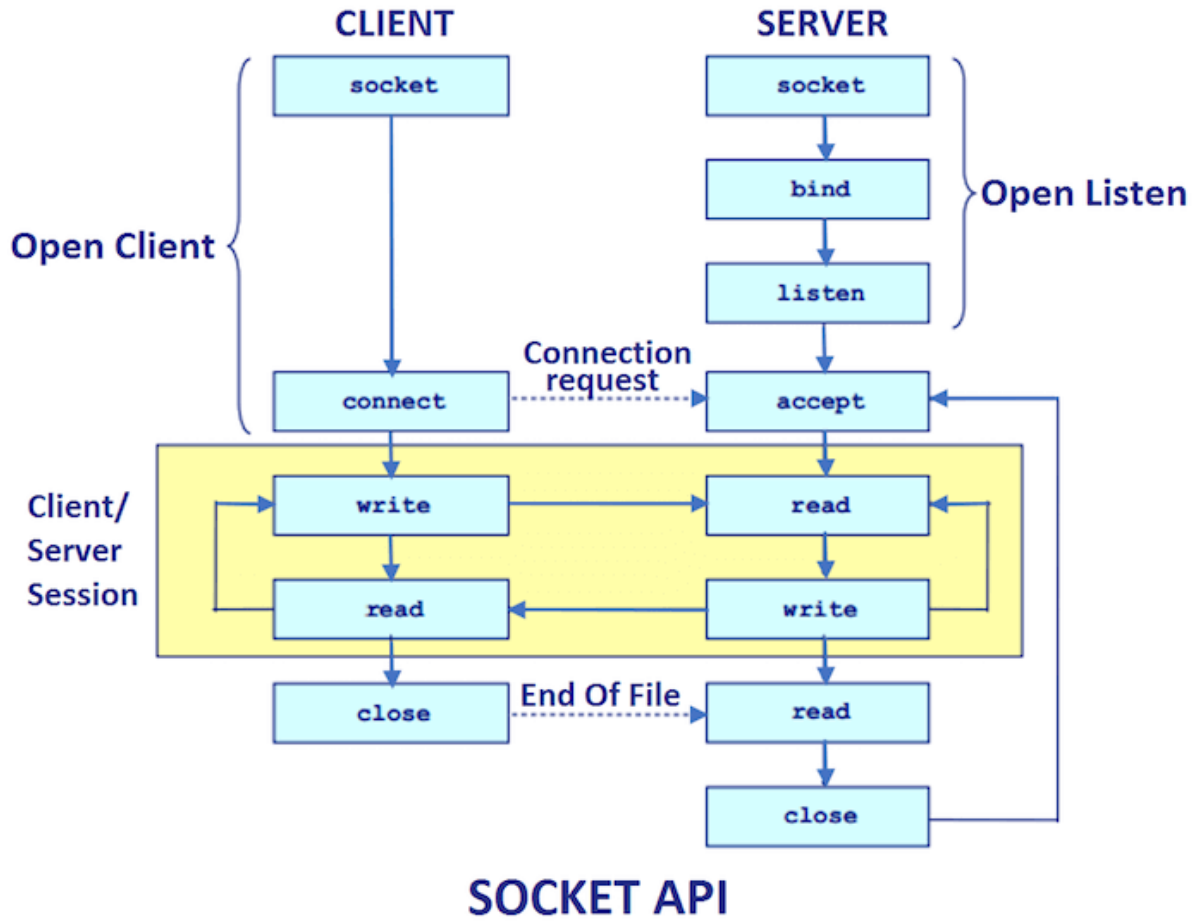


Figure 1 - Socket API

Example of Java Socket Programming

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

1. ServerSocket ss=**new** ServerSocket(6666);
2. Socket s=ss.accept();//establishes connection and waits for the client

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

```
1. Socket s=new Socket("localhost",6666);
```

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

File: MyServer.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyServer {
4. public static void main(String[] args){
5. try{
6. ServerSocket ss=new ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=new DataInputStream(s.getInputStream());
9. String  str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

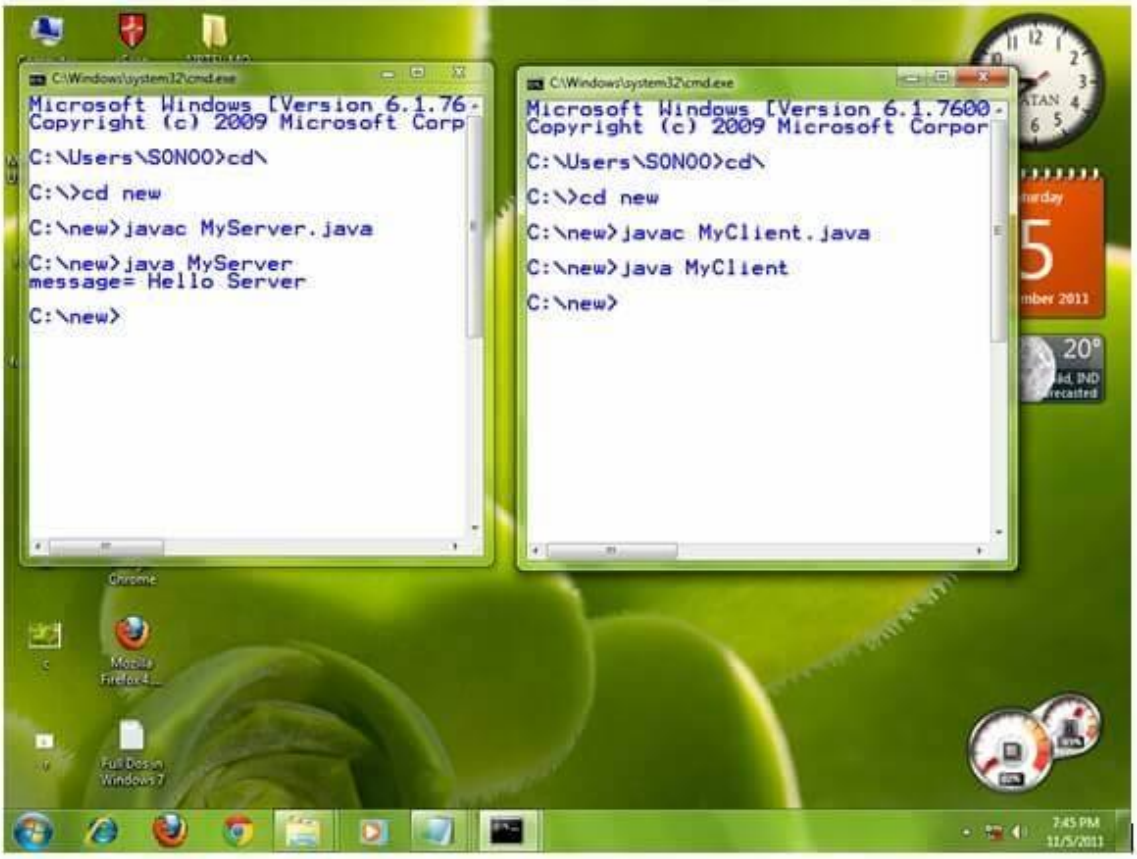
File: MyClient.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyClient {
4. public static void main(String[] args) {
5. try{
6. Socket s=new Socket("localhost",6666);
7. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
```

12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



ADDITIONAL/ PRACTICE PROGRAMS –

Q1. Implement Datagram UDP socket programming in java.

Q2. Implement Socket programming for TCP in Java Server and Client Sockets.

VIVA QUESTIONS –

Q1. Explain the difference between TCP and UDP sockets?

Q2. Describe the process of establishing a TCP connection using sockets.

Q3. How would you handle socket exceptions in your code?
Q4. If you were to design a chat application, would you choose TCP or UDP sockets and why?
Q5. What is the main difference between a server socket and a client socket?
Q6. Can you describe the roles of bind(), listen(), and accept() functions in a socket programming?
Q7. What is a socket timeout and how do you set it in your code?
Q8. Can you explain the role of buffers in socket programming?
Q9. Can you explain the role of the select() function in socket programming?
Q10. Can you explain the use of SO_REUSEADDR and SO_REUSEPORT socket options?

PROGRAM 2

Aim- Write a Java program to demonstrate the concept of applet programming.

Applet - A Java application that is integrated into a webpage is called an applet. It functions as a front-end and is run within the web computer. It makes a page more interactive and dynamic by operating inside the web browser. Applets are hosted on web servers and inserted into HTML pages via the OBJECT or APPLET tags.

It can be compared to a tiny application that runs on the address bar. In addition to updating content in real-time and responding to human input, it may also play basic puzzles or graphics.

The Applet class is contained within java.applet package. The applet contains several methods that offer you detailed control over the execution of your applet.

java.applet defines three interfaces :

AppletContext,

AudioClip, and

AppletStub.

Basically, in Java we have two types of applications:

1. Standalone Applications: The applications that are executed in the context of a local machine are called standalone applications. Their applications use w-1 of system resources and the resources are not sharable. This kind of application contains the main() method.

2.Distributed Applications: The applications that are executed under the control of a browser are called distributed applications. The amount of resources required is very minimum and the resources are sharable. These applications will not contain the main() method. To develop a distributed GUI we use Applet.

Types of Applets –

Java applets can be classified as either local or remote, depending on where they are stored and how easily they can be accessed.

1. **Local Applet-** A local applet is created locally and kept on the local machine. When a web page detects a local applet in the Java system's memory, it does not need to obtain data directly from the internet in order to function. It is defined or provided by the pathname or folder name. When constructing an applet, two properties are used: the source folder, which defines the path name, and the code itself, which defines the filename containing the applet's programming.

2. **Remote Applet-** The remote applet is stored or accessible on another computer that is linked to the world over the internet. We must have internet access on the system to be able to obtain and use the applet that resides on the other machine. We need to be familiar with a remote applet's Uniform Resource Locator (URL) or web location in order to find and download it.

Key Components of Applet Architecture:

1) Applet Class (java.applet.Applet):

- All applets in Java are derived from the Applet class.

- It provides a base class for creating applets and defines several methods that you can override to control the applet's behavior during its lifecycle.

2) Lifecycle Methods:

- **init():** This method is called when an applet is initialized. It's used for applet initialization tasks like setting up resources.
- **start():** Invoked after the init() method or when the applet is revisited. Use it to start or resume operations.
- **stop():** Called when the applet is no longer visible or needs to stop its operations.
- **destroy():** Invoked when the applet is about to be unloaded or destroyed. Cleanup tasks are typically performed here.

3) Graphics Class (java.awt.Graphics):

- Used for rendering and drawing shapes, text, and images on the applet window.
- The paint(Graphics g) method of the applet class receives a Graphics object as an argument that you can use for drawing operations.

4) Event Handling:

- Applets can handle user interactions like mouse clicks, keypresses, etc., by implementing event-handling methods such as mouseClicked(), keyPressed(), etc.

5) Security Restrictions:

- Applets are subject to security restrictions imposed by the browser or Java runtime environment (JRE). They have limited access to resources and can't perform certain operations for security reasons.

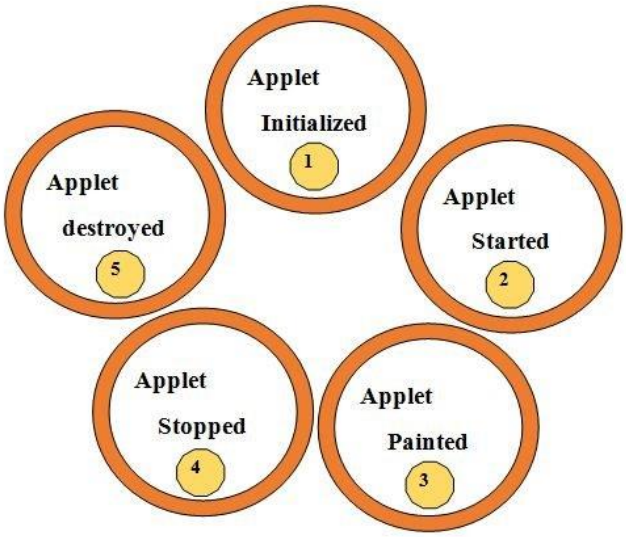


Figure 2. Applet Lifecycle

How to run an Applet Program - An Applet program is compiled in the same way as you have been compiling your console programs. However, there are two ways to run an applet.

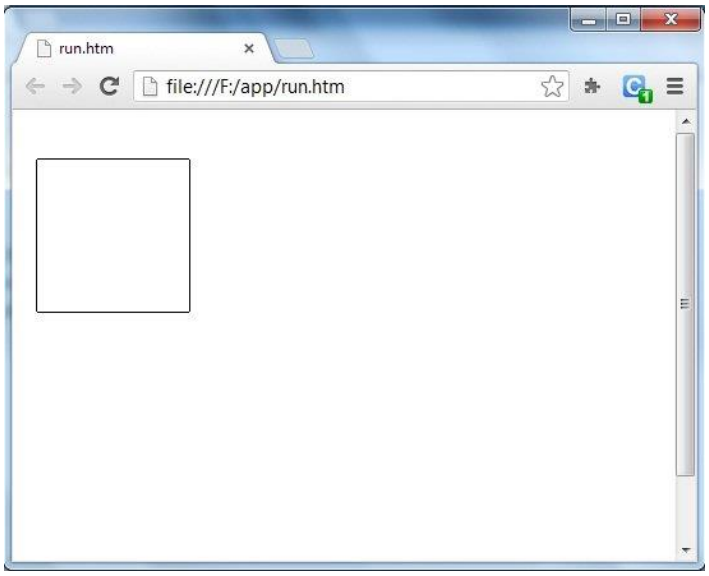
a. Executing the Applet within Java-compatible web browser.

For executing an Applet in an web browser, create short **HTML file** in the same directory. Inside **body** tag of the file, include the following code. (**applet** tag loads the Applet class)

< applet code = "MyApplet" width=400 height=400 >

< /applet >

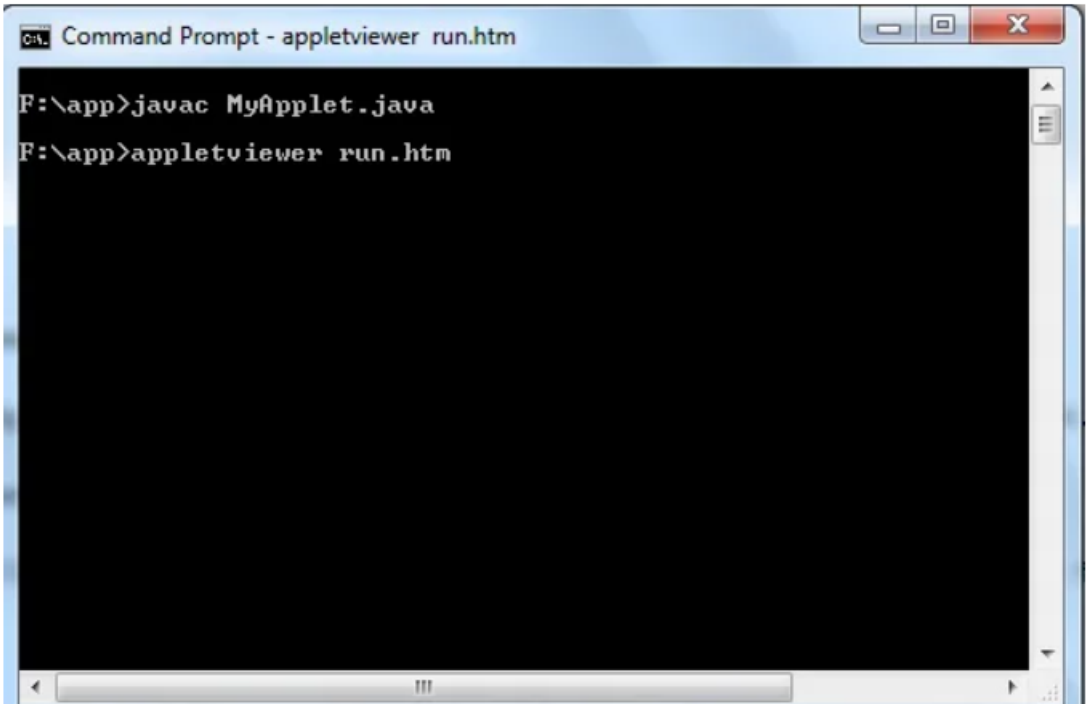
Run the HTML file



b). Running Applet using Applet Viewer

To execute an Applet with an applet viewer, write short HTML file as discussed above. If you name it as run.htm, then the following command will run your applet program.

f:/>appletviewer run.htm



Creating Hello World Applet in Java

```
import java.applet.Applet;
import java.awt.Graphics;
// HelloWorld class extends Applet
public class HelloWorldApplet extends Applet
{
   // Overriding paint() method
   @Override public void paint (Graphics g)
   {
      g.drawString ("Hello World", 20, 20);
   }
}
```

ADDITIONAL/PRACTICE PROGRAMS

Q1. Write an applet for event handling which prints a message when clicked on the button.

Q2. Write a program to Pass Parameters to Applet in Java

Q3. Creating a Simple Banner using Applet in Java

Q4. Implement Painting using mouseDragged() method of MouseMotionListener in Applet.

VIVA-VOICE QUESTIONS

Q1. What are the restrictions imposed on Java applets?

Q2. What are untrusted applets?

Q3. What is the difference between applets loaded over the internet and applets loaded via the file system?

Q4. What is the applet security manager, and what does it provide?

Q5. Which method is called by Applet class to load an image?

Q6. Which method call for a single time?

Q7. What is java plug in software?

Q8. Which class implements upon ImageObserver interface?

Q9. Which facility for communication class provide b/w applet?

Q10. How will you communicate between two applets?

PROGRAM 3

AIM- Write a Java program to demonstrate the concept of multi-threading.

Multithreading- In Java, Multithreading refers to a process of executing two or more threads simultaneously for maximum utilization of the CPU. A thread in Java is a lightweight process requiring fewer resources to create and share the process resources.

Multithreading is a feature in Java that concurrently executes two or more parts of the program for utilizing the CPU at its maximum. The part of each program is called Thread which is a lightweight process. According to the definition, it can be deduced that it expands the concept of multitasking in the program by allowing certain operations to be divided into smaller units using a single application.

Each Thread operates concurrently and permits the execution of multiple tasks inside the same application.

Multithreading vs. Multiprocessing in Java

Multithreading	Multiprocessing
In this, **multiple threads are created** for increasing computational power using a single process.	In this, **CPUs are added** in order to increase computational power.
Many **threads** of a process are executed simultaneously.	Many **processes** are executed simultaneously.
It is **not classified** into any categories.	**Classified** into two categories, symmetric and asymmetric.
The creation of a process is **economical**.	Creation of a process is **time-consuming**.
In this, a **common space of address** is shared by all threads	Every process in this owns a **separate space of address**.

How does Java Support Multithreading?

Java supports multithreading through its built-in features for creating and managing threads. It provides a Thread class that can be extended to create custom threads or Runnable interface to define tasks for threads. To use it, you can either extend the Thread class and override its run() method to define the thread's task or implement the Runnable interface and pass an instance of the class to the Thread constructor. The start() method is then called to begin the execution of the thread, which runs concurrently with other threads in the JVM, enabling multithreading capabilities.

Lifecycle of Thread –

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.

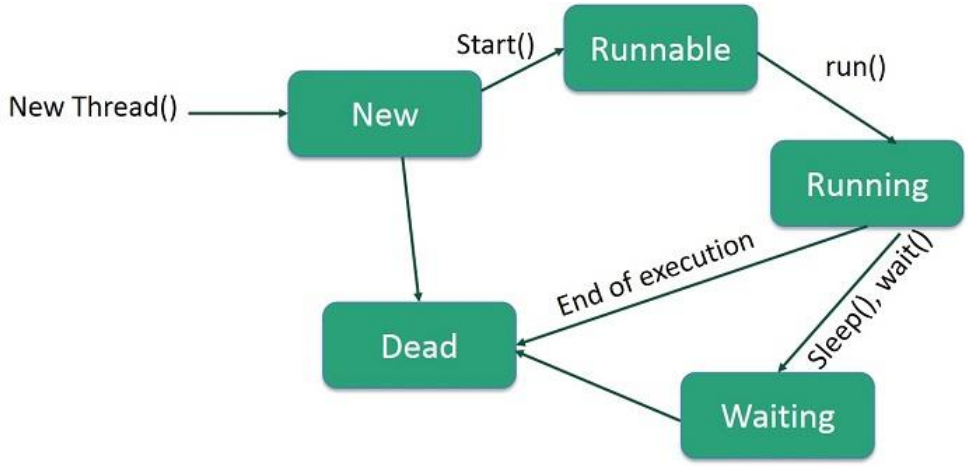


Figure- Lifecycle of thread

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

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();
        }
    }
}
```

PROGRAM TO DEMONSTRATE MULTITHREADING IN JAVA

```
class MultiThread extends Thread{
    public void run(){
        System.out.println("Running Thread Name: "+ this.currentThread().getName());
        System.out.println("Running Thread Priority: "+ this.currentThread().getPriority());
    }
}
public class MultiThrd {
    public static void main(String[] args) {
        MultiThread multiThread1 = new MultiThread();
        multiThread1.setName("First Thread");
```

```
        multiThread1.setPriority(Thread.MIN_PRIORITY);
        MultiThread multiThread2 = new MultiThread();
        multiThread2.setName("Second Thread");
        multiThread2.setPriority(Thread.MAX_PRIORITY);
        MultiThread multiThread3 = new MultiThread();
        multiThread3.setName("Third Thread");
        multiThread1.start();
        multiThread2.start();
        multiThread3.start();
    }
}
```

ADDITIONAL PROGRAMS

Q1. Implement Producer-Consumer Problem using multithreading.

Q2. Illustrate Priorities in Multithreading via help of getPriority() and setPriority() method.

Q3. Illustrate Deadlock in multithreading.

VIVA-VOICE 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 preemptive scheduling and time slicing?

Q5. What is the difference between wait() and sleep() method?

Q6. Can we make the user thread as daemon thread if the thread is started?

Q7. What is the difference between wait() and sleep() method?

PROGRAM-4

Aim- Write a Java program to demonstrate the use of Java Beans.

JavaBeans is a portable, platform-independent model written in Java Programming Language. Its components are referred to as beans.

In simple terms, JavaBeans are classes which encapsulate several objects into a single object. It helps in accessing these object from multiple places. JavaBeans contains several elements like Constructors, Getter/Setter Methods and much more.

JavaBeans has several conventions that should be followed:

1. Beans should have a default constructor (no arguments)
2. Beans should provide getter and setter methods
3. A getter method is used to read the value of a readable property
4. To update the value, a setter method should be called
5. Beans should implement java.io.serializable, as it allows to save, store and restore the state of a JavaBean you are working on

JavaBeans Properties

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.A JavaBean property may be **read, write, read only**, or **write only**. JavaBean properties are accessed through two methods in the JavaBean's implementation class –

S.No.	Method & Description
1	get**PropertyName**() For example, if property name is firstName, your method name would be **getFirstName()** to read that property. This method is called accessor.
2	set**PropertyName**() For example, if property name is firstName, your method name would be **setFirstName()** to write that property. This method is called mutator.

A read-only attribute will have only a getPropertyName() method, and a write-only attribute will have only a setPropertyName() method.

The example program shown below demonstrates how to implement JavaBeans.

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

Next program is written in order to access the JavaBean class that we created above:

```
public class Employee1 {
public static void main(String args[])
{
   Employee s = new Employee();
   s.setName("Chandler");
   System.out.println(s.getName());
}
}
```

Output:

Chandler

Java Beans in JSPs and Servlets

Java Beans can be used effectively in JSPs (Java Server Pages) and Servlets, two key technologies in the world of Java web applications. They can be used to encapsulate data that can be reused across multiple JSPs or Servlets**.**

```
<%@ page import="com.example.StudentBean" %>

<jsp:useBean id="student" class="com.example.StudentBean" />

<jsp:setProperty name="student" property="name" value="John Doe" />

<jsp:setProperty name="student" property="age" value="22" />

Student Name: <jsp:getProperty name="student" property="name" />

Student Age: <jsp:getProperty name="student" property="age" />
```

In this example, we're using the StudentBean we defined earlier. The jsp:useBean tag is used to instantiate the Bean, and the jsp:setProperty tags are used to set the values of the Bean's properties. Finally, the jsp:getProperty tags are used to retrieve and display these values.

Using Java Beans in JSPs and Servlets can greatly simplify your code, making it cleaner and easier to maintain. However, as your applications get more complex, you might find that Java Beans are not flexible enough to meet your needs. In such cases, you might want to consider using more powerful alternatives such as POJOs (Plain Old Java Objects) or EJBs (Enterprise JavaBeans).

ADDITIONAL/PRACTICE QUESTIIONS

Q1. Implement a program Java Bean to represent person details.

Q2. Write a program in java to demonstrate encapsulation in java beans.

VIVA-VOICE QUESTIONS

Q1. What is the relationship between Enterprise JavaBeans and JavaBeans?

Q2. How to implement a bound property in your bean application?

Q3. How and when will the JavaBeans Migration Assistant to ActiveX be available?

Q4. Can both Java applets and JavaBeans components use the InfoBus?

Q5.JavaBeans has mechanisms like bound properties for data transfer between components. Why is the InfoBus necessary?

Q6. What are the properties of java beans?

Q7. What is the relation between the InfoBus and RMI?

Q8. How can you differentiate between the different types of EJBs: Session, Entity, and Message-Driven Beans?

PROGRAM 5

AIM- Write a Java program to insert data into a table using JSP.

The database is used for storing various types of data which are huge and has storing capacity in gigabytes. JSP can connect with such databases to create and manage the records.

For insert data in MySQL using JSP first we have to create a table in data base.

The INSERT INTO statement is used to insert new data to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,...)
```

For creating table the SQL query is:

SQL Query

```
CREATE TABLE users
(
id int NOT NULL AUTO_INCREMENT,
first_name varchar(50),
last_name varchar(50),
city_name varchar(50),
email varchar(50),
PRIMARY KEY (id)
);
```

Here we using 2 files for insert data in MySQL:

- **index.html**:for getting the values from the user
- **process.jsp**:A JSP file that process the request

index.html

```
<!DOCTYPE html>
<html>
<body>
<form method="post" action="process.jsp">
First name:<br>
<input type="text" name="first_name">
<br>
Last name:<br>
<input type="text" name="last_name">
<br>
City name:<br>
<input type="text" name="city_name">
<br>
Email Id:<br>
<input type="email" name="email">
<br><br>
<input type="submit" value="submit">
```

```
</form>
</body>
</html>
```

process.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@page import="java.sql.*,java.util.*"%>
<%
String first_name=request.getParameter("first_name");
String last_name=request.getParameter("last_name");
String city_name=request.getParameter("city_name");
String email=request.getParameter("email");
try
{
        Class.forName("com.mysql.jdbc.Driver");
          Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root",
"");
          Statement st=conn.createStatement();
          int i=st.executeUpdate("insert into
users(first_name,last_name,city_name,email)values('"+first_name+"','"+last_name+"','"+city_name+"'
,'"+email+"')");
       out.println("Data is successfully inserted!");
       }
       catch(Exception e)
       {
       System.out.print(e);
       e.printStackTrace();
       }
 %>
```

ADDITIONAL / PRACTICE PROGRAMS

Q1. Create a database in MySQL using JSP and perform insertion and retrieval operations.

Q2. Create a Java JSP login and Sign Up form with Session using MySQL.

VIVA-VOICE QUESTIONS

Q1. What is JSP?

Q2. What do you mean by a scriptlet in JSP? And what is its syntax?

Q3. What is the use of JSP expressions?

Q4. What are the types of JSP directives?

Q5. What does the id and scope attribute mean in the action elements?

Q6. Explain the life cycle of a JSP

Q7. What are the various methods used to read data using JSP?

PROGRAM 6

AIM- Write JSP program to implement form data validation.

WebLogic JSP form validation tags provide a convenient way to validate the entries an end user makes to HTML form text fields generated by JSP pages. Using the WebLogic JSP form validation tags prevents unnecessary and repetitive coding of commonly used validation logic. The validation is performed by several custom JSP tags that are included with the WebLogic Server distribution.

The tags can

- Verify that required fields have been filled in (**Required Field Validator** class).
- Validate the text in the field against a regular expression (**Regular Expression Validator** class).
- Compare two fields in the form (**Compare Validator** class).
- Perform custom validation by means of a Java class that you write (**Custom Validator** class).
- WebLogic JSP form validation tags include:
- **<wl:summary>**
- **<wl:form>**
- **<wl:validator>**

When a validation tag determines that data in a field is not been input correctly, the page is re-displayed and the fields that need to be re-entered are flagged with text or an image to alert the end user. Once the form is correctly filled out, the end user's browser displays a new page specified by the validation tag.

Validation Tag Attribute Reference

This section describes the WebLogic form validation tags and their attributes. Note that the prefix used to reference the tag can be defined in the taglib directive on your JSP page. For clarity, the wl prefix is used to refer to the WebLogic form validation tags throughout this document.

<wl:summary> - It is the parent tag for validation. Place the opening <wl:summary> tag before any other element or HTML code in the JSP. Place the closing </wl:summary> tag anywhere after the closing </wl:form> tag(s).

- **name**—(Optional) Name of a vector variable that holds all validation error messages generated by the <wl:validator> tags on the JSP page. If you do not define this attribute, the default value, errorVector, is used. The text of the error message is defined with the errorMessage attribute of the <wl:validator> tag.

- `headerText`—A variable that contains text that can be displayed on the page. If you only want this text to appear when errors occur on the page, you can use a scriptlet to test for this condition. For example:

  <% if(summary.size() >0) {

  out.println(headerText);

  }

  %>

  Where *summary* is the name of the vector assigned using the `name` attribute of the <wl:summary> tag.

- redirectPage—URL for the page that is displayed if the form validation does not return errors. This attribute is not required if you specify a URL in the action attribute of the <wl:form> tag.

<wl:form>

The **<wl:form>** tag is similar to the HTML **<form>** tag and defines an HTML form that can be validated using the WebLogic JSP form validation tags. You can define multiple forms on a single JSP by uniquely identifying each form using the **name** attribute.

- **method**—Enter **GET** or **POST**. Functions exactly as the **method** attribute of the HTML **<form>** tag.
- **action**—URL for the page that is displayed if the form validation does not return errors. The value of this attribute takes precedence over the value of the redirectPage attribute of the **<wl:summary>** tag and is useful if you have multiple forms on a single JSP page.

  Do not set the **action** attribute to the same page containing the **<wl:form>** tag—you will create an infinite loop causing a **StackOverFlow** exception.

- **name**—Functions exactly as the **name** attribute of the HTML **<form>** tag. Identifies the form when multiple forms are used on the same page. The **name** attribute is also useful for JavaScript references to a form.

<wl:validator> - Use one or more **<wl:validator>** tags for each form field. If, for instance, you want to validate the input against a regular expression and also require that something be entered into the field you would use two **<wl:validator>** tags, one using the **RequiredFieldValidator** class and another using the **RegExpValidator** class. (You need to use both of these validators because blank values are evaluated by the Regular Expression Field Validator as valid.)

- **errorMessage**—A string that is stored in the vector variable defined by the **name** attribute of the **<wl:summary>** tag.
- **expression**—When using the **RegExpValidator** class, the regular expression to be evaluated. If you are not using **RegExpValidator**, you can omit this attribute.
- **fieldToValidate**—Name of the form field to be validated. The name of the field is defined with the **name** attribute of the HTML **<input>** tag.
- **validatorClass**—The name of the Java class that executes the validation logic.

Using a Custom Validator Class

To use your own validator class:

1. Write a Java class that extends the weblogicx.jsp.tags.validators.CustomizableAdapter abstract class.
2. Implement the validate() method. In this method:

   a) Look up the value of the field you are validating from the ServletRequest object. For example:

   ```
   String val = req.getParameter("field_1");
   ```

   b) Return a value of true if the field meets the validation criteria.

3. Compile the validator class and place the compiled .class file in the WEB-INF/classes directory of your Web application.

4. Use your validator class in a <wl:validator> tag by specifying the class name in the validatorClass attribute. For example:

   <wl:validator errorMessage="This field is required" fieldToValidate="field_1"

   validatorClass="mypackage.myCustomValidator">

Extending the CustomizableAdapter Class

The CustomizableAdapter class is an abstract class that implements the Customizable interface and provides the following helper methods:

1. getFieldToValidate()—Returns the name of the field being validated (defined by the fieldToValidate attribute in the <wl:validator> tag)
2. getErrorMessage()—Returns the text of the error message defined with the errorMessage attribute in the <wl:validator> tag.
3. getExpression()—Returns the text of the expression attribute defined in the <wl:validator> tag.

JSP with WebLogic JSP Form Validation Tags

```
<%@ taglib uri="tagl" prefix="wl" %>

<%@ taglib uri="input" prefix="input" %>

<wl:summary

name="summary"

headerText="<font color=red>Some fields have not been filled out correctly.</font>"

redirectPage="successPage.jsp"

>

<html>

<head>

<title>Untitled Document</title>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

</head>

<body bgcolor="#FFFFFF">

<% if(summary.size() >0 )   {

   out.println("<h3>" + headerText + "</h3>");

} %>
```

```
<% if (summary.size() > 0) {
out.println("<H2>Error Summary:</h2>");
for (int i=0; i < summary.size(); i++) {
out.println((String)summary.elementAt(i));
out.println("<br>");
}
} %>
<wl:form method="GET" action="successPage.jsp">
 User Name: <input:text name="username"/>
 <wl:validator
  fieldToValidate="username"
  validatorClass="weblogicx.jsp.tags.validators.RequiredFieldValidator"
  errorMessage="User name is a required field!"
 >
  <img src=images/warning.gif> This is a required field!
 </wl:validator>
<p>
 Password: <input type="password" name="password">
 <wl:validator
  fieldToValidate="password"
  validatorClass="weblogicx.jsp.tags.validators.RequiredFieldValidator"
  errorMessage="Password is a required field!"
 >
  <img src=images/warning.gif> This is a required field!
 </wl:validator>
 <p>
```

```
Re-enter Password: <input type="password" name="password2">
<wl:validator
  fieldToValidate="password,password2"
  validatorClass="weblogicx.jsp.tags.validators.CompareValidator"
  errorMessage="Passwords don't match"
>
  <img src=images/warning.gif> Passwords don't match.
</wl:validator>
<p>
<input type="submit" value="Submit Form">  </p>
</wl:form>
</wl:summary>
</body>
</html>
```

ADDITIONAL PROGRAMS-

Q1. Implement Regular Expressions validation before submitting data in JSP.

Q2. Implement Customizable adapter class in a registration form in JSP.

VIVA-VOICE QUESTIONS

Q1**.** Which implicit object in JSP represents the client's requested information?

Q2. What do you mean by Context Initialization Parameters?

Q3. What is the major difference between ServletContext and PageContext?

Q4. Why are the request.getRequestDispatcher() and context.getRequestDispatcher() used?

Q5. Differentiate between JSP Scriptlet tag and Declaration tag.

PROGRAM 7

AIM- Write a Java program to show user validation using Servlet.

Overview- A Servlet is an instance of a class that implements javax.servlet.Servlet. Most Servlets, however, are extended to one of the standard implementations of that interface, as it namely javax.servlet.GenericServlet and the javax.servlet.http.HttpServlet. Developers use this as a blueprint when crafting their Servlets code. Implementing the Servlet's interface or extending the GenericServlet/HttpServlet class provides a framework for creating a Servlet as well as significant default functionality to it. The Servlet's code spec must override at least one method when any custom functionality is being implemented.

In order to initialize the Servlet, the Web server:

1. Loads the Servlet class [and probably other classes which are referenced by the Servlet] and creates an instance by calling the no-arguments constructor
2. Calls the Servlet's init(ServletConfig config). init() is called only once when the Servlet is loaded in memory for the first time and stored in the ServletConfig object. The Servlet creator can encapsulate any code that must run before the main code spec of the Servlet is executed such as connecting to a database for instance. The ServletConfig object supplies the Servlets with info about its initializing parameters. These parameters are given to the Servlet itself and are not associated with any single request. The ServletConfig object can be retrieved by calling a Servlet's getServletConfig(), and This is handled by GenericServlet

Any Java-enabled Web server automatically invokes the Servlet's service() in response to a client request. This means when a Servlet is initialized, its service (ServletRequest request, ServletResponse response) is called for every request to the Servlet. Hence, service() must be overridden to provide customized functionality the moment a Servlet is invoked. However, if the Servlet developer chooses not to override the service(), there are other methods that can be invoked in response to a client's request to it. When the Servlet needs to be unloaded, for example, because a new version should be loaded or the server is shutting down, destroy() is called.

```
public dass SkeletonServet extends HttpServlet
{
   public void init()
   {
      // Initialization code goes here
   }
   publicvoid service()
   {
      // Meaningful work happens here
   }
   public void destroy()
   {
      // Free resources here
   }
}
```

Form Validation in Java Servlet | Verifying the pattern and format of form data before it is getting used in the business logic as inputs are called form validations, otherwise business logic may give invalid results or exceptions by taking the inputs.

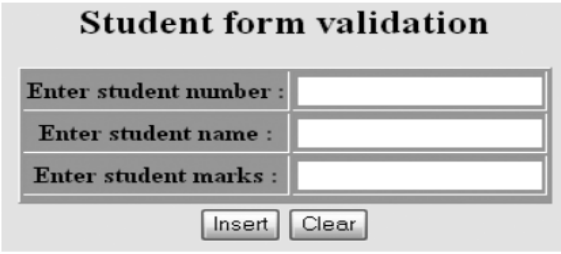
Example:- Checking required fields of the form are filled or not, checking whether age types as a numeric value and e.t.c.
Difference between form validation logic and business logic? In form validation logic, the pattern and format of the form data will be verified. Whereas business logic/request processing logic takes form data as inputs and uses them to process the request and generate the results.
Example:-

Form Validation Examples	Business Logic Examples
Credit card number is having 16 digits or not? (format verification)	Credit card number is existing or a valid number?
The given date value is in MM/DD/YYYY pattern or not? (Pattern verification)	Getting sales report for given data value.
Whether age is entered as a number or not, and it is in the range of 1 to 150?	Checking whether the person is eligible to vote or not based on the given age.

Form validation logic in web application

Client Side	Server Side
Place JavaScript code as form validation logic.	Place Java code as form validation logic.
Executes in browser by coming browser along with form page.	Always resides and executes in the server by becoming part of servlet component code.

Example- Develop a html form and validate that data by using a servlet.



Validations:
stno - must contain data and it should contain always int data.
sname - must contain data and no special characters are allowed.
smarks - must contain data and it should contain float data.
Student database:

```
create table Student (
stno number (3),
stname varchar2 (15),
stmarks number (5,2)
```

```
);
```

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>ValidationServ</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/validation</url-pattern>
    </servlet-mapping>
</web-app>
```

validpro.html:

```
<html>
    <title>Validation Project</title>
    <head><center><h2>Student form validation</h2></center></head>
    <body bgcolor="#FFE4C4">
    <center>
        <form name="validpro" action="./validation" method="post">
            <table border="1" bgcolor="A9A9A9">
                <tr>
                    <th>Enter student number : </th>
                    <td><input type="text" name="validpro_sno" value=""></td>
                </tr>
                <tr>
                    <th>Enter student name : </th>
                    <td><input type="text" name="validpro_sname" value=""></td>
                </tr>
                <tr>
                    <th>Enter student marks : </th>
                    <td><input type="text" name="validpro_smarks" value=""></td>
                </tr>
                <tr>
                <table>
                    <tr>
                        <td><input type="submit" name="validpro_insert" value="Ins
ert"></td>
                        <td><input type="reset" name="validpro_clear" value="Clear
"></td>
                    </tr>
                </table>
                </tr>
            </table>
        </form>
    </center>
    </body>
</html>
```

ValidationServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
```

```
import java.io.*;
import java.sql.*;
import java.util.*;

public class ValidationServ extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws Serv
letException, IOException {
        ArrayList al = new ArrayList();
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        String sno1 = req.getParameter("validpro_sno");
        String sname = req.getParameter("validpro_sname");
        String smarks1 = req.getParameter("validpro_smarks");
        int sno = 0;
        float smarks = 0;
        if ((sno1 == null) || (sno1.equals(""))) {
            al.add("PROVIDE STUDENT NUMBER...");
        } else {
            try {
                sno = Integer.parseInt("sno1");
            } catch (NumberFormatException nfe) {
                al.add("PROVIDE int DATA IN STUDENT NUMBER...");
            }
        }
        if ((sname == null) || (sname.equals(""))) {
            al.add("PROVIDE STUDENT NAME...");
        }
        if ((smarks1 == null) || (smarks1.equals(""))) {
            al.add("PROVIDE STUDENT MARKS...");
        } else {
            try {
                smarks = Float.parseFloat("smarks1");
            } catch (NumberFormatException nfe) {
                al.add("PROVIDE float DATA IN STUDENT MARKS...");
            }
        }
        if (al.size() != 0) {
            pw.println(al);
        } else {
            try {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con = DriverManager.getConnection("jdbc:oracle:thin:@lo
calhost:1521:Hanuman", "scott", "tiger");
                PreparedStatement ps = con.prepareStatement("insert into Student v
alues (?,?,?)");
                ps.setInt(1, sno);
                ps.setString(2, sname);
                ps.setFloat(3, smarks);
                int i = ps.executeUpdate();
                if (i > 0) {
                    pw.println("RECORD INSERTED...");
                } else {
                    pw.println("RECORD NOT INSERTED...");
                }
                con.close();
            } catch (Exception e) {
```

```
                res.sendError(503, "PROBLEM IN DATABASE...");
            }
        }
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res) throws Ser
vletException, IOException {
        doGet(req, res);
    }
};
```

ADDITIONAL/PRACTICE QUESTIONS :

Q1. Implement form validation in marriage application input.html form page using JavaScript

1. Person name is required.
2. Person's name must have a minimum of 5 characters.
3. Personage is required.
4. Personage must be a numeric value
5. Personage must be there between 1 to 125.

Q2. Implement the following form validations in Election Commission Voter Registration Form

1. The name must be valid. It should not be null (not filled), or empty (only spaces), and the length of the name should not be lesser than 5 characters.
2. Age must be valid. It should not be null (not filled), or empty(only spaces), and the length of the name should not be zero.
3. The age should be in numeric format, (not in words, or special characters).
4. The age should not be less than 0 and greater than 125

VIVA-VOICE QUESTIONS

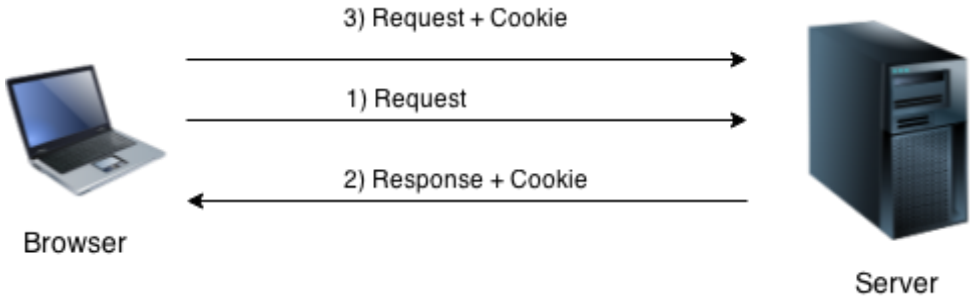
Q1. How do you write a servlet that is part of a web application?
Q2. What do you mean by server-side include (SSI) functionality in Servlets?
Q3. Explain the server-side include expansion.

Q4. Define ‘init’ and ‘destroy’ methods in servlets.
Q5. How is retrieving information different in Servlets as compared to CGI?
Q6. What is the life cycle contract that a servlet engine must conform to?
Q7. What do you mean by Servlet Reloading?
Q8. How can a servlet get the name of the server and the port number for a particular request?
Q9. How can a servlet get information about the client machine?
Q10. Explain Request parameters associated with servlets.

PROGRAM 8

AIM: - Write a program to set cookie information using Servlet.

A cookie is a small piece of information that is persisted between the 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.



Types of Cookie-

1. Non-persistent cookie

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

2. 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-

Constructor	Description
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:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **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
```

How to delete Cookie?

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

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

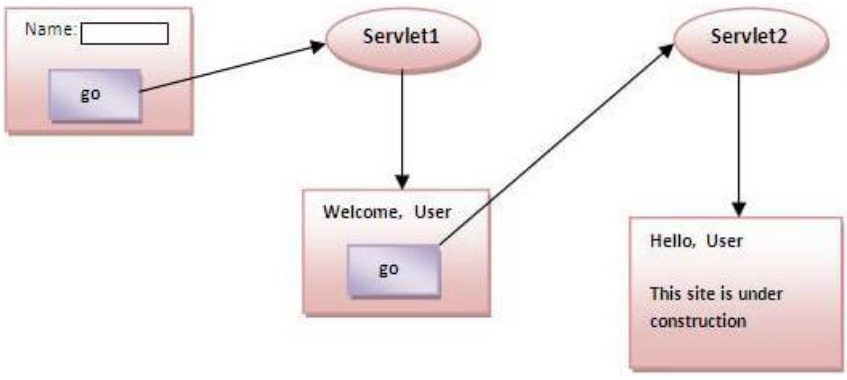
How to get Cookies?

Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++)
{
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response)
{
   try
{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   String n=request.getParameter("userName");
   out.print("Welcome "+n);
   Cookie ck=new Cookie("uname",n);//creating cookie object
   response.addCookie(ck);//adding cookie in the response
   //creating submit button
   out.print("<form action='servlet2'>");
   out.print("<input type='submit' value='go'>");
   out.print("</form>");
   out.close();
      }catch(Exception e){System.out.println(e);}
  }
}
```

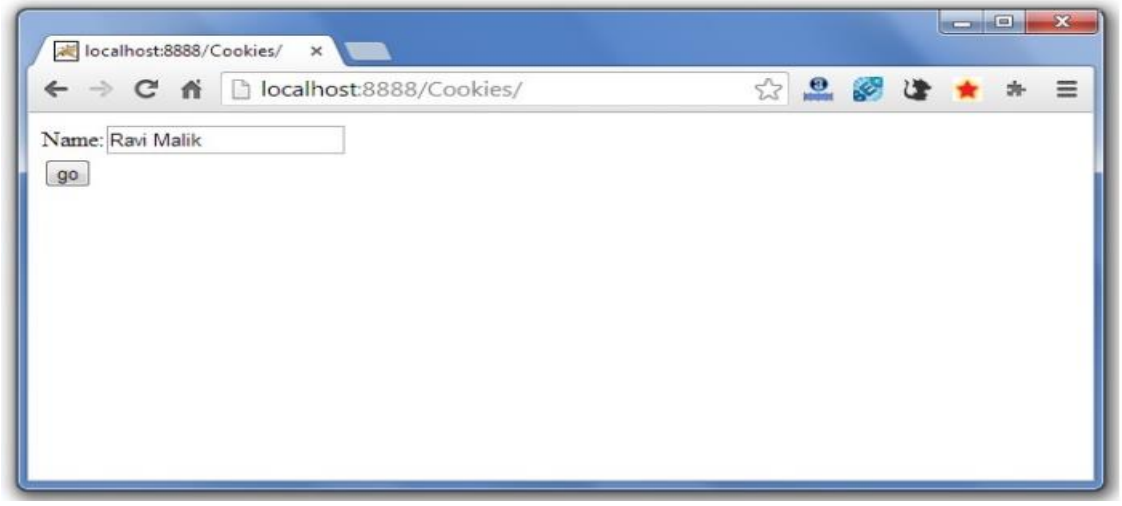
SecondServlet.java

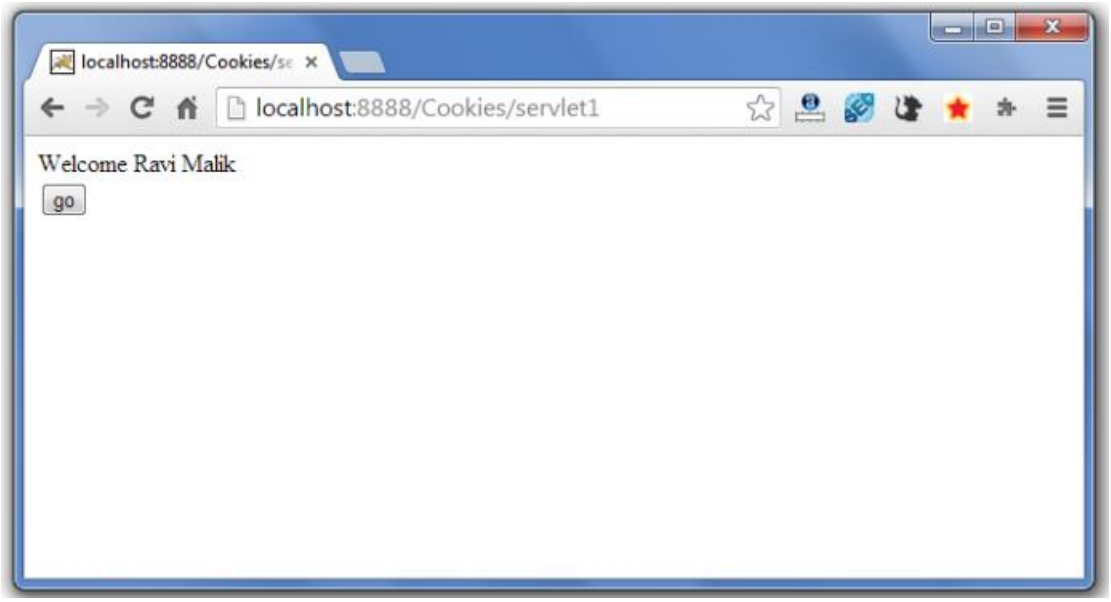
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response){
   try{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   Cookie ck[]=request.getCookies();
   out.print("Hello "+ck[0].getValue());
   out.close();
       }catch(Exception e){System.out.println(e);}
   }
}
```

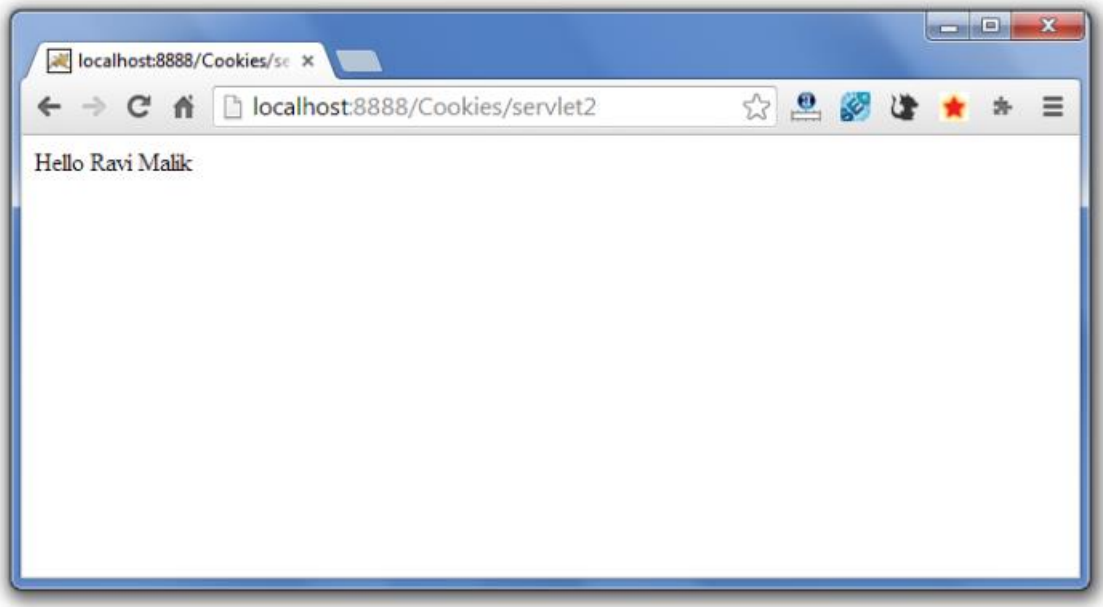
web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
</web-app>
```

OUTPUT-







ADDITIONAL/PRACTICE PROGRAMS-

Q1. Design Servlet Login and Logout using Cookies.
Q2. Create a servlet that prints all the request headers it receives, along with their associated values.
Q3. Create a servlet that recognizes a 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”.

VIVA-VOICE QUESTIONS

Q1. How PrintWriter is different from ServletOutputStream?
Q2. What is the difference between a Generic Servlet and HTTP Servlet?
Q3. What is the use of RequestDispatcher Interface?
Q4. Why is init() method is used in Servlets?

Q5. What is load-on-startup in Servlet?
Q6. What is a WAR file?
Q7. Can you create a Deadlock condition on a servlet?
Q8. Can we fetch the attributes related to a servlet on a different servlet?
Q9. What do you mean by MIME type?
Q10. What is the difference between ServletConfig and ServletContext?

PROGRAM-9

AIM- Develop a small web program using Servlets, JSPs with Database connectivity.

Servlets are mainly used in Dynamic web applications which provides dynamic responses to client requests. In most cases, Dynamic web applications access a database to provide the client requested data. We can use Java standard database connection – JDBC in Servlets to perform database operations.

Servlet – Database connection

A Servlet can generate dynamic HTML by retrieving data from the database and sending it back to the client as a response. We can also update the database based on data passed in the client HTTP request. We will create a simple servlet to fetch/retrieve data from the database based on the client's request. In this example, we will be using Eclipse IDE and PostgreSQL database.

Steps for JDBC connection in Servlets

Install JDBC Driver:

We need to install the appropriate JDBC driver in our program to connect with the database. As we are using PostgreSQL database, install "postgresql.jar" file with the latest version from Maven Repository. After installation, place your jar file under "WebContent -> WEB-INF -> lib" folder.

Import JDBC packages:

To access and process the Database operations, we need to import all the required "java.sql" packages in the program.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

Register the JDBC Driver:

After importing all the packages, we need to register the JDBC driver which we installed into our program. Registering the driver tells the JVM to load the driver's class file into the memory so that we can implement the JDBC operations. We can register the driver by using "Class.forName()" method:

Class.forName():

```
try {
    // Register PostgreSQL Driver
    Class.forName("org.postgresql.Driver");
}
catch (ClassNotFoundException e) {
    System.out.println("Unable to load Driver class");
```

```
        // e.printStackTrace(); OR you
        // can directly print the stack trace
        System.exit(1);
    }
```

Establish the Connection to Database:

Establish the connection to the database using the "DriverManager.getConnection()" method.

```
String URL = "jdbc:postgresql://localhost/postgres";
String USER = "username";
String PASSWORD = "password";
Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
```

As you can see, we need to pass the URL, Username, and Password of the database we are using as parameters to the connection.

- URL: This is the address of the database you are using. In URL, you need to mention the "jdbc:database:hostname:port:databasename".
- Username and Password: You need to specify the username/password set to your database.

Create a JDBC Statement object:

After successful connection, prepare JDBC statement object using the Connection object.

```
Statement stmt = conn.createStatement();
```

Execute the SQL query:

Construct the SQL query based on the client request and execute the query using statement object.

```
stmt.executeQuery(sql);
```

Close Database connection:

After processing the required operations, finally, close all the database connection objects.

```
stmt.close();
conn.close();
```

Example

In this example, we will create

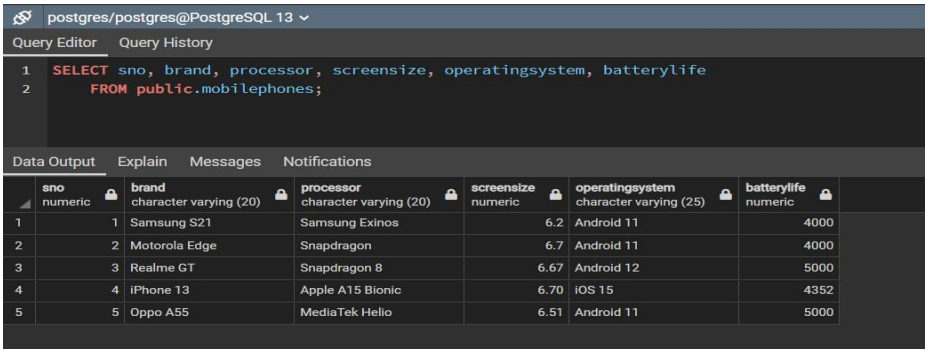
A table in PostgreSQL – to fetch the data from it

An HTML form – for the client access

A Servlet class – to process the client request and generate the response.

PostgreSQL table:

Create a table in the PostgreSQL database and insert some records in it like below.



	sno numeric	brand character varying (20)	processor character varying (20)	screensize numeric	operatingsystem character varying (25)	batterylife numeric
1	1	Samsung S21	Samsung Exinos	6.2	Android 11	4000
2	2	Motorola Edge	Snapdragon	6.7	Android 11	4000
3	3	Realme GT	Snapdragon 8	6.67	Android 12	5000
4	4	iPhone 13	Apple A15 Bionic	6.70	iOS 15	4352
5	5	Oppo A55	MediaTek Helio	6.51	Android 11	5000

PostgreSQL – Table

home.html:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Home Page</title>
</head>
<body>
    <form action="fetch" method="get">
            Fetch Mobile phone details:<input type="submit" value="Search" />
    </form>
</body>
</html>
```

- Create “home.html” file under “WebContent -> home.html”.
- This is the welcome file, where the browser accesses this form page.
- Here we are simply displaying a submit button to fetch the data from the database.
- In the form tag, mentioned the action “fetch” and method “get” to map the URL to a specific servlet class when the form is submitted.
- Create “FetchServlet.java” under “src” folder.

ADDITIONAL/PRACTICE PROGRAMS

Q1. Create User Registration using Jsp, Servlet and Jdbc.
Q2. Create Employee Registration Form using a combination of JSP, Servlet, JDBC and MySQL database.

VIVA-VOICE QUESTIONS

Q1. What is Java Server Template Engines?
Q2. Which methods are used for reading form data using JSP?
Q3. How does JSP processing take place?
Q4. What are the JDBC API components?
Q5. What are the JDBC statements?
Q6. What is the return type of Class.forName() method?
Q7. What are the differences between execute, executeQuery, and executeUpdate?
Q8. What are the differences between ResultSet and RowSet?
Q9. What is the role of the JDBC DriverManager class?
Q10. Which interface is responsible for transaction management in JDBC?
Q11. What are CLOB and BLOB data types in JDBC?
Q12. What are the different types of lockings in JDBC?

8. SAMPLE VIVA-VOICE QUESTIONS

1. How does a JIT compiler differ from a standard compiler?
2. Name the access modifiers in Java.
3. What are constructors in Java?
4. When can you use the super keyword?
5. Explain what synchronization is, using a relevant example.
6. Define a Java Class.
7. What's the difference between this () and super ()?
8. Explain what double brace initialization is and its uses.
9. In simple terms, describe a market interface.
10. What's object cloning in Java?
11. Explain the life cycle of a servlet.
12. What are java objects and java applications?
13. Advantages and disadvantages of Java Sockets.
14. Explain the different ways of using thread?
15. What is the difference between ArrayList and vector?
16. What is an Iterator?
17. What is synchronization and why is it important?
18. What is static in java?
19. What if I do not provide the String array as the argument to the method?
20. Can an application have multiple classes having the main method?
21. Do I need to import java.lang package any time? Why?
22. What makes Java platform independent?
23. Explain the difference between the abstract and final keywords?
24. In Java, what are the differences between heap and stack memory?
25. What do the terms autoboxing and unboxing mean in Java?
26. What are wrapper classes in Java?
27. Can you override a private method or static method in Java?
28. What is the difference between equals() and == in Java?
29. Can you write multiple catch blocks under a single try block?
30. In Java, what are the differences between methods and constructors?
31. What is method hiding?
32. What is a singleton class, and how can it be used?
33. What is reflection and why it is useful?
34. Can == be used on enum ?
35. Is Java pass by reference or pass by value?
36. What is Java Bean ?
37. What is the difference between fail-fast and fail-safe?
38. What is the main difference between StringBuilder and StringBuffer?
39. Why does Java have transient fields?
40. Explain Marshalling and Demarshalling?
41. What exactly is marker interface in Java?
42. What is the difference between Serial and Throughput Garbage collector?
43. When to use LinkedList over ArrayList?
44. Explain the difference between ResultSet Vs. RowSet vs in JDBC?
45. Can you get a null ResultSet?
46. Explain the different drivers of JDBC.
47. Explain which is the most commonly used and fastest JDBC driver.
48. What are the data types used for storing images and files in the database table?
49. Explain what DatabaseMetaData is and why would you use it?
50. Explain the differences between JDBC and ODBC?
51. Explain the term connection pooling.
52. Are there any advantages of using a Prepared Statement in Java?

53. Explain the meaning of hot backup and cold backup.
54. Explain the differences between Statement and PreparedStatement interface?
55. How can we set a null value in JDBC PreparedStatement?
56. Explain the execution of stored procedures using Callable Statement?
57. Mention the functions of the JDBC Connection interface?
58. Why using cookie to store session info is a better idea than just using session info in the request ?
59. Which method of the Cookie class is used to set the maximum age of a cookie?
60. What is singleton session bean?
61. What is Session Facade?
62. Name the attributes of javax.ejb.Stateful.
63. Mention the Java types that can be mapped using the @Lob annotation.
64. Compare Java Beans from Microsoft's Active X Controls?
65. What is a Stored Procedure in JDBC?
66. What is DatabaseMetaData?
67. What is a Prepared Statement?
68. Why JDBC is needed once we have ODBC in hand?
69. Enlist the Declarative Transaction types?
70. Differentiate 'Stateful Session' from 'Entity Bean' ?