



POORNIMA

COLLEGE OF ENGINEERING

ISI-6, RIICO Institutional Area, Sitapura, Jaipur-302022, Rajasthan

Phone/Fax: 0141-2770790-92, www.pce.poornima.org

Advanced Java Lab Manual

(Lab Code: 5CS4-24)

5th Semester, 3rd Year



Department of Computer Engineering

Session: 2023-24

TABLE OF CONTENT

S. No.	Topic/Name of Experiment	Page Number
GENERAL DETAILS		
1	Vision & Mission of Institute and Department	iii
2	RTU Syllabus and Marking Scheme	iv
3	Lab Outcomes and its Mapping with POs and PSOs	v-vii
4	Rubrics of Lab	viii-ix
5	Lab Conduction Plan	x
6	Lab Rotar Plan	xi
7	General Lab Instructions	xii-xiii
8	Lab Specific Safety Rules	xiii
LIST OF EXPERIMENTS (AS PER RTU SYLLABUS)		
1	Zero Lab	1
2	Introduction To Swing, MVC Architecture, Applets, Applications and Pluggable Look and Feel, Basic swing components: Text Fields, Buttons, Toggle Buttons, Checkboxes, and Radio Buttons.	2-5
3	Java database Programming, java.sql Package, JDBC driver, Network Programming With java.net Package, Client and Server Programs, Content And Protocol Handlers.	6-8
4	RMI architecture, RMI registry, Writing distributed application with RMI, Naming services, Naming And Directory Services, Overview of JNDI, Object serialization and Internationalization.	10-11
5	J2EE architecture, Enterprise application concepts, n-tier application concepts, J2EE platform, HTTP protocol, web application, Web containers and Application servers.	12
6	Server side programming with Java Servlet, HTTP and Servlet, Servlet API, life cycle, configuration and context, Request and Response objects, Session handling and event handling, Introduction to filters with writing simple filter application.	13-16
7	JSP architecture, JSP page life cycle, JSP elements, Expression Language, Tag Extensions, Tag Extension API, Tag handlers, JSP Fragments, Tag Files, JSTL, Core Tag library, overview of XML Tag library, SQL Tag library and Functions Tag library.	17-18
8	Beyond the syllabus Exp-1	19-20
9	Beyond the syllabus Exp-2	21-27

INSTITUTE VISION & MISSION

VISION

To create knowledge-based society with scientific temper, team spirit and dignity of labor to face the global competitive challenges.

MISSION

To evolve and develop skill-based systems for effective delivery of knowledge so as to equip young professionals with dedication & commitment to excellence in all spheres of life

DEPARTMENT VISION & MISSION

VISION

Evolve as a center of excellence with wider recognition and to adapt the rapid innovation in Computer Engineering.

MISSION

- To provide a learning-centered environment that will enable students and faculty members to achieve their goals empowering them to compete globally for the most desirable careers in academia and industry.
- To contribute significantly to the research and the discovery of new arenas of knowledge and methods in the rapid developing field of Computer Engineering.
- To support society through participation and transfer of advanced technology from one sector to another.

RTU SYLLABUS AND MARKING SCHEME

5CS4-24: ADVANCED JAVA LAB	
Credit: 1	Max. Marks: 100 (IA:60, ETE:40)
0L+0T+2P	End Term Exam: 2 Hours
S. No.	NAME OF EXPERIMENTS
1	Introduction To Swing, MVC Architecture, Applets, Applications and Pluggable Look and Feel, Basic swing components : Text Fields, Buttons, Toggle Buttons, Checkboxes, and Radio Buttons.
2	Java database Programming, java.sql Package, JDBC driver, Network Programming With java.net Package, Client and Server Programs, Content And Protocol Handlers.
3	RMI architecture, RMI registry, Writing distributed application with RMI, Naming services, Naming And Directory Services, Overview of JNDI, Object serialization and Internationalization.
4	J2EE architecture, Enterprise application concepts, n-tier application concepts, J2EE platform, HTTP protocol, web application, Web containers and Application servers.
5	Server side programming with Java Servlet, HTTP and Servlet, Servlet API, life cycle, configuration and context, Request and Response objects, Session handling and event handling, Introduction to filters with writing simple filter application.
6	JSP architecture, JSP page life cycle, JSP elements, Expression Language, Tag Extensions, Tag Extension API, Tag handlers, JSP Fragments, Tag Files, JSTL, Core Tag library, overview of XML Tag library, SQL Tag library and Functions Tag library.

EVALUATION SCHEME

I+II Mid Term Examination			Attendance and performance			End Term Examination			Total Marks
Experiment	Viva	Total	Attendance	Performance	Total	Experiment	Viva	Total	
30	10	40	10	30	40	30	10	40	100

DISTRIBUTION OF MARKS FOR EACH EXPERIMENT

Attendance	Record	Performance	Total
2	3	5	10

LAB OUTCOME AND ITS MAPPING WITH PO & PSO

LAB OUTCOMES

After completion of this course, students will be able to –

5CS4-24.1	To apply event handling on AWT and Swing components.
5CS4-24.2	To Design a page using Swing , Servlet , JSP and JDBC connectivity.
5CS4-24.3	To create a project based on societal problem.
5CS4-24.4	To map Java classes and object associations to relational database tables with Hibernate mapping files

LO-PO-PSO MAPPING MATRIX OF COURSE

LO/PO/ PSO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
5CS4-24.1	-	-	3	-	-	-	-	-	-	-	-	-	3	-	-
5CS4-24.2	-	-	-	-	3	-	-	-	-	-	-	-	3	-	-
5CS4-24.3	-	-	-	-	-	3	-	-	-	-	-	-	-	3	-
5CS4-24.4	-	-	-	-	-	-	3	-	-	-	-	-	-	3	3

PROGRAM OUTCOMES (POs)

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate

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

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1	The ability to understand and apply knowledge of mathematics, system analysis & design, Data Modelling, Cloud Technology, and latest tools to develop computer based solutions in the areas of system software, Multimedia, Web Applications, Big data analytics, IOT, Business Intelligence and Networking systems.
PSO2	The ability to understand the evolutionary changes in computing, apply standards and ethical practices in project development using latest tools & Technologies to solve societal problems and meet the challenges of the future.
PSO3	The ability to employ modern computing tools and platforms to be an entrepreneur, lifelong learning and higher studies.

RUBRICS FOR LAB

Laboratory Evaluation Rubrics:

S. No.	Criteria	Sub Criteria and Marks Distribution			Outstanding (>90%)	Admirable (70-90%)	Average (40-69%)	Inadequate (<40%)
		Mid-Term	End-Team	Continues Evaluation				
A	PERFORMANCE (PO1, PO8, PO9)	Procedure Followed M.M. 50 = 3 M.M. 75 = 4 M.M. 100 = 6	Procedure Followed M.M. 50 = 3 M.M. 75 = 4 M.M. 100 = 6	Procedure Followed M.M. 50 = 1 M.M. 75 = 2 M.M. 100 = 2	<ul style="list-style-type: none"> All possible system and Input/ Output variables are taken into account Performance measures are properly defined Experimental scenarios are very well defined 	<ul style="list-style-type: none"> Most of the system and Input/ Output variables are taken into account Most of the Performance measures are properly defined Experimental scenarios are defined correctly 	<ul style="list-style-type: none"> Some of the system and Input/ Output variables are taken into account Some of the Performance measures are properly defined Experimental scenarios are defined but not sufficient 	<ul style="list-style-type: none"> System and Input/ Output variables are not defined Performance measures are not properly defined Experimental scenarios not defined
		Individual/Team Work M.M. 50 = 3 M.M. 75 = 4 M.M. 100 = 6	Individual/Team Work M.M. 50 = 3 M.M. 75 = 4 M.M. 100 = 6	Individual/Team Work M.M. 50 = 1 M.M. 75 = 2 M.M. 100 = 2	<ul style="list-style-type: none"> Coordination among the group members in performing the experiment was excellent 	<ul style="list-style-type: none"> Coordination among the group members in performing the experiment was good 	<ul style="list-style-type: none"> Coordination among the group members in performing the experiment was average 	<ul style="list-style-type: none"> Coordination among the group members in performing the experiment was very poor
		Precision in data collection M.M. 50 = 3 M.M. 75 = 4 M.M. 100 = 6	Precision in data collection M.M. 50 = 3 M.M. 75 = 4 M.M. 100 = 6	Precision in data collection M.M. 50 = 2 M.M. 75 = 2 M.M. 100 = 4	<ul style="list-style-type: none"> Data collected is correct in size and from the experiment performed 	<ul style="list-style-type: none"> Data collected is appropriate in size and but not from proper sources. 	<ul style="list-style-type: none"> Data collected is not so appropriate in size and but from proper sources. 	<ul style="list-style-type: none"> Data collected is neither appropriate in size and nor from proper sources
B	LAB RECORD/WRITTEN WORK (PO1, PO8, PO10)	NA	NA	Timing of Evaluation of Experiment M.M. 50 = 3 M.M. 75 = 4 M.M. 100 = 6	<ul style="list-style-type: none"> On the Same Date of Performance 	<ul style="list-style-type: none"> On the Next Turn from Performance 	<ul style="list-style-type: none"> Before Dead Line 	<ul style="list-style-type: none"> On the Dead Line
		Data Analysis M.M. 50 = 3 M.M. 75 = 5 M.M. 100 = 6	Data Analysis M.M. 50 = 3 M.M. 75 = 5 M.M. 100 = 6	Data Analysis M.M. 50 = 2 M.M. 75 = 3 M.M. 100 = 4	<ul style="list-style-type: none"> Data collected is exhaustively analyzed & appropriate features are selected 	<ul style="list-style-type: none"> Data collected is analyzed & but appropriate features are not selected 	<ul style="list-style-type: none"> Data collected is not analyzed properly. •Features selected are not appropriate 	<ul style="list-style-type: none"> Data collected is not analyzed & the features are not selected

		<p>Results and Discussion</p> <p>M.M. 50 = 3 M.M. 75 = 5 M.M. 100 = 6</p>	<p>Results and Discussion</p> <p>M.M. 50 = 3 M.M. 75 = 5 M.M. 100 = 6</p>	<p>Results and Discussion</p> <p>M.M. 50 = 2 M.M. 75 = 3 M.M. 100 = 4</p>	<ul style="list-style-type: none"> • All results are very well presented with all variables • Well prepared neat diagrams/plots/ tables for all performance measured • Discussed critically behavior of the system with reference to performance measures • Very well discussed pros n cons of outcome 	<ul style="list-style-type: none"> • All results presented but not all variables mentioned • Prepared diagrams /plots/ tables for all performance measured but not so neat • Discussed behavior of the system with reference to performance measures but not critical • Discussed pros n cons of outcome in brief 	<ul style="list-style-type: none"> • Partial results are included • Prepared diagrams /plots/ tables partially for the performance measures • Behavior of the system with reference to performance measures has been superficially presented • Discussed pros n cons of outcome but not so relevant 	<ul style="list-style-type: none"> • Results are included but not as per experimental scenarios • No proper diagrams /plots/ tables are prepared • Behavior of the system with reference to performance measures has not been presented • Did not discuss pros n cons of outcome
C	VIVA (PO1, PO10)	<p>Way of presentation</p> <p>M.M. 50 = 2.5 M.M. 75 = 4 M.M. 100 = 5</p>	<p>Way of presentation</p> <p>M.M. 50 = 2.5 M.M. 75 = 4 M.M. 100 = 5</p>	<p>Way of presentation</p> <p>M.M. 50 = 2 M.M. 75 = 3 M.M. 100 = 4</p>	<ul style="list-style-type: none"> • Presentation was very good 	<ul style="list-style-type: none"> • Presentation was good 	<ul style="list-style-type: none"> • Presentation was satisfactory 	<ul style="list-style-type: none"> • Presentation was poor
		<p>Concept Explanation</p> <p>M.M. 50 = 2.5 M.M. 75 = 4 M.M. 100 = 5</p>	<p>Concept Explanation</p> <p>M.M. 50 = 2.5 M.M. 75 = 4 M.M. 100 = 5</p>	<p>Concept Explanation</p> <p>M.M. 50 = 2 M.M. 75 = 3 M.M. 100 = 4</p>	<ul style="list-style-type: none"> • Conceptual explanation was excellent 	<ul style="list-style-type: none"> • Conceptual explanation was good 	<ul style="list-style-type: none"> • Conceptual explanation was somewhat good 	<ul style="list-style-type: none"> • Conceptual explanation was Poor
D	ATTENDANCE	NA	NA	<p>Attendance</p> <p>M.M. 50 = 5 M.M. 75 = 8 M.M. 100 = 10</p>	<ul style="list-style-type: none"> • Present more than 90% of lab sessions 	<ul style="list-style-type: none"> • Present more than 75% of lab sessions 	<ul style="list-style-type: none"> • Present more than 60% of lab sessions 	<ul style="list-style-type: none"> • Present in less than 60% lab sessions

LAB CONDUCTION PLAN

Total number of Experiments - 07

Total number of turns required -07

Number of turns required for: -

Experiment Number	Scheduled Week
Experiment -1	Week 1
Experiment -2	Week 2
Experiment -3	Week 3
I Mid Term	Week 4
Experiment -4	Week 5
Experiment -5	Week 6
Experiment -6	Week 7
II Mid Term	Week 8

DISTRIBUTION OF LAB HOURS

S. No.	Activity	Distribution of Lab Hours	
		Time (180 minute)	Time (120 minute)
1	Attendance	5	5
2	Explanation of Experiment & Logic	30	30
3	Performing the Experiment	60	30
4	File Checking	40	20
5	Viva/Quiz	30	20
6	Solving of Queries	15	15

LAB ROTAR PLAN**ROTOR-1**

Ex. No.	NAME OF EXPERIMENTS
1	Introduction To Swing, MVC Architecture, Applets, Applications and Pluggable Look and Feel, Basic swing components : Text Fields, Buttons, Toggle Buttons, Checkboxes, and Radio Buttons.
2	Java database Programming, java.sql Package, JDBC driver, Network Programming With java.net Package, Client and Server Programs, Content And Protocol Handlers.
3	RMI architecture, RMI registry, Writing distributed application with RMI, Naming services, Naming And Directory Services, Overview of JNDI, Object serialization and Internationalization.

ROTOR-2

Ex. No.	NAME OF EXPERIMENTS
4	J2EE architecture, Enterprise application concepts, n-tier application concepts, J2EE platform, HTTP protocol, web application, Web containers and Application servers.
5	Server side programming with Java Servlet, HTTP and Servlet, Servlet API, life cycle, configuration and context, Request and Response objects, Session handling and event handling, Introduction to filters with writing simple filter application.
6	JSP architecture, JSP page life cycle, JSP elements, Expression Language, Tag Extensions, Tag Extension API, Tag handlers, JSP Fragments, Tag Files, JSTL, Core Tag library, overview of XML Tag library, SQL Tag library and Functions Tag library.

GENERAL LAB INSTRUCTIONS

DO'S

1. Enter the lab on time and leave at proper time.
2. Wait for the previous class to leave before the next class enters.
3. Keep the bag outside in the respective racks.
4. Utilize lab hours in the corresponding.
5. Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
6. Leave the labs at least as nice as you found them.
7. If you notice a problem with a piece of equipment (e.g., a computer doesn't respond) or the room in general (e.g., cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.

DON'TS

1. Don't abuse the equipment.
2. Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
3. Do not attempt to reboot a computer. Report problems to lab staff.
4. Do not remove or modify any software or file without permission.
5. Do not remove printers and machines from the network without being explicitly told to do so by lab staff.
6. Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, log out before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
7. Don't use internet, internet chat of any kind in your regular lab schedule.
8. Do not download or upload of MP3, JPG or MPEG files.
9. No games are allowed in the lab sessions.
10. No hardware including USB drives can be connected or disconnected in the labs without

prior permission of the lab in-charge.

11. No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
12. Don't bring any external material in the lab, except your lab record, copy and books.
13. Don't bring the mobile phones in the lab. If necessary, then keep them in silence mode.
14. Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.
15. If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.

LAB SPECIFIC SAFETY RULES

Before entering in the lab

1. All the students are supposed to prepare the theory regarding the next experiment/ Program.
2. Students are supposed to bring their lab records as per their lab schedule.
3. Previous experiment/program should be written in the lab record.
4. If applicable trace paper/graph paper must be pasted in lab record with proper labeling.
5. All the students must follow the instructions, failing which he/she may not be allowed in the lab.

While working in the lab

1. Adhere to experimental schedule as instructed by the lab in-charge/faculty.
2. Get the previously performed experiment/ program signed by the faculty/ lab in charge.
3. Get the output of current experiment/program checked by the faculty/ lab in charge in the lab copy.
4. Each student should work on his/her assigned computer at each turn of the lab.
5. Take responsibility of valuable accessories.

Zero Lab

Advance Java

It is a part of Java programming language. It is an advanced technology or advance version of Java specially designed to develop web-based, network-centric or enterprise applications. It includes the concepts like Servlet, JSP, JDBC, RMI, Socket programming, etc. It is a specialization in specific domain.

Most of the applications developed using advance Java uses tow-tier architecture i.e. Client and Server. All the applications that runs on Server can be considered as advance Java applications.

Why advance Java?

It simplifies the complexity of a building n-tier application.

- Standardizes and API between components and application sever container.
- JEE application Server and Containers provides the framework services.
- Benefits of Advance Java

The four major benefits of advance Java that are, network centric, process simplification, and futuristic imaging standard.

JEE (advance Java) provides libraries to understand the concept of Client-Server architecture for web- based applications.

We can also work with web and application servers such as Apache Tomcat and Glassfish Using these servers, we can understand the working of HTTP protocol. It cannot be done in core Java. It is also important understand the advance Java if you are dealing with trading technologies like Hadoop, cloud-native and data science.

It provides a set of services, API and protocols, that provides the functionality which is necessary for developing multi-tiered application, web-based application.

There is a number of advance Java frameworks like, Spring, Hibernate, Struts, that enables us to develop secure transaction-based web applications such as banking application, inventory management application.

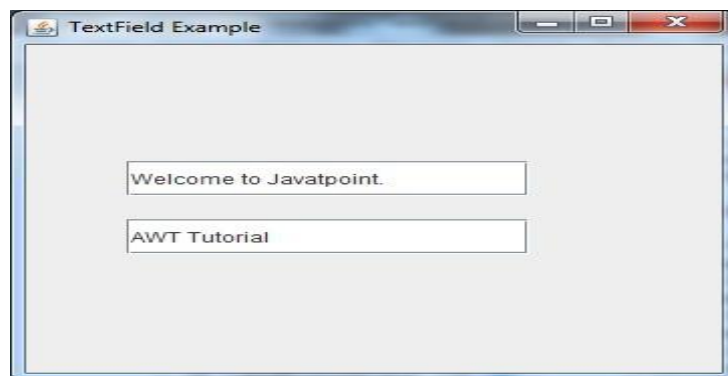
EXPERIMENT-1

OBJECTIVE

- a) Explain Swing in JAVA
 - b) Describe Basic swing components : Text Fields, Buttons, Toggle Buttons, Checkboxes, and Radio Buttons.
- (a) **Swing:** **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.

(b) Text Field:

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Welcome to Javatpoint.");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("AWT Tutorial");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



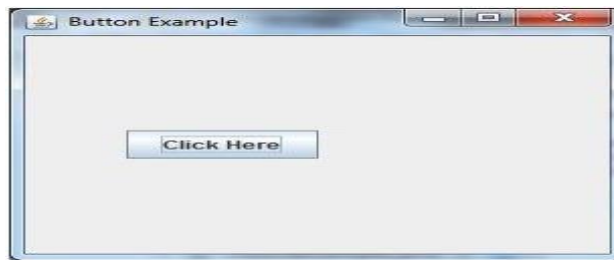
(a) Buttons:

```
import javax.swing.*;
```

```

public class ButtonExample {
    public static void main(String[] args) {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}

```

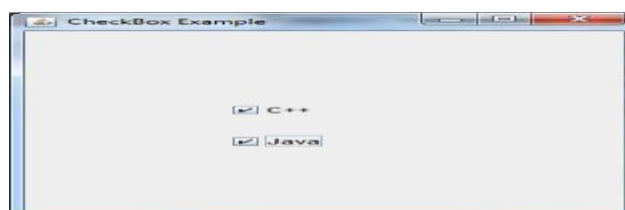


(c) CheckBox:

```

public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckBoxExample();
    }
}

```



(a) Radiobutton

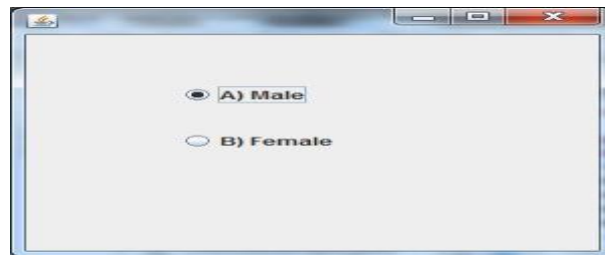
```

import javax.swing.*;

```



```
public class RadioButtonExample {
    JFrame f;
    RadioButtonExample(){
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);bg.add(r2);
        f.add(r1);f.add(r2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new RadioButtonExample();
    }
}
```

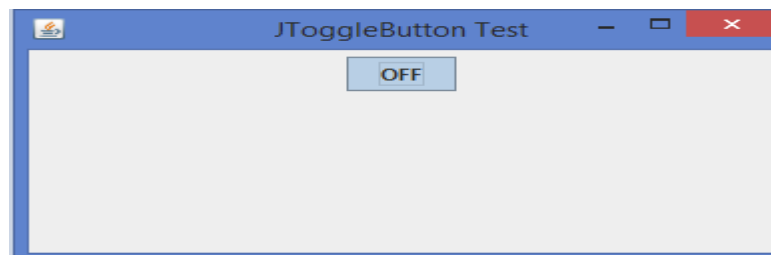


(a) Toggle Button

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class JToggleButtonTest extends JFrame implements ItemListener {
    private JToggleButton jtb;
    JToggleButtonTest() {
        setTitle("JToggleButton Test");
        setLayout(new FlowLayout());
        setJToggleButton();
        setAction();
        setSize(450, 300);
        setLocationRelativeTo(null);
    }
}
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    private void setJToggleButton() {
        jtb = new JToggleButton("ON");
        add(jtb);
    }
    private void setAction() {
        jtb.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent eve) {
        if(jtb.isSelected())
            jtb.setText("OFF");
        else
            jtb.setText("ON");
    }
}

public class MainApp {
    public static void main(String[] args) {
        new JToggleButtonTest();
    }
}
```



EXPERIMENT-2

OBJECTIVE

- (a) Explain MVC Architecture
- (b) Write program using Model View Controller Architecture.

- (a) **The Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application.

```
class Student
{
    private String rollNo;
    private String name;

    public String getRollNo()
    {
        return rollNo;
    }

    public void setRollNo(String rollNo)
    {
        this.rollNo = rollNo;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }
}

class StudentView
{
    public void printStudentDetails(String studentName, String
studentRollNo)
    {
        System.out.println("Student: ");
        System.out.println("Name: " + studentName);
        System.out.println("Roll No: " + studentRollNo);
    }
}
```

```
    }  
}  
class StudentController  
{  
    private Student model;  
    private StudentView view;  
    public StudentController(Student model, StudentView view)  
    {  
        this.model = model;  
        this.view = view;  
    }  
    public void setStudentName(String name)  
    {  
        model.setName(name);  
    }  
    public String getStudentName()  
    {  
        return model.getName();  
    }  
    public void setStudentRollNo(String rollNo)  
    {  
        model.setRollNo(rollNo);  
    }  
    public String getStudentRollNo()  
    {  
        return model.getRollNo();  
    }  
    public void updateView()  
    {  
        view.printStudentDetails(model.getName(),  
model.getRollNo());  
    }  
}  
class MVCPattern  
{  
    public static void main(String[] args)  
    {
```

```
        Student model = retrieveStudentFromDatabase();
        StudentView view = new StudentView();
        StudentController controller = new
        StudentController(model, view);
            controller.updateView();
            controller.setStudentName("Merry");
            controller.updateView();
    }
    private static Student retrieveStudentFromDatabase()
    {
        Student student = new Student();
        student.setName("Alex");
        student.setRollNo("15UCS157");
        return student;
    } }
```

Output:

Student:

Name: Alex

Roll No: 15UCS157

Student:

Name: Merry

Roll No: 15UCS157

EXPERIMENT-3

OBJECTIVE

- (a) Explain Remote Method Invocation
- (b) Write program using concept of Remote Method Invocation.

(a) Remote Method Invocation:

- (b) Create the remote interface

```
import java.rmi.*;
public interface Adder extends Remote{
    public int add(int x,int y)throws RemoteException;
}
```

Provide the implementation of the remote interface

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
    AdderRemote()throws RemoteException{
        super();
    }
    public int add(int x,int y){return x+y;}
}
```

create the stub and skeleton objects using the rmic tool.

```
rmic AdderRemote
```

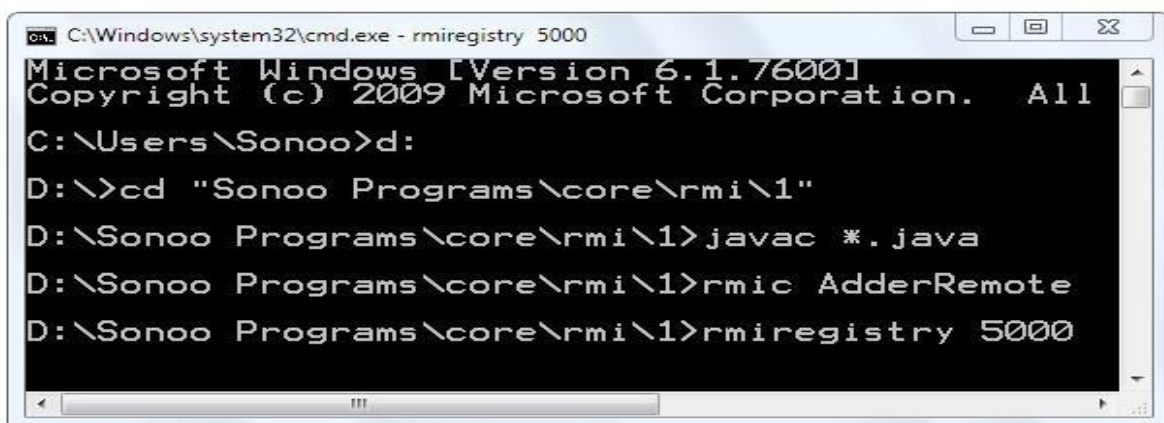
create and run the Server application

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
    public static void main(String args[]){
        try{
            Adder stub=new AdderRemote();
```

```
Naming.rebind("rmi://localhost:5000/sonoo",stub);  
}catch(Exception e){System.out.println(e);}  
}  
}
```

Create and run the client application

```
import java.rmi.*;  
public class MyClient{  
    public static void main(String args[]){  
        try{  
            Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");  
            System.out.println(stub.add(34,4));  
        }catch(Exception e){}  
    }  
}
```



```
C:\Windows\system32\cmd.exe - rmiregistry 5000  
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
C:\Users\Sonoo>d:  
D:\>cd "Sonoo Programs\core\rmi\1"  
D:\Sonoo Programs\core\rmi\1>javac *.java  
D:\Sonoo Programs\core\rmi\1>rmic AdderRemote  
D:\Sonoo Programs\core\rmi\1>rmiregistry 5000
```



```
C:\Windows\system32\cmd.exe - java MyServer  
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
C:\Users\Sonoo>d:  
D:\>cd "Sonoo Programs\core\rmi\1"  
D:\Sonoo Programs\core\rmi\1>java MyServer
```

EXPERIMENT-4

OBJECTIVE

- (a) Explain JNDI, Object Serialization and Internationalization
- (b) Write program using concept of Serialization and Internationalization

(a) JNDI (Java Naming and Directory Interface) enables Java platform-based applications to access multiple naming and directory services. Part of the Java Enterprise application programming interface (API) set, JNDI makes it possible for developers to create portable applications that are enabled for a number of different naming and directory services, including: file systems; directory services such as Lightweight Directory Access Protocol (LDAP), Novell Directory Services, and Network Information System (NIS); and distributed object systems such as the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), and Enterprise JavaBeans (EJB).

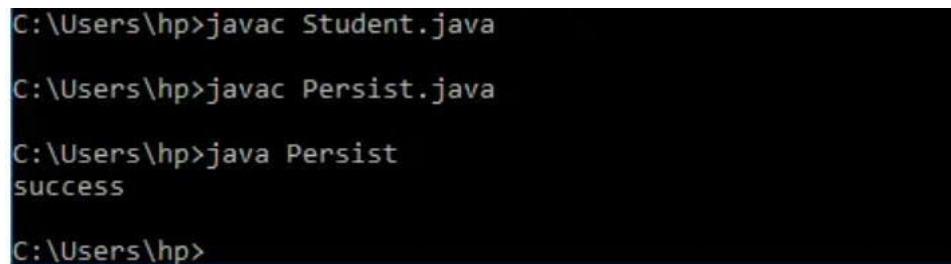
Serialization: Serialization is a mechanism of converting the state of an object into a byte stream. To make a Java object serializable we implement the `java.io.Serializable` interface. The `ObjectOutputStream` class contains `writeObject()` method for serializing an `Object`. For serializing the object, we call the `writeObject()` method of `ObjectOutputStream`, and for deserialization we call the `readObject()` method of `ObjectInputStream` class.

Internationalization: the process of designing web applications in such a way that which provides support for various countries, various languages and various currency automatically without performing any change in the application is called Internationalization (I18N). It is known as I18N because between I and N there is 18 characters that's why I18N. We can implement Internationalization by using the following 3 classes:

- `Locale`
- `NumberFormat`
- `DateFormat`

(b) **Student.java**
 `import java.io.Serializable;`
 `public class Student implements Serializable{`
 `int id;`
 `String name;`
 `public Student(int id, String name) {`


```
this.id = id;
this.name = name;
}
}
Persist.java
import java.io.*;
class Persist{
    public static void main(String args[]){
    try{
    Student s1 =new Student(123,"Ankit");
    FileOutputStream fout=new FileOutputStream("abc.txt");
    ObjectOutputStream out=new ObjectOutputStream(fout);
    out.writeObject(s1);
    out.flush();
    out.close();
    System.out.println("success");
    }catch(Exception e)
    {System.out.println(e);}
    }
}
```



```
C:\Users\hp>javac Student.java
C:\Users\hp>javac Persist.java
C:\Users\hp>java Persist
success
C:\Users\hp>
```

Internationalization:

```
import java.util.*;
import java.text.*;

class NumberFormatDemo {

    public static void main(String[] args)

    {

        double d = 123456.789;

        NumberFormat nf = NumberFormat.getInstance(Locale.ITALY);

        System.out.println("ITALY representation of " + d + " : " + nf.format(d));

    }

}
```

Output: ITALY representation of 123456.789 : 123.456,789

EXPERIMENT-5

OBJECTIVE

- (a) Network Programming With java.net Package, Client and Server Programs, Content And Protocol Handlers.
- (b) write program using using java.net Package for client and server program.

(a) The term network programming or networking associates with writing programs that can be executed over various computer devices, in which all the devices are connected to each other to share resources using a network.

(b) A socket is one endpoint of a two-way communication link between two programs running on the network. The socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. In java socket programming example tutorial, we will learn how to write **java socket server** and **java socket client** program. We will also learn how server client program read and write data on the socket. **java.net.Socket** and **java.net.ServerSocket** are the java classes that implements Socket and Socket server.

© Java Socket Server Example:

```
package com.journaldev.socket;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.ClassNotFoundException;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * This class implements java Socket server
 * @author pankaj
 *
 */
public class SocketServerExample {
```

```
//static ServerSocket variable
private static ServerSocket server;

//socket server port on which it will listen
private static int port = 9876;

public static void main(String args[]) throws IOException,
ClassNotFoundException{
    //create the socket server object
    server = new ServerSocket(port);
    //keep listens indefinitely until receives 'exit' call or program terminates
    while(true){
        System.out.println("Waiting for the client request");
        //creating socket and waiting for client connection
        Socket socket = server.accept();
        //read from socket to ObjectInputStream object
        ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
        //convert ObjectInputStream object to String
        String message = (String) ois.readObject();
        System.out.println("Message Received: " + message);
        //create ObjectOutputStream object
        ObjectOutputStream oos = new
ObjectOutputStream(socket.getOutputStream());
        //write object to Socket
        oos.writeObject("Hi Client "+message);
        //close resources
        ois.close();
        oos.close();
        socket.close();
        //terminate the server if client sends exit request
        if(message.equalsIgnoreCase("exit")) break;
    }
    System.out.println("Shutting down Socket server!!");
    //close the ServerSocket object
    server.close();
}
```

```
}  
}
```

Java Socket Client Example:

```
package com.journaldev.socket;  
    import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.InetAddress;  
import java.net.Socket;  
import java.net.UnknownHostException;  
  
/**  
 * This class implements java socket client  
 * @author pankaj  
 *  
 */  
public class SocketClientExample {  
  
    public static void main(String[] args) throws UnknownHostException,  
IOException, ClassNotFoundException, InterruptedException{  
        //get the localhost IP address, if server is running on some other IP, you need to  
use that  
        InetAddress host = InetAddress.getLocalHost();  
        Socket socket = null;  
        ObjectOutputStream oos = null;  
        ObjectInputStream ois = null;  
        for(int i=0; i<5;i++){  
            //establish socket connection to server  
            socket = new Socket(host.getHostName(), 9876);  
            //write to socket using ObjectOutputStream  
            oos = new ObjectOutputStream(socket.getOutputStream());  
            System.out.println("Sending request to Socket Server");  
            if(i==4)oos.writeObject("exit");  
        }  
    }  
}
```

```
        else oos.writeObject(""+i);  
        //read the server response message  
        ois = new ObjectInputStream(socket.getInputStream());  
  
        String message = (String) ois.readObject();  
        System.out.println("Message: " + message);  
        //close resources  
        ois.close();  
        oos.close();  
        Thread.sleep(100);  
    }  
}  
}
```

Output:

output of java socket server program

```
Waiting for the client request  
Message Received: 0  
Waiting for the client request  
Message Received: 1  
Waiting for the client request  
Message Received: 2  
Waiting for the client request  
Message Received: 3  
Waiting for the client request  
Message Received: exit  
Shutting down Socket server!!
```

output of java socket Client program

```
Sending request to Socket Server  
Message: Hi Client 0  
Sending request to Socket Server  
Message: Hi Client 1  
Sending request to Socket Server  
Message: Hi Client 2  
Sending request to Socket Server  
Message: Hi Client 3  
Sending request to Socket Server  
Message: Hi Client exit
```

EXPERIMENT-6

OBJECTIVE

- (a) Explain java database connectivity with concept of JDBC driver?
- (b) Write program using Java database connectivity.

JDBC driver: JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

Types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

(b) The steps for connecting to a database with JDBC are as follows:

- (i) Install or locate the database you want to access.
- (ii) Include the JDBC library.
- (iii) Ensure the JDBC driver you need is on your classpath.
- (iv) Use the JDBC library to obtain a connection to the database.
- (v) Use the connection to issue SQL commands.
- (vi) Close the connection when you are finished.

How to set the temporary classpath for MySQL

Open command prompt and write:

Set classpath=C:\Program Files (x86)\Java\jre-1.8\lib\ext\mysql-connector-java-8.0.27.jar

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysql-connector-java-8.0.27.jar;; as C:\Program Files (x86)\Java\jre-1.8\lib\ext\mysql-connector-java-8.0.27.jar

```
import java.sql.*;
class MySqlQuery{
public static void main(String args[])
{
try
{
Class.forName("com.mysql.cj.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","root");
//here mydb is database name, root is username and password
Statement stmt=con.createStatement();
System.out.println("Connected");
ResultSet rs=stmt.executeQuery("select * from student");
while(rs.next())
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getInt(3));
con.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

BEYOND THE SYLLABUS EXPERIMENT-1**OBJECTIVE**

J2EE Architecture, Enterprise application concepts, n-tier application concepts, J2EE Platform.

J2EE – Java 2 Enterprise Edition is a platform that provides an environment to develop enterprise applications using multitier architecture. This technology is highly intensified provides developer scalable, efficient and faster solutions for information management. This improves the productivity of development and standards for deploying enterprise applications.

J2EE uses three architecture:

Client Tier: Client tier consists of user programs that interact with the user for request and response. Clients can be classified as a Web Client and Application Client.

- i) **Web client** consists of dynamic web pages of various mark-up languages that are generated by web components running in web tier or web browser which renders pages received from the server. Web pages received from web tier embedded an Applet these run on a web browser. Web components are APIs for creating a web client program. Web components enable the user to design cleaner and more modular applications. They provide a way to separate application programming.

- ii) **The application client** runs on the client machine and handles the tasks that give richer user interfaces. Typically the GUI is created by Swings or AWT. Application clients can directly access EJBs running in the business tier using an HTTP connection.

iii) **Middle Tier:** Middle tier usually contains enterprise beans and web services that distribute business logic for the applications.

iv) **Web Tier /Web Component :** Web components can be servlets or JSP pages. Servlets can dynamically process the request and generate the responses. Compared to JSP and servlets – servlets are dynamic pages to some extent but JSP pages are static in nature. During application assembly process Client's Static HTML programs and applet, codes are bundled in web tier/ Web Component. Actually these HTML and applets are not considered as elements of web components. Server-side utility classes are also bundled with web component but they are not considered as web components. Web tier might include EJB components for processing user inputs and sends the input to Enterprise bean running in the business tier.

V) **EJB Tier /EJB Component :** Enterprise components handle usually business code that is logic to solve particular business domains such as banking or finance are handled by enterprise bean running in the business tier. Enterprise Container receives data from client processes if necessary, sends it to the enterprise information system for storage. Enterprise bean also retrieves data from storage, processes it and sends it back to the client.

Enterprise Data Tier: Enterprise data is stored in a relational database. This tier contains containers, components and services. This tier consists of database servers, enterprise resource planning systems and other data sources. Resources are typically located on a separate machine than the J2EE Server and accessed by components on the business tier.

Technologies used in EIS Tier:

- Java Database Connectivity API (JDBC).
- Java Persistence API.
- Java Connector Architecture.
- Java Transaction API.

BEYOND THE SYLLABUS EXPERIMENT-2

OBJECTIVE

- (a) Explain JSP architecture and its life cycle
- (b) write program using JSP.

(a) Java Server Pages or as is normally called JSP is a Java based technology that simplifies the developing of dynamic web sites. JSP pages are HTML pages with embedded code that allows to access data from Java code running on the server. Java Server Pages are part of a 3-tier architecture. A server(generally referred to as application or web server) supports the Java Server Pages. This server will act as a mediator between the client browser and a database.

A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle.

- Compilation
- Initialization
- Execution
- Cleanup

JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps –

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the `jspInit()` method before servicing any requests. If you need to perform JSP-specific initialization, override the `jspInit()` method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed. Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the `_jspService()` method in the JSP. The

_jspService() method takes an HttpServletRequest and an HttpServletResponse as its parameters as follows

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container. The jspDestroy() method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

(b) Simple JSP Example with Eclipse and Tomcat

We can use Eclipse IDE for building dynamic web project with JSPs and use Tomcat to run it. Please read [Java Web Applications](/community/tutorials/java-web-application-tutorial-for-beginners#first-web-app-servlet) tutorial to learn how can we easily create JSPs in Eclipse and run it in tomcat. A simple JSP example pageexample is: `home.jsp`

...

```
<% @ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">

<title>First JSP</title>

</head>

<% @ page import="java.util.Date" %>

<body>

<h3>Hi Pankaj</h3><br>

<strong>Current Time is</strong>: <%=new Date() %>

</body>

</html>
```

If you have a simple JSP that uses only JRE classes, we are not required to put it as WAR file. Just create a directory in the tomcat webapps folder and place your JSP file in the newly created directory. For example, if your JSP is located at apache-tomcat/webapps/test/home.jsp, then you can access it in browser with URL `https://localhost:8080/test/home.jsp`. If your host and port is different, then you need to make changes in URL accordingly.

JSP Files location in Web Application WAR File

We can place JSP files at any location in the WAR file, however if we put it inside the WEB-INF directory, we won't be able to access it directly from client. We can configure JSP just like servlets in web.xml, for example if I have a JSP example page like below inside WEB-INF directory: `test.jsp`

```

...
<% @ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Test JSP</title>
</head>
<body>
Test JSP Page inside WEB-INF folder.<br>
Init Param "test" value =<%=config.getInitParameter("test") %><br>
HashCode of this object=<%=this.hashCode() %>
</body>
</html>
...

```

And I configure it in web.xml configuration as:

```

...
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xmlns="https://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="https://java.sun.com/xml/ns/javaee
    https://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <display-name>FirstJSP</display-name>

    <servlet>
        <servlet-name>Test</servlet-name>
        <jsp-file>/WEB-INF/test.jsp</jsp-file>
        <init-param>
            <param-name>test</param-name>
            <param-value>Test Value</param-value>
        </init-param>
    </servlet>

```

```

<servlet-mapping>
<servlet-name>Test</servlet-name>
<url-pattern>/Test.do</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>Test1</servlet-name>
<jsp-file>/WEB-INF/test.jsp</jsp-file>
</servlet>

<servlet-mapping>
<servlet-name>Test1</servlet-name>
<url-pattern>/Test1.do</url-pattern>
</servlet-mapping>
</web-app>

```

Then I can access it with both the URLs <https://localhost:8080/FirstJSP/Test.do> and <https://localhost:8080/FirstJSP/Test1.do> Notice that container will create two instances in this case and both will have their own servlet config objects, you can confirm this by visiting these URLs in browser. For Test.do URI, you will get response like below.

```

Test JSP Page inside WEB-INF folder.
Init Param "test" value =Test Value
HashCode of this object=1839060256

```

For Test1.do URI, you will get response like below.

```

Test JSP Page inside WEB-INF folder.
Init Param "test" value =null
HashCode of this object=38139054

```

Notice the init param value in second case is null because it's not defined for the second servlet, also notice the hashcode is different. If you will make further requests, the hashcode value will not change because the requests are processed by spawning a new thread by the container. Did you noticed the use of **config** variable in above JSP example but there is no variable declared, it's because its one of the 9 implicit objects available in JSP page, read more about them at [**JSP Implicit Objects**]([community/tutorials/jsp-implicit-objects](https://community.tutorials/jsp-implicit-objects) "JSP Implicit Objects with Examples").

JSP transformed Servlet Source Code and Class File location in Tomcat

Once JSP files are translated to Servlet source code, the source code (.java) and

compiled classes both are place in

****Tomcat/work/Catalina/localhost/FirstJSP/org/apache/jsp**** directory. If the JSP files are inside other directories of application, the directory structure is maintained. For JSPs inside WEB-INF directory.

```

/*
 * Generated by the Jasper component of Apache Tomcat
 * Version: Apache Tomcat/7.0.32
 * Generated at: 2013-08-21 03:40:59 UTC
 * Note: The last modified time of this file was set to
 *       the last modified time of the source file after
 *       generation to assist with modification tracking.
 */
package org.apache.jsp.WEB_002dINF;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class test_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    private static final javax.servlet.jsp.JspFactory _jspxFactory =
        javax.servlet.jsp.JspFactory.getDefaultFactory();

    private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

    private javax.el.ExpressionFactory _el_expressionfactory;
    private org.apache.tomcat.InstanceManager _jsp_instancemanager;

    public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
        return _jspx_dependants;
    }

    public void _jspInit() {
        _el_expressionfactory =
            _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
        _jsp_instancemanager =
            org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
    }

    public void _jspDestroy() {
    }

    public void _jspService(final javax.servlet.http.HttpServletRequest request, final
        javax.servlet.http.HttpServletResponse response)
        throws java.io.IOException, javax.servlet.ServletException {

```

```

final javax.servlet.jsp.PageContext pageContext;
javax.servlet.http.HttpSession session = null;
final javax.servlet.ServletContext application;
final javax.servlet.ServletConfig config;
javax.servlet.jsp.JspWriter out = null;
final java.lang.Object page = this;
javax.servlet.jsp.JspWriter _jspx_out = null;
javax.servlet.jsp.PageContext _jspx_page_context = null;

try {
    response.setContentType("text/html; charset=US-ASCII");
    pageContext = _jspxFactory.getPageContext(this, request, response,
        null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    out.write("\n");
    out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01
Transitional//EN\" \"https://www.w3.org/TR/html4/loose.dtd\">\n");
    out.write("<html>\n");
    out.write("<head>\n");
    out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=US-
ASCII\">\n");
    out.write("<title>Test JSP</title>\n");
    out.write("</head>\n");
    out.write("<body>\n");
    out.write("Test JSP Page inside WEB-INF folder.<br>\n");
    out.write("Init Param \"test\" value =");
    out.print(config.getInitParameter("test") );
    out.write("<br>\n");
    out.write("HashCode of this object=");
    out.print(this.hashCode() );
    out.write("\n");
    out.write("</body>\n");
    out.write("</html>");
} catch (java.lang.Throwable t) {
    if (!(t instanceof javax.servlet.jsp.SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            try { out.clearBuffer(); } catch (java.io.IOException e) {}
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
        else throw new ServletException(t);
    }
} finally {
    _jspxFactory.releasePageContext(_jspx_page_context);
}

```

```
}  
}  
}  
...
```

Notice following points in above servlet code;

- The package of class starts with org.apache.jsp and if JSPs are inside other folders, it includes directory hierarchy too. Usually we dont care about it.
- The generated servlet class is final and can't be extended.
- It extends `org.apache.jasper.runtime.HttpJspBase` that is similar to HttpServlet except that it's internal to Tomcat JSP Translator implementation. HttpJspBase extends HttpServlet and implements HttpJspPage interface.