Department of Computer Science and Engineering

Rubrics for Lab Assessment

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

INDEX

Name: Vedant Nagar

Enrollment number : 03614812721

Branch: Computer Science and Technology

Group: 6FSD-2c

S. No.	Experiment	Date of Performance	Date of Checking	Signature	Remark
1					
2					
3					
4					
5					
6					
7					
8					

9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					

20					
21					
22					
23					
24					
25					
26					

PROGRAM-1

Aim: Write a program to implement parametrized constructor in Java.

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.

Overview: A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

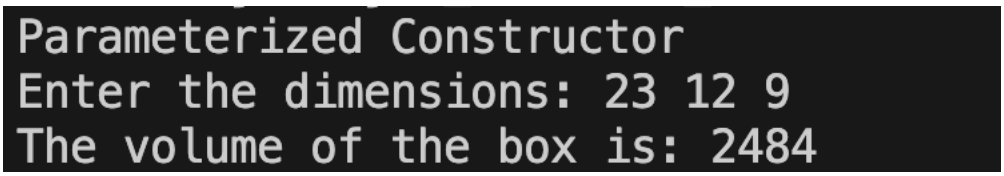
Syntax of Parameterized Constructor in Java:

```
class ClassName {
   TypeName variable1;
   TypeName variable2;
   ClassName(TypeName variable1, TypeName variable2) {
   this.variable1 = variable1;
   this.variable2 = variable2;
   }
}
```

Source Code:

```
package Java;
import java.util.Scanner;

class Box {
   int width,height,depth;
   Box (int w, int h, int d) {
      width=w;
      height=h;
      depth=d;
   }
   int calVolume() {
      return width*height*depth;
   }
}

public class Program1 {
   public static void main(String[] args) {
      System.out.println("Parameterized Constructor");
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the dimensions: ");
      int w=s.nextInt();
      int h=s.nextInt();
      int d=s.nextInt();
      Box b = new Box(w,h,d);
      System.out.println("The volume of the box is: "+ b.calVolume());
   }
}
```

Output:


Parameterized Constructor
Enter the dimensions: 23 12 9
The volume of the box is: 2484

PROGRAM-2

Aim: Write a program to implement method overriding in Java.

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.

Overview: If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding. Method overriding is one of the ways by which Java achieves Run Time Polymorphism.

Syntax of Method Overriding in Java:

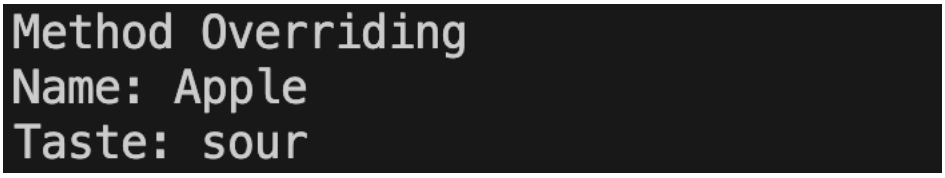
```
class ClassName1 {
   TypeName variable1;
   TypeName variable2;
   returnType method (parameters) {}
}
Class ClassName2 extends ClassName1{
      returnType method (parameters) {}
}
```

Source Code:

```
package Java;

class Fruit {
   String name,taste;
   int size;
   void eat() {
      System.out.println("Inside Fruit class");
   }
}
class Apple extends Fruit {
   Apple (String n, String t, int s) {
      name=n;
      taste=t;
      size=s;
   }
   void eat() {
      System.out.println("Name: "+ name +"\nTaste: "+ taste);
   }
}
class Orange extends Fruit {
   Orange (String n, String t, int s) {
      name=n;
      taste=t;
      size=s;
   }
   void eat() {
      System.out.println("Name: "+ name +"\nTaste: "+ taste);
   }
```

```
}

public class Program2 {
    public static void main(String[] args) {
        Apple a = new Apple("Apple","sour",10);
        System.out.println("Method Overriding");
        a.eat();
    }
}
```

Output:



PROGRAM-3

Aim: Write a program to implement polymorphism in Java.

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();

Overview: Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms. There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.

Syntax of polymorphism in Java is:

```
class ClassName1 {}
class ClassName2 extends ClassName1 {}
ClassName1 cn1 = new ClassName2();
```

Source Code:

```
package Java;

class Shape {
   void draw() {
      System.out.println("Drawing Shape");
   }
   void erase() {
      System.out.println("Erasing Shape");
   }
}
class Circle extends Shape {
   void draw() {
      System.out.println("Drawing Circle");
   }
   void erase() {
      System.out.println("Erasing Circle");
   }
}
class Triangle extends Shape {
   void draw() {
      System.out.println("Drawing Triangle");
   }
   void erase() {
      System.out.println("Erasing Triangle");
   }
}
class Square extends Shape {
   void draw() {
```

```
        System.out.println("Drawing Square");
    }
    void erase() {
        System.out.println("Erasing Square");
    }
}

public class Program3 {
    public static void main(String[] args) {
        Shape c = new Circle();
        Shape t = new Triangle();
        Shape s = new Square();
        c.draw();   c.erase();
        t.draw();   t.erase();
        s.draw();   s.erase();
    }
}
```

Output:

```
Polymorphism in Java
Drawing Circle
Erasing Circle
Drawing Triangle
Erasing Triangle
Drawing Square
Erasing Square
```

PROGRAM-4

Aim: Write a program to implement Exception handling in Java.

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).

Overview: An exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed.Java exceptions cover almost all the general types of exceptions that may occur in the programming. However, we sometimes need to create custom exceptions.

Syntax of Custom Exception in Java is:

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

Source Code:

```
package Java;
import java.util.Scanner;

class ExceptionClass extends Exception {
   public String toString() {
      return "Error! Marks are not in the range(1,100)";
   }
}
public class Program4 {
   Scanner s = new Scanner(System.in);
   String name;
   float sum=0;
   int marks[]=new int[3];
   void input() throws ExceptionClass {
      System.out.print("Enter name of the Student: ");
      name=s.nextLine();
      System.out.print("Enter the marks of "+ name +" in Physics, Chemistry and Maths: ");
      for (int i=0;i<3; i++) {
         marks[i]=Integer.parseInt(s.next());
         if (marks[i]<0 || marks[i]>100) {
            throw new ExceptionClass();
         }
         sum+=marks[i];
      }
      System.out.println("Average of marks of "+ name +" is "+ sum/3);
   }
   public static void main(String[] args) {
      System.out.println("NumberFormatException, Custom Exception in Java");
      Program4 a=new Program4();
```

```
        Program4 b=new Program4();
        try {
            a.input();
            b.input();
        }
        catch (NumberFormatException n) {
            System.out.println("NumberFormatException caught "+ n.getMessage());
        }
        catch (ExceptionClass e) {
            System.out.println(e);
        }
    }
}
```

Output:

```
NumberFormatException, Custom Exception in Java
Enter name of the Student: Abhishek
Enter the marks of Abhishek in Physics, Chemistry and Maths: 88 91 90
Average of marks of Abhishek is 89.666664
Enter name of the Student: Varun
Enter the marks of Varun in Physics, Chemistry and Maths: 90 94 78
Average of marks of Varun is 87.333336
```

```
NumberFormatException, Custom Exception in Java
Enter name of the Student: Abhishek
Enter the marks of Abhishek in Physics, Chemistry and Maths: 88 91 sw
NumberFormatException caught For input string: "sw"
```

```
NumberFormatException, Custom Exception in Java
Enter name of the Student: Abhishek
Enter the marks of Abhishek  in Physics, Chemistry and Maths: 88 91 -2
Error! Marks are not in the range(1,100)
```

PROGRAM-5

Aim: Write a program to implement Exception handling in Java.

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

Overview: An exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed. The ArrayIndexOutOfBoundsException occurs whenever we are trying to access any item of an array at an index which is not present in the array. In other words, the index may be negative or exceed the size of an array. The NumberFormatException is thrown when we try to convert a string into a numeric value such as float or integer, but the format of the input string is not appropriate or illegal.

Syntax to handle any Exception in Java is:

```
try {
    // Block of code to try
}
catch (Exception e) {
    // Block of code to handle errors
}
```

Source Code:

```
import java.util.Scanner;

public class Program5 {
   public static void main(String[] args) {
      System.out.println("ArrayIndexOutOfBoundsException, NumberFormatException in Java");
      try {
         Scanner s = new Scanner(System.in);
         System.out.print("Enter the number of elements in the array: ");
         int n=Integer.parseInt(s.next());
         int arr[]=new int[n];
         System.out.print("Enter the elements of the array: ");
         for (int i=0;i<n;i++) {
            arr[i]=Integer.parseInt(s.next());
         }
         System.out.print("Enter the index at which element is to be found: ");
         int a=Integer.parseInt(s.next());
         System.out.println("Element at index "+ a +" is "+ arr[a]);
      }
      catch (ArrayIndexOutOfBoundsException a) {
         System.out.println("ArrayIndexOutOfBoundsException caught "+ a.getMessage());
      }
      catch (NumberFormatException n) {
         System.out.println("NumberFormatException caught "+ n.getMessage());
      }
   }
}
```

Output:

```
ArrayIndexOutOfBoundsException, NumberFormatException in Java
Enter the number of elements in the array: 4
Enter the elements of the array: 2 1 4 10
Enter the index at which element is to be found: 3
Element at index 3 is 10
```

```
ArrayIndexOutOfBoundsException, NumberFormatException in Java
Enter the number of elements in the array: 4
Enter the elements of the array: 2 1 4 10
Enter the index at which element is to be found: 5
ArrayIndexOutOfBoundsException caught Index 5 out of bounds for length 4
```

```
ArrayIndexOutOfBoundsException, NumberFormatException in Java
Enter the number of elements in the array: 4
Enter the elements of the array: 2 1 sw 10
NumberFormatException caught For input string: "sw"
```

Experiment - 6

Aim :- Implement Datagram UDP socket programming in java

Overview:

The provided Java code demonstrates Datagram UDP socket programming. The UDP server listens on port 9876, receiving messages from clients and optionally sending a response. The UDP client sends a "Hello" message to the server, prints the sent message, and optionally receives a response from the server. This implementation enables simple communication between a UDP server and client using datagrams in Java

Code:

UDP Server

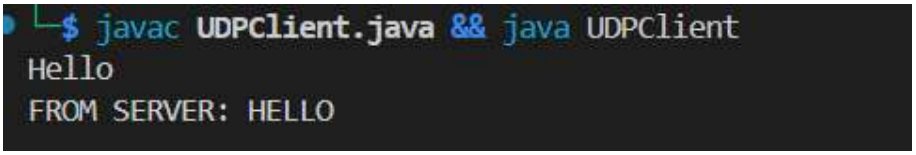
```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception {
    DatagramSocket serverSocket = new DatagramSocket(9876);
    byte[] receiveData = new byte[1024];
    byte[] sendData;
    while (true) {
      DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
      serverSocket.receive(receivePacket);
      String sentence = new String(receivePacket.getData(), 0,
receivePacket.getLength());
      InetAddress IPAddress = receivePacket.getAddress();
      int port = receivePacket.getPort();
      String capitalizedSentence = sentence.toUpperCase();
      sendData = capitalizedSentence.getBytes();
      DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
          IPAddress, port);
      serverSocket.send(sendPacket);
    }
  }
}
```

UDP Server

```
import java.io.*;
import java.net.*;

class UDPClient {
  public static void main(String args[]) throws Exception {
    BufferedReader inFromUser = new BufferedReader(
```

```
        new InputStreamReader(System.in));
    DatagramSocket clientSocket = new DatagramSocket();
    InetAddress IPAddress = InetAddress.getByName("localhost");
    byte[] sendData;
    byte[] receiveData = new byte[1024];
    String sentence = inFromUser.readLine();
    sendData = sentence.getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
        IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence = new String(receivePacket.getData(),
        0, receivePacket.getLength());
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
  }
}
```

Output:



Experiment - 7

Aim :- Implement Socket programming for TCP in Java Server and Client Sockets.

Overview:

This Java code implements TCP socket programming with a server and client. The TCP server listens on a specified port, accepts client connections, and can handle multiple clients concurrently. The client connects to the server, sends a message ("Hello, TCP Server!"), and receives a response. This implementation facilitates communication between a TCP server and client through sockets in Java.

Code:

TCP Server

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception {
    String clientSentence;
    String capitalizedSentence;
    ServerSocket welcomeSocket = new ServerSocket(6789);
    while (true) {
      Socket connectionSocket = welcomeSocket.accept();
      BufferedReader inFromClient = new BufferedReader(
        new InputStreamReader(connectionSocket.getInputStream())
      );
      DataOutputStream outToClient = new DataOutputStream(
        connectionSocket.getOutputStream()
      );
      clientSentence = inFromClient.readLine();
      capitalizedSentence = clientSentence.toUpperCase() + '\n';
      outToClient.writeBytes(capitalizedSentence);
    }
  }
}
```

TCP Client

```
import java.io.*;
import java.net.*;

class TCPClient {

  public static void main(String argv[]) throws Exception {
```

```
    String sentence;
    String modifiedSentence;
    BufferedReader inFromUser = new BufferedReader(
      new InputStreamReader(System.in)
    );
    Socket clientSocket = new Socket("localhost", 6789);
    DataOutputStream outToServer = new DataOutputStream(
      clientSocket.getOutputStream()
    );
    BufferedReader inFromServer = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream())
    );
    sentence = inFromUser.readLine();
    outToServer.writeBytes(sentence + '\n');
    modifiedSentence = inFromServer.readLine();
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
  }
}
```

Output:

Experiment - 8

Aim :- Socket Programming

Overview:

This Java code represents a basic implementation of TCP socket programming with a server and client. The server waits for a connection, accepts a client, and reads messages until the client sends "Over." The client connects to the server, sends messages from the terminal to the server until "Over" is entered, and then closes the connection. This practical demonstrates simple bidirectional communication between a TCP server and client, establishing a connection and exchanging messages over sockets in a networked environment.

Code:

Server

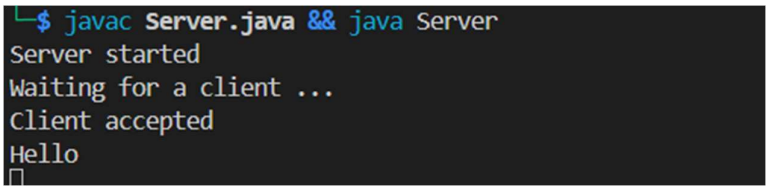
```
// A Java program for a Server
import java.io.*;
import java.net.*;

public class Server {

  //initialize socket and input stream
  private Socket socket = null;
  private ServerSocket server = null;
  private DataInputStream in = null;

  // constructor with port
  public Server(int port) {
    // starts server and waits for a connection
    try {
      server = new ServerSocket(port);
      System.out.println("Server started");
      System.out.println("Waiting for a client ...");
      socket = server.accept();
      System.out.println("Client accepted");
      // takes input from the client socket
      in =
        new DataInputStream(new BufferedInputStream(socket.getInputStream()));
      String line = "";
      // reads message from client until "Over" is sent
      while (!line.equals("Over")) {
        try {
          line = in.readUTF();
          System.out.println(line);
        } catch (IOException i) {
          System.out.println(i);
```

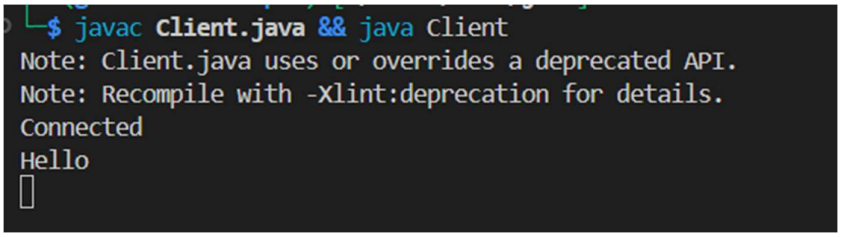
```
      }
     }
     System.out.println("Closing connection");
     // close connection
     socket.close();
     in.close();
    } catch (IOException i) {
     System.out.println(i);
    }
  }

  public static void main(String args[]) {
   Server server = new Server(5000);
  }
}
```

Client

```
// A Java program for a Client
import java.io.*;
import java.net.*;

public class Client {

  // initialize socket and input output streams
  private Socket socket = null;
  private DataInputStream input = null;
  private DataOutputStream out = null;

  // constructor to put ip address and port
  public Client(String address, int port) {
   // establish a connection
   try {
    socket = new Socket(address, port);
    System.out.println("Connected");
    // takes input from terminal
    input = new DataInputStream(System.in);
    // sends output to the socket
    out = new DataOutputStream(socket.getOutputStream());
   } catch (UnknownHostException u) {
    System.out.println(u);
    return;
   } catch (IOException i) {
    System.out.println(i);
    return;
   }
   // string to read message from input
   String line = "";
```

```
    // keep reading until "Over" is input
    while (!line.equals("Over")) {
      try {
        line = input.readLine();
        out.writeUTF(line);
      } catch (IOException i) {
        System.out.println(i);
      }
    }
    // close the connection
    try {
      input.close();
      out.close();
      socket.close();
    } catch (IOException i) {
      System.out.println(i);
    }
  }

  public static void main(String args[]) {
    Client client = new Client("127.0.0.1", 5000);
  }
}
```





PROGRAM-9

Aim: Implement Producer-Consumer Problem using multithreading.

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

- The producer's job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

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

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

Source Code:

```
import java.util.LinkedList;

public class Program9 {
   public static void main(String[] args) throws InterruptedException {
      System.out.println("Producer-Consumer Problem using Multithreading");
      final PC pc = new PC();
      Thread t1 = new Thread(() -> {
         try {
            pc.produce();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      });
      Thread t2 = new Thread(() -> {
         try {
            pc.consume();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      });
      t1.start();
      t2.start();
      t1.join();
      t2.join();
   }
   public static class PC {
      LinkedList<Integer> list = new LinkedList<>();
      int capacity = 2;
      public void produce() throws InterruptedException {
         int value = 0;
         while (true) {
            synchronized (this) {
               while (list.size() == capacity)
                  wait();
```

```
                System.out.println("Producer produced-" + value);
                list.add(value++);
                notify();
                Thread.sleep(1000);
            }
        }
    }
    public void consume() throws InterruptedException {
        while (true) {
            synchronized (this) {
                while (list.size() == 0)
                    wait();
                int val = list.removeFirst();
                System.out.println("Consumer consumed-" + val);
                notify();
                Thread.sleep(1000);
            }
        }
    }
}
```

Output:

```
Producer-Consumer Problem using Multithreading
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Producer produced-5
Consumer consumed-4
Consumer consumed-5
```

PROGRAM-10

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

Overview: Priorities in threads is a concept where each thread is having a priority which in layman's language one can say every object is having priority here which is represented by numbers ranging from 1 to 10.
The default priority is set to 5 as excepted. (public static int NORM_PRIORITY)
Minimum priority is set to 1. (public static int MIN_PRIORITY)
Maximum priority is set to 10. (public static int MAX_PRIORITY)

Source Code:

```
public class Program10 extends Thread {
   public void run(){
      System.out.println("Inside run method");
   }
   public static void main(String[] args) {
      System.out.println("Priorities in Multithreading");
      Program10 t1 = new Program10();
      Program10 t2 = new Program10();
      Program10 t3 = new Program10();
      System.out.println("t1 thread priority: " + t1.getPriority());
      System.out.println("t2 thread priority: " + t2.getPriority());
      System.out.println("t3 thread priority: " + t3.getPriority());
      t1.setPriority(2);
      t2.setPriority(5);
      t3.setPriority(8);
      System.out.println("t1 thread priority: " + t1.getPriority());
      System.out.println("t2 thread priority: " + t2.getPriority());
      System.out.println("t3 thread priority: " + t3.getPriority());
      System.out.println("Currently Executing Thread: "+ Thread.currentThread().getName());
      System.out.println( "Main thread priority: "+ Thread.currentThread().getPriority());
      Thread.currentThread().setPriority(10);
      System.out.println("Main thread priority: "+ Thread.currentThread().getPriority());
   }
}
```

Output:

```
Priorities in Multithreading
t1 thread priority: 5
t2 thread priority: 5
t3 thread priority: 5
t1 thread priority: 2
t2 thread priority: 5
t3 thread priority: 8
Currently Executing Thread: main
Main thread priority: 5
Main thread priority: 10
```

PROGRAM-11

Aim: Illustrate Deadlock in multithreading.

Overview: Deadlock in Java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.
Synchronization keyword is used to make the class or method thread-safe which means only one thread can have lock of synchronized method and use it, other threads have to wait till the lock releases and anyone of them acquire that lock.

Source Code:

Demonstrating Deadlock Situation:

```
public class Program11a {
   public static Object Lock1 = new Object();
   public static Object Lock2 = new Object();
   public static void main(String args[]) {
      System.out.println("Deadlock Situation");
      ThreadDemo1 T1 = new ThreadDemo1();
      ThreadDemo2 T2 = new ThreadDemo2();
      T1.start();
      T2.start();
   }
   private static class ThreadDemo1 extends Thread {
      public void run() {
         synchronized (Lock1) {
            System.out.println("Thread 1: Holding lock 1...");
            try {
               Thread.sleep(10);
            } catch (InterruptedException e) {}
            System.out.println("Thread 1: Waiting for lock 2...");
            synchronized (Lock2) {
               System.out.println("Thread 1: Holding lock 1 & 2...");
            }
         }
      }
   }
   private static class ThreadDemo2 extends Thread {
      public void run() {
         synchronized (Lock2) {
            System.out.println("Thread 2: Holding lock 2...");
            try {
               Thread.sleep(10);
            } catch (InterruptedException e) {}
            System.out.println("Thread 2: Waiting for lock 1...");
            synchronized (Lock1) {
               System.out.println("Thread 2: Holding lock 1 & 2...");
            }
         }
      }
   }
}
```

Output:

```
Deadlock Situation
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 1: Waiting for lock 2...
Thread 2: Waiting for lock 1...
```

Source Code:

Deadlock Solution:

```
public class Program11b {
    public static Object lock1 = new Object();
    public static Object lock2 = new Object();
    public static void main(String[] args) {
        System.out.println("Deadlock Solution");
        new Thread1().start();
        new Thread2().start();
    }
    private static class Thread1 extends Thread {
        @Override
        public void run() {
            synchronized (lock1) {
                System.out.println("Thread-1 acquired lock1");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    System.out.println("Thread-1 interrupted.");
                }
                System.out.println("Thread-1 waiting for lock2");
                synchronized (lock2) {
                    System.out.println("Thread-1 acquired lock2");
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        System.out.println("Thread-1 interrupted.");
                    }
                }
                System.out.println("Thread-1 releases lock2");
            }
            System.out.println("Thread-1 releases lock1");
        }
    }
    private static class Thread2 extends Thread {
        @Override
        public void run() {
            synchronized (lock1) {
                System.out.println("Thread-2 acquired lock1");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    System.out.println("Thread-2 interrupted.");
                }
                System.out.println("Thread-2 waiting for lock2");
```

```
            synchronized (lock2) {
                System.out.println("Thread-2 acquired lock2");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    System.out.println("Thread-2 interrupted.");
                }
            }
            System.out.println("Thread-2 releases lock2");
        }
        System.out.println("Thread-2 releases lock1");
    }
  }
}
```

Output:

```
Deadlock Solution
Thread-1 acquired lock1
Thread-1 waiting for lock2
Thread-1 acquired lock2
Thread-1 releases lock2
Thread-1 releases lock1
Thread-2 acquired lock1
Thread-2 waiting for lock2
Thread-2 acquired lock2
Thread-2 releases lock2
Thread-2 releases lock1
```

Experiment – 12

Aim :- Implement Producer-Consumer Problem using multithreading.

Overview:

The Producer-Consumer problem epitomizes the challenge of coordinating concurrent threads sharing a bounded buffer. Producers generate items and deposit them, while consumers retrieve and process them. Synchronization mechanisms like semaphores or mutexes ensure orderly access to the buffer, preventing issues like overflow or underflow. Through this synchronization, threads collaborate efficiently, balancing resource utilization with system stability.

Code:

```
import java.util.LinkedList;

class ProducerConsumer {
  private final LinkedList<Integer> buffer = new LinkedList<>();
  private final int capacity = 5;
  private int producedCount = 0;
  private int consumedCount = 0;
  private final int totalIterations = 10; // Total number of iterations

  public void produce() throws InterruptedException {
    while (producedCount < totalIterations) {
      synchronized (this) {
        while (buffer.size() == capacity)
          wait();
        int value = producedCount++;
        System.out.println("Producer produced: " + value);
        buffer.add(value);
        notify();
      }
    }
  }

  public void consume() throws InterruptedException {
    while (consumedCount < totalIterations) {
      synchronized (this) {
        while (buffer.size() == 0)
          wait();
        int consumedValue = buffer.removeFirst();
        System.out.println("Consumer consumed: " + consumedValue);
        consumedCount++;
        notify();
      }
    }
  }
}

public class Exp12 {
```

```
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer();
        Thread producerThread = new Thread(() -> {
            try {
                pc.produce();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        Thread consumerThread = new Thread(() -> {
            try {
                pc.consume();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        producerThread.start();
        consumerThread.start();
    }
}
```

```
└─$ javac Exp12.java && java Exp12
Producer produced: 0
Producer produced: 1
Producer produced: 2
Producer produced: 3
Producer produced: 4
Consumer consumed: 0
Consumer consumed: 1
Consumer consumed: 2
Consumer consumed: 3
Consumer consumed: 4
Producer produced: 5
Producer produced: 6
Producer produced: 7
Producer produced: 8
Producer produced: 9
Consumer consumed: 5
Consumer consumed: 6
Consumer consumed: 7
Consumer consumed: 8
Consumer consumed: 9
```

Experiment – 13

Aim :- Illustrate Priorities in Multithreading via help of getPriority() and setPriority() method.

Overview:

Thread priorities in multithreading range from 1 to 10, with 1 being the lowest priority and 10 being the highest. Higher priority threads are scheduled for execution before lower priority threads. getPriority() retrieves a thread's priority, and setPriority() sets it. Use priorities judiciously, as they may not always have a significant impact and can lead to platform-specific behavior.

Code:

```
class Task extends Thread {

   public Task(String name) {
      super(name);
   }
   public void run() {
      for (int i = 0; i < 5; i++) {
         System.out.println(getName() + " is executing");
         try {
            Thread.sleep(100); // Simulate some work
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }  } }

public class Exp13 {
   public static void main(String[] args) {
      Task highPriorityThread = new Task("High Priority Thread");
      Task lowPriorityThread = new Task("Low Priority Thread");
      highPriorityThread.setPriority(Thread.MAX_PRIORITY);
      lowPriorityThread.setPriority(Thread.MIN_PRIORITY);
      highPriorityThread.start();
      lowPriorityThread.start();
   }
}
```

```
└─$ javac Exp13.java && java Exp13
High Priority Thread is executing
Low Priority Thread is executing
High Priority Thread is executing
Low Priority Thread is executing
Low Priority Thread is executing
High Priority Thread is executing
High Priority Thread is executing
Low Priority Thread is executing
Low Priority Thread is executing
High Priority Thread is executing
```

Experiment – 14

Aim :- Illustrate Deadlock in multithreading.

Overview: Deadlock in multithreading occurs when two or more threads are stuck indefinitely, each waiting for the other to release a resource. This results from improper synchronization, where threads hold resources needed by others. Prevention involves careful resource management and avoiding circular dependencies. Detection requires monitoring thread states, and resolution strategies include timeouts and avoiding resource contention. Proper design and synchronization are crucial to prevent deadlock in multithreaded applications.

Code:

```
public class Exp14 {
  private static final Object lock1 = new Object();
  private static final Object lock2 = new Object();
  public static void main(String[] args) {
    Thread thread1 = new Thread(() -> {
      synchronized (lock1) {
        System.out.println("Thread 1: Holding lock 1...");
        try {
          Thread.sleep(100); // Simulating some work
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        System.out.println("Thread 1: Waiting for lock 2...");
        synchronized (lock2) {
          System.out.println("Thread 1: Holding lock 1 and lock 2...");
        }
      }
    });
    Thread thread2 = new Thread(() -> {
      synchronized (lock2) {
        System.out.println("Thread 2: Holding lock 2...");
        try {
          Thread.sleep(100); // Simulating some work
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        System.out.println("Thread 2: Waiting for lock 1...");
        synchronized (lock1) {
          System.out.println("Thread 2: Holding lock 1 and lock 2...");
        }
      }
    });
    thread1.start();
    thread2.start();
  }
}
```

```
└─$ javac Exp14.java && java Exp14
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 2: Waiting for lock 1...
Thread 1: Waiting for lock 2...
^C
```

Experiment – 15

Aim :- Implement a program Java Bean to represent person details.

Overview: A Java Bean representing person details is a simple Java class that encapsulates information about a person. It typically includes private fields for attributes such as name, age, and address, along with getter and setter methods for accessing and modifying these attributes. This ensures data encapsulation and abstraction, allowing controlled access to the person's details while maintaining flexibility and ease of use within Java applications.

Code:

```
public class PersonExp15 {
  private String name;
  private int age;
  private String address;
  public PersonExp15(String name, int age, String address) {
    this.name = name;
    this.age = age;
    this.address = address;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  public String getAddress() {
    return address;
  }
  public void setAddress(String address) {
    this.address = address;
  }
  @Override
  public String toString() {
    return (
      "Person [name=" + name + ", age=" + age + ", address=" + address + "]"
    );
  }
  public static void main(String[] args) {
    PersonExp15 person = new PersonExp15("John Doe", 30, "123 Main St");
    System.out.println(person);
  }
}
```

```
└─$ javac PersonExp15.java && java PersonExp15
Person [name=John Doe, age=30, address=123 Main St]
```

Experiment – 16

Aim :- Write a program in java to demonstrate encapsulation in java beans.

Overview: Encapsulation in Java beans is a fundamental concept in object-oriented programming (OOP). It refers to the practice of bundling the data (attributes) and methods (functions) that operate on the data into a single unit or entity, known as a bean. Encapsulation helps in hiding the internal state of an object and only exposing necessary operations through methods.

Code:

```
public class Exp16 {
   private String name;
   private int age;
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      if (age >= 0) {
         this.age = age;
      } else {
         System.out.println("Age cannot be negative.");
      }
   }
   public static void main(String[] args) {
      Exp16 person = new Exp16();
     person.setName("Random Person xyz");
      person.setAge(45);
      System.out.println("Name: " + person.getName());
      System.out.println("Age: " + person.getAge());
      person.setAge(-5);
   }
}
```

```
└─$ javac Exp16.java && java Exp16
Name: Random Person xyz
Age: 45
Age cannot be negative.
```

Experiment – 17

Aim :- Create a database in MySQL using JSP and perform insertion and retrieval operations.

Overview: Creating a database in MySQL using JSP involves several steps. First, you need to set up the MySQL server and create a database using the MySQL command-line interface or a graphical tool like phpMyAdmin. Next, you can connect your JSP application to the MySQL database using JDBC (Java Database Connectivity) by including the MySQL JDBC driver in your project. In your JSP code, you can write SQL queries to insert data into the database using INSERT statements and retrieve data using SELECT statements. Finally, you can display the retrieved data on your JSP pages. This integration allows for dynamic content generation and interaction with a persistent data store.

Code:

SQL Commands:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL
);
```

JSP Files: index.jsp: This page contains a form to insert user data into the database.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Insert User</title>
</head>
<body>
  <h2>Insert User</h2>
  <form action="insert.jsp" method="post">
    Username: <input type="text" name="username"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

index.jsp: This page inserts user data into the database.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

```
<%@ page import="java.sql.*" %>
<%@ page import="java.io.PrintWriter" %>
<%@ page import="java.io.IOException" %>
<%
String username = request.getParameter("username");
String email = request.getParameter("email");
try {
   Class.forName("com.mysql.jdbc.Driver");
   Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase",
"root", "");
   Statement stmt = con.createStatement();
   String sql = "INSERT INTO users (username, email) VALUES (?, ?)";
   PreparedStatement pstmt = con.prepareStatement(sql);
   pstmt.setString(1, username);
   pstmt.setString(2, email);
   pstmt.executeUpdate();
   out.println("User inserted successfully.");
   pstmt.close();
   con.close();
} catch (Exception e) {
   out.println("Error: " + e.getMessage()); }
%>
```

retrieve.jsp: This page retrieves user data from the database and displays it.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<%@ page import="java.sql.*" %>
<%@ page import="java.io.PrintWriter" %>
<%@ page import="java.io.IOException" %>
<%
try {
   Class.forName("com.mysql.jdbc.Driver");
   Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase",
"root", "");
   Statement stmt = con.createStatement();
   String sql = "SELECT * FROM users";
   ResultSet rs = stmt.executeQuery(sql);
   out.println("<h2>User List</h2>");
   out.println("<table border='1'><tr><th>ID</th><th>Username</th><th>Email</th></tr>");
   while (rs.next()) {
      out.println("<tr><td>" + rs.getInt("id") + "</td><td>" + rs.getString("username") +
"</td><td>" + rs.getString("email") + "</td></tr>");
   }
   out.println("</table>");
   rs.close();
   stmt.close();
```

Name

Name

Email

Email

Phone

Phone

Password

Password

Submit

Account Already Exists. Login Here

Experiment – 18

Aim :- Create a Java JSP login and Sign Up form with Session using MySQL.

Overview: A Java JSP login and sign-up form with session management using MySQL provides secure user authentication and registration. JSP pages handle user input, connecting to MySQL for data storage. Session management ensures user persistence across pages. Security measures like hashing passwords prevent data breaches. This system enhances web application functionality and security.

Code:

```
DOCTYPE html>
<html>
<head>
  <title>Form Validation with Regular Expressions</title>
</head>
<body>
<h2>User Registration</h2>
<form method="post" action="processData.jsp" onsubmit="return validateForm();">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required>
  <br>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>
  <br>
  <label for="password">Password:</label>
  <input type="password" id="password" name="password" required>
  <br>
  <input type="submit" value="Submit">
</form>
<script>
  function validateForm() {
    var username = document.getElementById("username").value;
    var email = document.getElementById("email").value;
    var password = document.getElementById("password").value;
    var usernamePattern = /^[a-zA-Z0-9_]{5,}$/; // Alphanumeric and underscore, minimum
length 5
    var emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/; // Email pattern
    var passwordPattern = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$/; // Minimum 8
characters, at least one uppercase letter, one lowercase letter, and one number
    if (!usernamePattern.test(username)) {
      alert("Username must be alphanumeric and at least 5 characters long.");
      return false;
    }
    if (!emailPattern.test(email)) {
      alert("Please enter a valid email address.");
      return false;
    }
    if (!passwordPattern.test(password)) {
      alert("Password must be at least 8 characters long, contain at least one uppercase letter,
one lowercase letter, and one number.");
      return false;
    }
```

```
    return true;
  }
</script>
</body>
</html>
```

User Registration

Username:

Email:

Password:

Submit

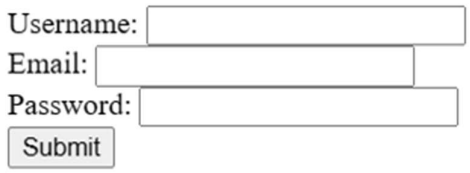
Experiment – 19

Aim :- Implement Regular Expressions validation before submitting data in JSP.

Overview: Regular Expressions validation in JSP involves verifying input data against predefined patterns before submission. Utilizing regex patterns, it ensures data conformity to desired formats, such as email addresses or alphanumeric strings. Implementation typically occurs on the server-side, enhancing security by preventing malicious or erroneous data from reaching backend systems. This validation process helps maintain data integrity and improves user experience by prompting corrective actions for invalid inputs.

Code:

```
import java.util.regex.*;

public class FormValidationServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String username = request.getParameter("username");
    String email = request.getParameter("email");
    String usernameRegex = "[a-zA-Z0-9_]+";
    String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-
zA-Z]{2,7}$";
    Pattern usernamePattern = Pattern.compile(usernameRegex);
    Pattern emailPattern = Pattern.compile(emailRegex);
    boolean isValidUsername = usernamePattern.matcher(username).matches();
    boolean isValidEmail = emailPattern.matcher(email).matches();
    if (isValidUsername && isValidEmail) {
      response.sendRedirect("success.jsp");
    } else {
      request.setAttribute("error", "Invalid username or email format");
      request.getRequestDispatcher("/form.jsp").forward(request, response);
    }
  }
}
```



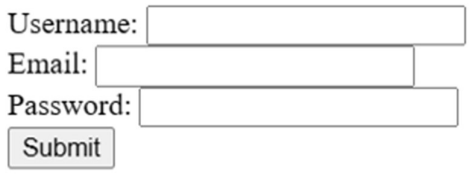
Experiment – 20

Aim :- Implement Customizable adapter class in a registration form in JSP

Overview: A customizable adapter class for a registration form in JSP acts as an intermediary between the JSP page and the database. It encapsulates the logic for user registration, including validation and database interaction. The adapter class typically provides methods to validate user input, connect to a database using JDBC, and execute SQL queries to store user information. By using an adapter class, the registration process becomes modular and easier to manage

Code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
public class RegistrationAdapter {
  private static final String DB_URL = "jdbc:mysql://localhost:3306/mydatabase";
  private static final String DB_USER = "root";
  private static final String DB_PASSWORD = "password";
  public boolean registerUser(String username, String password, String email) {
    boolean isSuccess = false;
    try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
      String query = "INSERT INTO users (username, password, email) VALUES (?, ?, ?)";
      try (PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString(1, username);
        stmt.setString(2, password);
        stmt.setString(3, email);
        int rowsInserted = stmt.executeUpdate();
        if (rowsInserted > 0) {
          isSuccess = true;
        }
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return isSuccess;
  }
}
```



Experiment – 21

Aim :- 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.

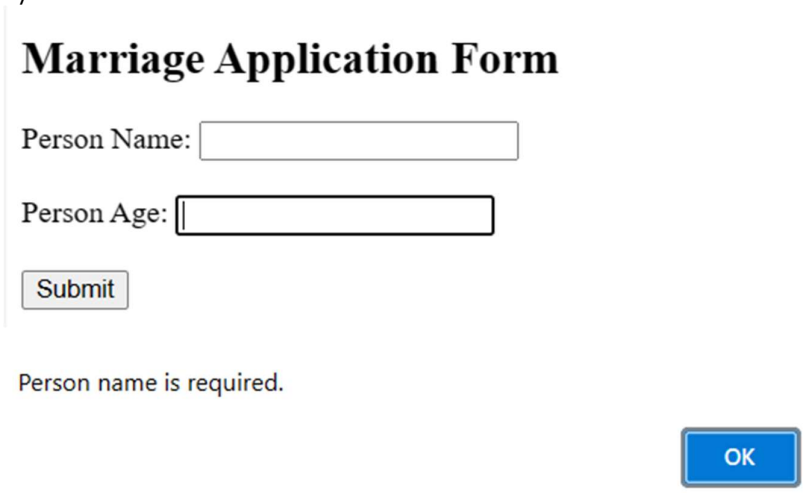
3. Personage is required.

4. Person age must be a numeric value. 5. Personage must be there between 1 to 125

Code:

```
DOCTYPE html>
<html>
<head>
  <title>Marriage Application Form</title>
  <script>
    function validateForm() {
      var name = document.forms["marriageForm"]["personName"].value;
      var age = document.forms["marriageForm"]["personAge"].value;
      if (name == "") {
        alert("Person name is required.");
        return false;
      }
      if (name.length < 5) {
        alert("Person's name must have a minimum of 5 characters.");
        return false;
      }
      if (age == "") {
        alert("Person age is required.");
        return false;
      }
      if (isNaN(age)) {
        alert("Person age must be a numeric value.");
        return false;
      }
      if (age < 1 || age > 125) {
        alert("Person age must be between 1 to 125.");
        return false;
      }
      return true;
    }
  </script>
</head>
<body>
  <h2>Marriage Application Form</h2>
  <form name="marriageForm" onsubmit="return validateForm()">
    <label for="personName">Person Name:</label>
    <input type="text" id="personName" name="personName"><br><br>
    <label for="personAge">Person Age:</label>
    <input type="text" id="personAge" name="personAge"><br><br>
    <input type="submit" value="Submit">
  </form>
```

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


Marriage Application Form
Person Name:
Person Age:
Submit

Person name is required.
OK

Experiment – 22

Aim :- Design Servlet Login and Logout using Cookies

Overview: Servlets can be used to implement login and logout functionalities by managing user sessions with cookies. When a user logs in, a cookie can be created to store a unique session identifier. This identifier can be used to track the user's session and maintain their logged-in state across requests. For logout, the cookie can be invalidated or expired to end the session. Servlets can retrieve the cookie information from the request to verify the user's session during subsequent requests. By using cookies, the login and logout processes become stateful, allowing for personalized user experiences and secure session management in web applications.

Code:

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>
<h1>Welcome to Login App by Cookie</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>
</body>
</html>
```

File: link.html

```
<a href="login.html">Login</a> |
<a href="LogoutServlet">Logout</a> |
<a href="ProfileServlet">Profile</a>
<hr>
```

File: login.html

```
<form action="LoginServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: LoginServlet.java

```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
```

```
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);
    String name=request.getParameter("name");
    String password=request.getParameter("password");
    if(password.equals("admin123")){
      out.print("You are successfully logged in!");
      out.print("<br>Welcome, "+name);
      Cookie ck=new Cookie("name",name);
      response.addCookie(ck);
    }else{
      out.print("sorry, username or password error!");
      request.getRequestDispatcher("login.html").include(request, response);
    }
    out.close();
  }
}
```

File: LogoutServlet.java

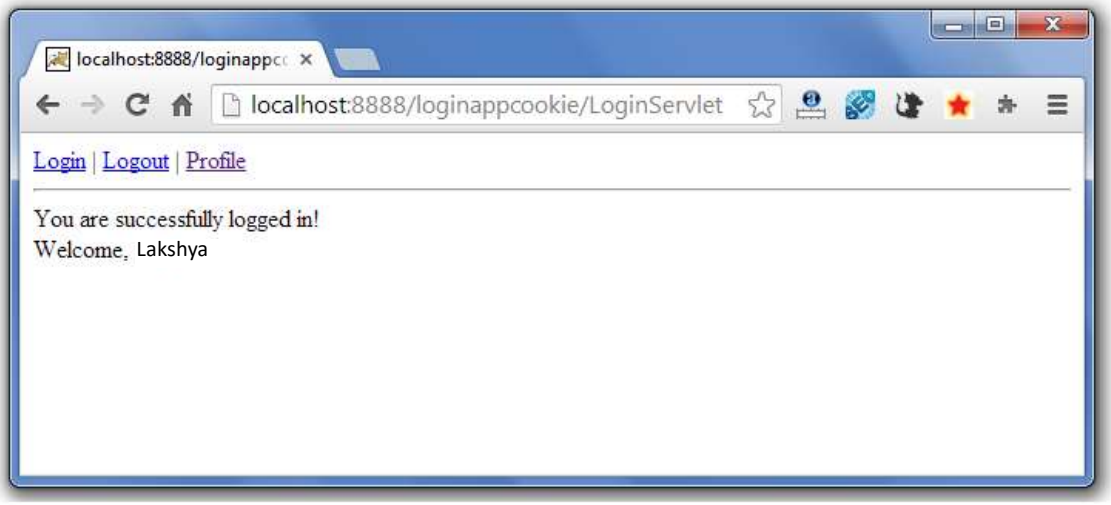
```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LogoutServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);
    Cookie ck=new Cookie("name","");
    ck.setMaxAge(0);
    response.addCookie(ck);
    out.print("you are successfully logged out!");
  }
}
```

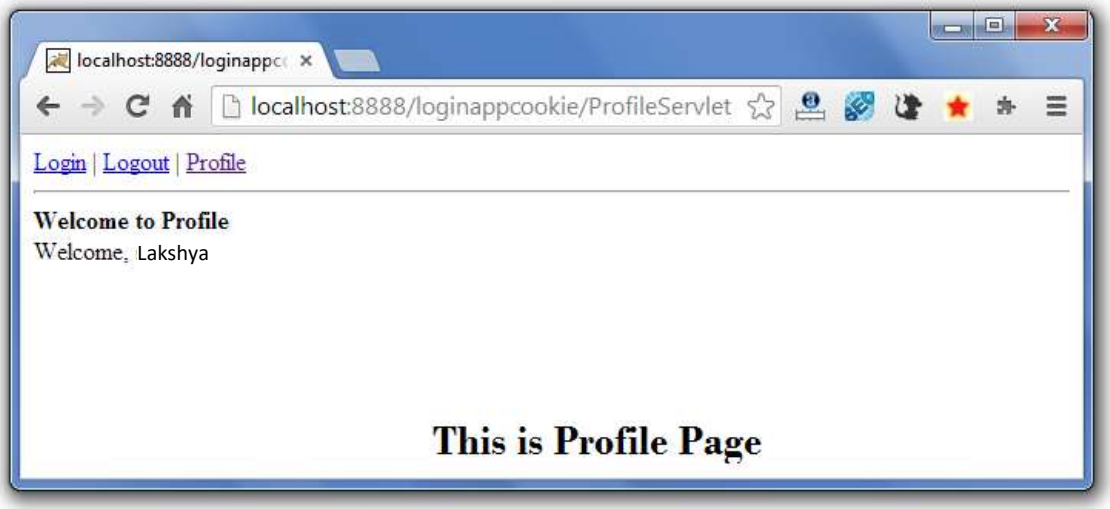
File: ProfileServlet.java

```
package com.javatpoint;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ProfileServlet extends HttpServlet {
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();

      request.getRequestDispatcher("link.html").include(request, response);

      Cookie ck[]=request.getCookies();
      if(ck!=null){
       String name=ck[0].getValue();
      if(!name.equals("")||name!=null){
         out.print("<b>Welcome to Profile</b>");
         out.print("<br>Welcome, "+name);
      }
      }else{
         out.print("Please login first");
         request.getRequestDispatcher("login.html").include(request, response);
      }
      out.close();
   }
}
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <servlet>
    <description></description>
    <display-name>LoginServlet</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.javatpoint.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>ProfileServlet</display-name>
    <servlet-name>ProfileServlet</servlet-name>
    <servlet-class>com.javatpoint.ProfileServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ProfileServlet</servlet-name>
    <url-pattern>/ProfileServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>LogoutServlet</display-name>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>com.javatpoint.LogoutServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
```

```
  <url-pattern>/LogoutServlet</url-pattern>
 </servlet-mapping>
</web-app>
```




localhost:8888/loginappc
localhost:8888/loginappcookie/ProfileServlet
Login | Logout | Profile
Welcome to Profile
Welcome, Lakshya
This is Profile Page


localhost:8888/loginappc
localhost:8888/loginappcookie/LogoutServlet
Login | Logout | Profile
you are successfully logged out!
This is logout or signout page

Experiment – 23

Aim :- Create a servlet that prints all the request headers it receives, along with their associated values.

Overview: A servlet can retrieve and display request headers using the `HttpServletRequest` object's `getHeaderNames()` method, which returns an enumeration of all the header names in the current request. By iterating over this enumeration and retrieving each header's value using the `getHeader()` method, the servlet can print each header and its associated value. This approach allows developers to inspect incoming HTTP headers, providing insights into the client's request and facilitating server-side processing tailored to the specific client or user agent interacting with the servlet.

Code:

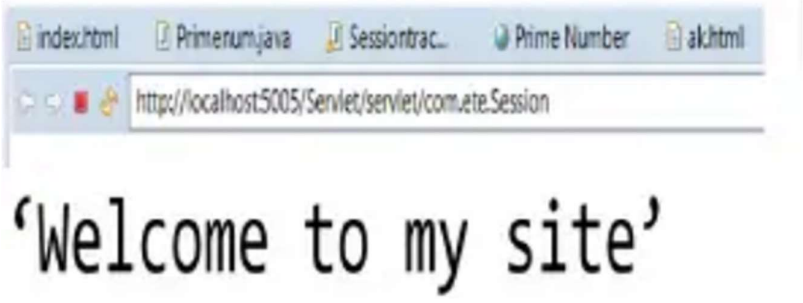
```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collections;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RequestHeadersServlet")
public class RequestHeadersServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><head><title>Request Headers</title></head><body>");
      out.println("<h2>Request Headers:</h2>");
      out.println("<ul>");
      Enumeration<String> headerNames = request.getHeaderNames();
      while (headerNames.hasMoreElements()) {
         String headerName = headerNames.nextElement();
         out.println("<li><strong>" + headerName + ":</strong> " +
request.getHeader(headerName) + "</li>");
      }
      out.println("</ul>");
      out.println("</body></html>");
   }
}
```

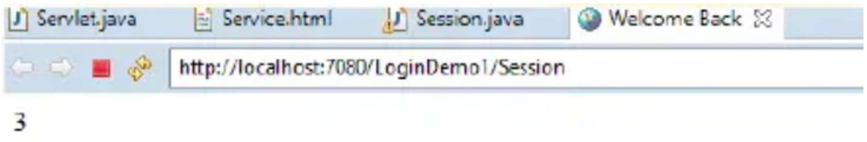
Experiment – 24

Aim :- 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".

Code:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/WelcomeServlet")
public class WelcomeServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      boolean isFirstTimeVisitor = true;
      Cookie[] cookies = request.getCookies();
      if (cookies != null) {
         for (Cookie cookie : cookies) {
            if ("visitedBefore".equals(cookie.getName())) {
               isFirstTimeVisitor = false;
               break;
            }
         }
      }
      Cookie visitedCookie = new Cookie("visitedBefore", "true");
      response.addCookie(visitedCookie);
      out.println("<html><head><title>Welcome</title></head><body>");
      if (isFirstTimeVisitor) {
         out.println("<h2>Welcome, you are visiting for the first time</h2>");
      } else {
         out.println("<h2>Welcome Back</h2>");
      }
      out.println("</body></html>");
   }
}
```





3

'WELCOME BACK'

Experiment – 25

Aim :- Create User Registration using Jsp, Servlet and Jdbc.

Code:

Step 1: Create database table for member

```
CREATE TABLE `member` (
`uname` varchar(45) NOT NULL,
`password` varchar(45) DEFAULT NULL,
`email` varchar(45) DEFAULT NULL,
`phone` varchar(45) DEFAULT NULL,
PRIMARY KEY (`uname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;</code>
```

Step 2: Create a memberRegister.jsp for the user registration

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1" %>
<!DOCTYPE html>
<html>
<head>
  <meta charset="ISO-8859-1">
  <title>Insert title here</title>
</head>
<body>
  <form action="Register" method="post">
    <table>
      <tr>
        <td>User Name</td>
        <td><input type="text" name="uname"></td>
      </tr>
      <tr>
        <td>Password</td>
        <td><input type="password" name="password"></td>
      </tr>
      <tr>
        <td>Email</td>
        <td><input type="text" name="email"></td>
      </tr>
      <tr>
        <td>Phone</td>
        <td><input type="text" name="phone"></td>
      </tr>
      <tr>
        <td>Submit</td>
        <td><input type="submit" value="register"></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

Step 3: Create a dto class Member.java

```
public class Member {
  private String uname, password, email, phone;
  public Member() {
    super();
  }
  public Member(String uname, String password, String email, String phone) {
    super();
    this.uname = uname;
    this.password = password;
    this.email = email;
    this.phone = phone;
  }
  public String getUname() {
    return uname;
  }
  public void setUname(String uname) {
    this.uname = uname;
  }
  public String getPassword() {
    return password;
  }
  public void setPassword(String password) {
    this.password = password;
  }
  public String getEmail() {
    return email;
  }
  public void setEmail(String email) {
    this.email = email;
  }
  public String getPhone() {
    return phone;
  }
  public void setPhone(String phone) {
    this.phone = phone;
  }
}
```

Step 4: Create a Servlet named Register.java

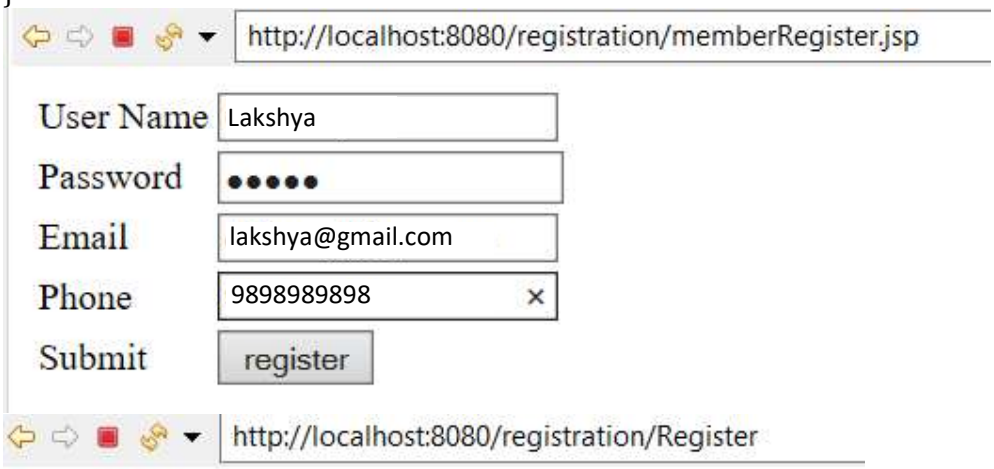
```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/Register")
public class Register extends HttpServlet {
  private static final long serialVersionUID = 1L;
  public Register() {
    super();
  }
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.getWriter().append("Served at: ").append(request.getContextPath());
  }
  protected void doPost(HttpServletRequest request,HttpServletResponse response) throws
ServletException, IOException {
    String uname = request.getParameter("uname");
    String password = request.getParameter("password");
    String email = request.getParameter("email");
    String phone = request.getParameter("phone");
    Member member = new Member(uname, password, email, phone);
    RegisterDao rdao = new RegisterDao();
    String result = rdao.insert(member);
    response.getWriter().println(result);
  }
}
```

Step 5: Create a Dao class RegisterDao.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class RegisterDao {
  private String dburl = "jdbc:mysql://localhost:3306/userdb";
  private String dbuname = "root";
  private String dbpassword = "mysql";
  private String dbdriver = "com.mysql.jdbc.Driver";
  public void loadDriver(String dbDriver) {
    try {
      Class.forName(dbDriver);
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
  public Connection getConnection() {
    Connection con = null;
    try {
      con = DriverManager.getConnection(dburl, dbuname, dbpassword);
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return con;
  }
  public String insert(Member member) {
    loadDriver(dbdriver);
    Connection con = getConnection();
    String sql = "insert into member values(?,?,?,?)";
    String result = "Data Entered Successfully";
    try {
      PreparedStatement ps = con.prepareStatement(sql);
      ps.setString(1, member.getUname());
      ps.setString(2, member.getPassword());
      ps.setString(3, member.getEmail());
      ps.setNString(4, member.getPhone());
```

```
    ps.executeUpdate();
   } catch (SQLException e) {
    result = "Data Not Entered Successfully";
    e.printStackTrace();
   }
   return result;
  }
}
```



Experiment – 26

Aim :- Create Employee Registration Form using a combination of JSP, Servlet, JDBC and MySQL Database

Overview:

Thread priorities in multithreading range from 1 to 10, with 1 being the lowest priority and 10 being the highest. Higher priority threads are scheduled for execution before lower priority threads. getPriority() retrieves a thread's priority, and setPriority() sets it. Use priorities judiciously, as they may not always have a significant impact and can lead to platform-specific behavior.

Code:

MySQL Database Setup

```
CREATE TABLE `employee` (
  `id` int(3) NOT NULL,
  `first_name` varchar(20) DEFAULT NULL,
  `last_name` varchar(20) DEFAULT NULL,
  `username` varchar(250) DEFAULT NULL,
  `password` varchar(20) DEFAULT NULL,
  `address` varchar(45) DEFAULT NULL,
  `contact` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Create a JavaBean - Employee.java:

```
import java.io.Serializable;
public class Employee implements Serializable {
   private static final long serialVersionUID = 1 L;
   private String firstName;
   private String lastName;
   private String username;
   private String password;
   private String address;
   private String contact;
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;  }
   public String getLastName() {
      return lastName; }
   public void setLastName(String lastName) {
      this.lastName = lastName; }
   public String getUsername() {
      return username;  }
   public void setUsername(String username) {
      this.username = username;  }
   public String getPassword() {
      return password;  }
```

```
   public void setPassword(String password) {
      this.password = password;  }
   public String getAddress() {
      return address;  }
   public void setAddress(String address) {
      this.address = address;  }
   public String getContact() {
      return contact; }
   public void setContact(String contact) {
      this.contact = contact;
 }
}
```

Create an EmployeeDao.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import net.javaguides.registration.model.Employee;
public class EmployeeDao {
   public int registerEmployee(Employee employee) throws ClassNotFoundException {
      String INSERT_USERS_SQL = "INSERT INTO employee" +
         "  (id, first_name, last_name, username, password, address, contact) VALUES " +
         " (?, ?, ?, ?, ?,?,?);";
      int result = 0;
      Class.forName("com.mysql.jdbc.Driver");
      try (Connection connection = DriverManager
         .getConnection("jdbc:mysql://localhost:3306/demo?useSSL=false", "root", "root");
         PreparedStatement preparedStatement =
connection.prepareStatement(INSERT_USERS_SQL)) {
         preparedStatement.setInt(1, 1);
         preparedStatement.setString(2, employee.getFirstName());
         preparedStatement.setString(3, employee.getLastName());
         preparedStatement.setString(4, employee.getUsername());
         preparedStatement.setString(5, employee.getPassword());
         preparedStatement.setString(6, employee.getAddress());
         preparedStatement.setString(7, employee.getContact());
         System.out.println(preparedStatement);
         result = preparedStatement.executeUpdate();
      } catch (SQLException e) {
         printSQLException(e);
      }
      return result; }
   private void printSQLException(SQLException ex) {
      for (Throwable e: ex) {
         if (e instanceof SQLException) {
            e.printStackTrace(System.err);
            System.err.println("SQLState: " + ((SQLException) e).getSQLState());
            System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
            System.err.println("Message: " + e.getMessage());
            Throwable t = ex.getCause();
            while (t != null) {
               System.out.println("Cause: " + t);
               t = t.getCause();
            } } } } }
```

Create an EmployeeServlet.java

```
package net.javaguides.employeemanagement.web;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.javaguides.employeemanagement.dao.EmployeeDao;
import net.javaguides.employeemanagement.model.Employee;
@WebServlet("/register")
public class EmployeeServlet extends HttpServlet {
   private static final long serialVersionUID = 1 L;
   private EmployeeDao employeeDao;
   public void init() {
      employeeDao = new EmployeeDao();  }
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      String firstName = request.getParameter("firstName");
      String lastName = request.getParameter("lastName");
      String username = request.getParameter("username");
      String password = request.getParameter("password");
      String address = request.getParameter("address");
      String contact = request.getParameter("contact");
      Employee employee = new Employee();
      employee.setFirstName(firstName);
      employee.setLastName(lastName);
      employee.setUsername(username);
      employee.setPassword(password);
      employee.setContact(contact);
      employee.setAddress(address);
      try {
         employeeDao.registerEmployee(employee);
      } catch (Exception e) {
         e.printStackTrace();  }
      response.sendRedirect("employeedetails.html"); } }
```

Create an employeeregister.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="ISO-8859-1">
   <title>Insert title here</title>
</head>
<body>
   <div align="center">
      <h1>Employee Register Form</h1>
      <form action="register" method="post">
         <table style="with: 80%">
            <tr> <td>First Name</td>
               <td><input type="text" name="firstName" /></td> </tr>
            <tr><td>Last Name</td>
               <td><input type="text" name="lastName" /></td></tr>
```

```
        <tr><td>UserName</td>
        <td><input type="text" name="username" /></td></tr>
        <tr><td>Password</td>
        <td><input type="password" name="password" /></td>   </tr>
        <tr><td>Address</td>
          <td><input type="text" name="address" /></td></tr>

        <tr> <td>Contact No</td>
          <td><input type="text" name="contact" /></td></tr>
      </table>
      <input type="submit" value="Submit" />
    </form>
  </div>
</body>
</html>
```

Create an employeedetail.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 <h1>Employee successfully registered !</h1>
</body>
</html>
```

Lakshya

Department of Computer Science and Engineering

Rubrics for Lab Assessment

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

INDEX

Name: Kshitij Tanwar

Enrollment number : 03114812721

Branch: Computer Science and Technology

Group: 6FSD-2c

S. No.	Experiment	Date of Performance	Date of Checking	Signature	Remark
1					
2					
3					
4					
5					
6					
7					
8					

9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					

20					
21					
22					
23					
24					
25					
26					

PROGRAM-1

Aim: Write a program to implement parametrized constructor in Java.

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.

Overview: A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

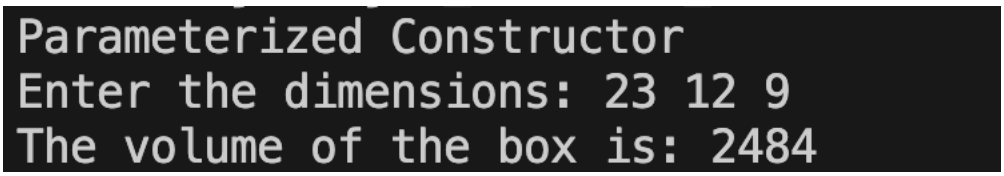
Syntax of Parameterized Constructor in Java:

```
class ClassName {
   TypeName variable1;
   TypeName variable2;
   ClassName(TypeName variable1, TypeName variable2) {
   this.variable1 = variable1;
   this.variable2 = variable2;
   }
}
```

Source Code:

```
package Java;
import java.util.Scanner;

class Box {
   int width,height,depth;
   Box (int w, int h, int d) {
      width=w;
      height=h;
      depth=d;
   }
   int calVolume() {
      return width*height*depth;
   }
}

public class Program1 {
   public static void main(String[] args) {
      System.out.println("Parameterized Constructor");
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the dimensions: ");
      int w=s.nextInt();
      int h=s.nextInt();
      int d=s.nextInt();
      Box b = new Box(w,h,d);
      System.out.println("The volume of the box is: "+ b.calVolume());
   }
}
```

Output:


Parameterized Constructor
Enter the dimensions: 23 12 9
The volume of the box is: 2484

PROGRAM-2

Aim: Write a program to implement method overriding in Java.

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.

Overview: If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding. Method overriding is one of the ways by which Java achieves Run Time Polymorphism.

Syntax of Method Overriding in Java:

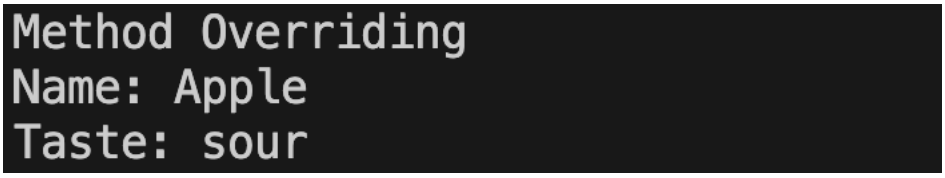
```
class ClassName1 {
   TypeName variable1;
   TypeName variable2;
   returnType method (parameters) {}
}
Class ClassName2 extends ClassName1{
      returnType method (parameters) {}
}
```

Source Code:

```
package Java;

class Fruit {
   String name,taste;
   int size;
   void eat() {
      System.out.println("Inside Fruit class");
   }
}
class Apple extends Fruit {
   Apple (String n, String t, int s) {
      name=n;
      taste=t;
      size=s;
   }
   void eat() {
      System.out.println("Name: "+ name +"\nTaste: "+ taste);
   }
}
class Orange extends Fruit {
   Orange (String n, String t, int s) {
      name=n;
      taste=t;
      size=s;
   }
   void eat() {
      System.out.println("Name: "+ name +"\nTaste: "+ taste);
   }
```

```
}

public class Program2 {
    public static void main(String[] args) {
        Apple a = new Apple("Apple","sour",10);
        System.out.println("Method Overriding");
        a.eat();
    }
}
```

Output:



PROGRAM-3

Aim: Write a program to implement polymorphism in Java.

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();

Overview: Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms. There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.

Syntax of polymorphism in Java is:

```
class ClassName1 {}
class ClassName2 extends ClassName1 {}
ClassName1 cn1 = new ClassName2();
```

Source Code:

```
package Java;

class Shape {
   void draw() {
      System.out.println("Drawing Shape");
   }
   void erase() {
      System.out.println("Erasing Shape");
   }
}
class Circle extends Shape {
   void draw() {
      System.out.println("Drawing Circle");
   }
   void erase() {
      System.out.println("Erasing Circle");
   }
}
class Triangle extends Shape {
   void draw() {
      System.out.println("Drawing Triangle");
   }
   void erase() {
      System.out.println("Erasing Triangle");
   }
}
class Square extends Shape {
   void draw() {
```

```
        System.out.println("Drawing Square");
    }
    void erase() {
        System.out.println("Erasing Square");
    }
}

public class Program3 {
    public static void main(String[] args) {
        Shape c = new Circle();
        Shape t = new Triangle();
        Shape s = new Square();
        c.draw();   c.erase();
        t.draw();   t.erase();
        s.draw();   s.erase();
    }
}
```

Output:

```
Polymorphism in Java
Drawing Circle
Erasing Circle
Drawing Triangle
Erasing Triangle
Drawing Square
Erasing Square
```

PROGRAM-4

Aim: Write a program to implement Exception handling in Java.

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).

Overview: An exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed.Java exceptions cover almost all the general types of exceptions that may occur in the programming. However, we sometimes need to create custom exceptions.

Syntax of Custom Exception in Java is:

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

Source Code:

```
package Java;
import java.util.Scanner;

class ExceptionClass extends Exception {
   public String toString() {
      return "Error! Marks are not in the range(1,100)";
   }
}
public class Program4 {
   Scanner s = new Scanner(System.in);
   String name;
   float sum=0;
   int marks[]=new int[3];
   void input() throws ExceptionClass {
      System.out.print("Enter name of the Student: ");
      name=s.nextLine();
      System.out.print("Enter the marks of "+ name +" in Physics, Chemistry and Maths: ");
      for (int i=0;i<3; i++) {
         marks[i]=Integer.parseInt(s.next());
         if (marks[i]<0 || marks[i]>100) {
            throw new ExceptionClass();
         }
         sum+=marks[i];
      }
      System.out.println("Average of marks of "+ name +" is "+ sum/3);
   }
   public static void main(String[] args) {
      System.out.println("NumberFormatException, Custom Exception in Java");
      Program4 a=new Program4();
```

```
        Program4 b=new Program4();
        try {
            a.input();
            b.input();
        }
        catch (NumberFormatException n) {
            System.out.println("NumberFormatException caught "+ n.getMessage());
        }
        catch (ExceptionClass e) {
            System.out.println(e);
        }
    }
}
```

Output:

```
NumberFormatException, Custom Exception in Java
Enter name of the Student: Abhishek
Enter the marks of Abhishek in Physics, Chemistry and Maths: 88 91 90
Average of marks of Abhishek is 89.666664
Enter name of the Student: Varun
Enter the marks of Varun in Physics, Chemistry and Maths: 90 94 78
Average of marks of Varun is 87.333336
```

```
NumberFormatException, Custom Exception in Java
Enter name of the Student: Abhishek
Enter the marks of Abhishek in Physics, Chemistry and Maths: 88 91 sw
NumberFormatException caught For input string: "sw"
```

```
NumberFormatException, Custom Exception in Java
Enter name of the Student: Abhishek
Enter the marks of Abhishek  in Physics, Chemistry and Maths: 88 91 -2
Error! Marks are not in the range(1,100)
```

PROGRAM-5

Aim: Write a program to implement Exception handling in Java.

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

Overview: An exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed. The ArrayIndexOutOfBoundsException occurs whenever we are trying to access any item of an array at an index which is not present in the array. In other words, the index may be negative or exceed the size of an array. The NumberFormatException is thrown when we try to convert a string into a numeric value such as float or integer, but the format of the input string is not appropriate or illegal.

Syntax to handle any Exception in Java is:

```
try {
    // Block of code to try
}
catch (Exception e) {
    // Block of code to handle errors
}
```

Source Code:

```
import java.util.Scanner;

public class Program5 {
   public static void main(String[] args) {
      System.out.println("ArrayIndexOutOfBoundsException, NumberFormatException in Java");
      try {
         Scanner s = new Scanner(System.in);
         System.out.print("Enter the number of elements in the array: ");
         int n=Integer.parseInt(s.next());
         int arr[]=new int[n];
         System.out.print("Enter the elements of the array: ");
         for (int i=0;i<n;i++) {
            arr[i]=Integer.parseInt(s.next());
         }
         System.out.print("Enter the index at which element is to be found: ");
         int a=Integer.parseInt(s.next());
         System.out.println("Element at index "+ a +" is "+ arr[a]);
      }
      catch (ArrayIndexOutOfBoundsException a) {
         System.out.println("ArrayIndexOutOfBoundsException caught "+ a.getMessage());
      }
      catch (NumberFormatException n) {
         System.out.println("NumberFormatException caught "+ n.getMessage());
      }
   }
}
```

Output:

```
ArrayIndexOutOfBoundsException, NumberFormatException in Java
Enter the number of elements in the array: 4
Enter the elements of the array: 2 1 4 10
Enter the index at which element is to be found: 3
Element at index 3 is 10
```

```
ArrayIndexOutOfBoundsException, NumberFormatException in Java
Enter the number of elements in the array: 4
Enter the elements of the array: 2 1 4 10
Enter the index at which element is to be found: 5
ArrayIndexOutOfBoundsException caught Index 5 out of bounds for length 4
```

```
ArrayIndexOutOfBoundsException, NumberFormatException in Java
Enter the number of elements in the array: 4
Enter the elements of the array: 2 1 sw 10
NumberFormatException caught For input string: "sw"
```

Experiment - 6

Aim :- Implement Datagram UDP socket programming in java

Overview:

The provided Java code demonstrates Datagram UDP socket programming. The UDP server listens on port 9876, receiving messages from clients and optionally sending a response. The UDP client sends a "Hello" message to the server, prints the sent message, and optionally receives a response from the server. This implementation enables simple communication between a UDP server and client using datagrams in Java

Code:

UDP Server

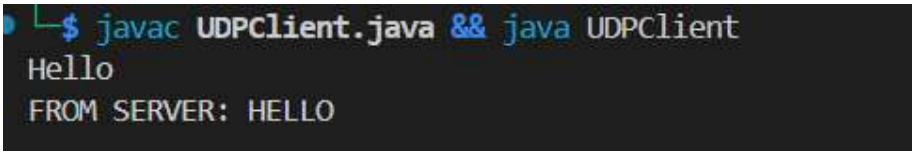
```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception {
    DatagramSocket serverSocket = new DatagramSocket(9876);
    byte[] receiveData = new byte[1024];
    byte[] sendData;
    while (true) {
      DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
      serverSocket.receive(receivePacket);
      String sentence = new String(receivePacket.getData(), 0,
receivePacket.getLength());
      InetAddress IPAddress = receivePacket.getAddress();
      int port = receivePacket.getPort();
      String capitalizedSentence = sentence.toUpperCase();
      sendData = capitalizedSentence.getBytes();
      DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
          IPAddress, port);
      serverSocket.send(sendPacket);
    }
  }
}
```

UDP Server

```
import java.io.*;
import java.net.*;

class UDPClient {
  public static void main(String args[]) throws Exception {
    BufferedReader inFromUser = new BufferedReader(
```

```
        new InputStreamReader(System.in));
    DatagramSocket clientSocket = new DatagramSocket();
    InetAddress IPAddress = InetAddress.getByName("localhost");
    byte[] sendData;
    byte[] receiveData = new byte[1024];
    String sentence = inFromUser.readLine();
    sendData = sentence.getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
        IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence = new String(receivePacket.getData(),
        0, receivePacket.getLength());
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
  }
}
```

Output:



Experiment - 7

Aim :- Implement Socket programming for TCP in Java Server and Client Sockets.

Overview:

This Java code implements TCP socket programming with a server and client. The TCP server listens on a specified port, accepts client connections, and can handle multiple clients concurrently. The client connects to the server, sends a message ("Hello, TCP Server!"), and receives a response. This implementation facilitates communication between a TCP server and client through sockets in Java.

Code:

TCP Server

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception {
    String clientSentence;
    String capitalizedSentence;
    ServerSocket welcomeSocket = new ServerSocket(6789);
    while (true) {
      Socket connectionSocket = welcomeSocket.accept();
      BufferedReader inFromClient = new BufferedReader(
        new InputStreamReader(connectionSocket.getInputStream())
      );
      DataOutputStream outToClient = new DataOutputStream(
        connectionSocket.getOutputStream()
      );
      clientSentence = inFromClient.readLine();
      capitalizedSentence = clientSentence.toUpperCase() + '\n';
      outToClient.writeBytes(capitalizedSentence);
    }
  }
}
```

TCP Client

```
import java.io.*;
import java.net.*;

class TCPClient {

  public static void main(String argv[]) throws Exception {
```

```
    String sentence;
    String modifiedSentence;
    BufferedReader inFromUser = new BufferedReader(
      new InputStreamReader(System.in)
    );
    Socket clientSocket = new Socket("localhost", 6789);
    DataOutputStream outToServer = new DataOutputStream(
      clientSocket.getOutputStream()
    );
    BufferedReader inFromServer = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream())
    );
    sentence = inFromUser.readLine();
    outToServer.writeBytes(sentence + '\n');
    modifiedSentence = inFromServer.readLine();
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
  }
}
```

Output:

Experiment - 8

Aim :- Socket Programming

Overview:

This Java code represents a basic implementation of TCP socket programming with a server and client. The server waits for a connection, accepts a client, and reads messages until the client sends "Over." The client connects to the server, sends messages from the terminal to the server until "Over" is entered, and then closes the connection. This practical demonstrates simple bidirectional communication between a TCP server and client, establishing a connection and exchanging messages over sockets in a networked environment.

Code:

Server

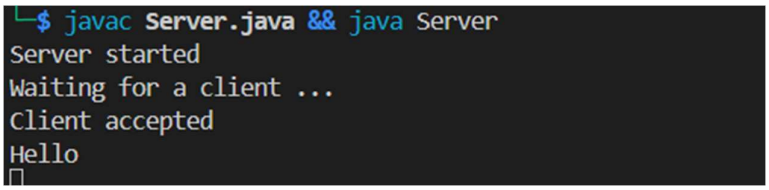
```
// A Java program for a Server
import java.io.*;
import java.net.*;

public class Server {

  //initialize socket and input stream
  private Socket socket = null;
  private ServerSocket server = null;
  private DataInputStream in = null;

  // constructor with port
  public Server(int port) {
    // starts server and waits for a connection
    try {
      server = new ServerSocket(port);
      System.out.println("Server started");
      System.out.println("Waiting for a client ...");
      socket = server.accept();
      System.out.println("Client accepted");
      // takes input from the client socket
      in =
        new DataInputStream(new BufferedInputStream(socket.getInputStream()));
      String line = "";
      // reads message from client until "Over" is sent
      while (!line.equals("Over")) {
        try {
          line = in.readUTF();
          System.out.println(line);
        } catch (IOException i) {
          System.out.println(i);
```

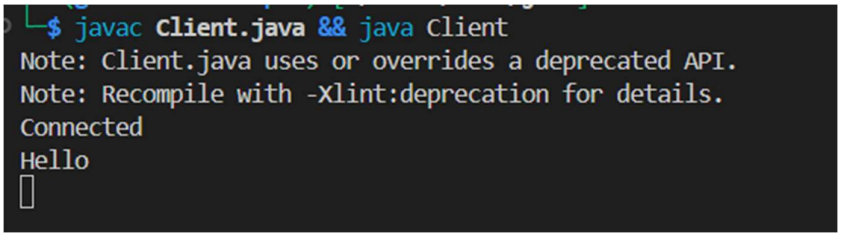
```
      }
     }
     System.out.println("Closing connection");
     // close connection
     socket.close();
     in.close();
    } catch (IOException i) {
     System.out.println(i);
    }
  }

  public static void main(String args[]) {
   Server server = new Server(5000);
  }
}
```

Client

```
// A Java program for a Client
import java.io.*;
import java.net.*;

public class Client {

  // initialize socket and input output streams
  private Socket socket = null;
  private DataInputStream input = null;
  private DataOutputStream out = null;

  // constructor to put ip address and port
  public Client(String address, int port) {
   // establish a connection
   try {
    socket = new Socket(address, port);
    System.out.println("Connected");
    // takes input from terminal
    input = new DataInputStream(System.in);
    // sends output to the socket
    out = new DataOutputStream(socket.getOutputStream());
   } catch (UnknownHostException u) {
    System.out.println(u);
    return;
   } catch (IOException i) {
    System.out.println(i);
    return;
   }
   // string to read message from input
   String line = "";
```

```
    // keep reading until "Over" is input
    while (!line.equals("Over")) {
      try {
        line = input.readLine();
        out.writeUTF(line);
      } catch (IOException i) {
        System.out.println(i);
      }
    }
    // close the connection
    try {
      input.close();
      out.close();
      socket.close();
    } catch (IOException i) {
      System.out.println(i);
    }
  }

  public static void main(String args[]) {
    Client client = new Client("127.0.0.1", 5000);
  }
}
```





PROGRAM-9

Aim: Implement Producer-Consumer Problem using multithreading.

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

- The producer's job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

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

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

Source Code:

```
import java.util.LinkedList;

public class Program9 {
   public static void main(String[] args) throws InterruptedException {
      System.out.println("Producer-Consumer Problem using Multithreading");
      final PC pc = new PC();
      Thread t1 = new Thread(() -> {
         try {
            pc.produce();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      });
      Thread t2 = new Thread(() -> {
         try {
            pc.consume();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      });
      t1.start();
      t2.start();
      t1.join();
      t2.join();
   }
   public static class PC {
      LinkedList<Integer> list = new LinkedList<>();
      int capacity = 2;
      public void produce() throws InterruptedException {
         int value = 0;
         while (true) {
            synchronized (this) {
               while (list.size() == capacity)
                  wait();
```

```
                System.out.println("Producer produced-" + value);
                list.add(value++);
                notify();
                Thread.sleep(1000);
            }
        }
    }
    public void consume() throws InterruptedException {
        while (true) {
            synchronized (this) {
                while (list.size() == 0)
                    wait();
                int val = list.removeFirst();
                System.out.println("Consumer consumed-" + val);
                notify();
                Thread.sleep(1000);
            }
        }
    }
}
```

Output:

```
Producer-Consumer Problem using Multithreading
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Producer produced-5
Consumer consumed-4
Consumer consumed-5
```

PROGRAM-10

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

Overview: Priorities in threads is a concept where each thread is having a priority which in layman's language one can say every object is having priority here which is represented by numbers ranging from 1 to 10.
The default priority is set to 5 as excepted. (public static int NORM_PRIORITY)
Minimum priority is set to 1. (public static int MIN_PRIORITY)
Maximum priority is set to 10. (public static int MAX_PRIORITY)

Source Code:

```
public class Program10 extends Thread {
   public void run(){
      System.out.println("Inside run method");
   }
   public static void main(String[] args) {
      System.out.println("Priorities in Multithreading");
      Program10 t1 = new Program10();
      Program10 t2 = new Program10();
      Program10 t3 = new Program10();
      System.out.println("t1 thread priority: " + t1.getPriority());
      System.out.println("t2 thread priority: " + t2.getPriority());
      System.out.println("t3 thread priority: " + t3.getPriority());
      t1.setPriority(2);
      t2.setPriority(5);
      t3.setPriority(8);
      System.out.println("t1 thread priority: " + t1.getPriority());
      System.out.println("t2 thread priority: " + t2.getPriority());
      System.out.println("t3 thread priority: " + t3.getPriority());
      System.out.println("Currently Executing Thread: "+ Thread.currentThread().getName());
      System.out.println( "Main thread priority: "+ Thread.currentThread().getPriority());
      Thread.currentThread().setPriority(10);
      System.out.println("Main thread priority: "+ Thread.currentThread().getPriority());
   }
}
```

Output:

```
Priorities in Multithreading
t1 thread priority: 5
t2 thread priority: 5
t3 thread priority: 5
t1 thread priority: 2
t2 thread priority: 5
t3 thread priority: 8
Currently Executing Thread: main
Main thread priority: 5
Main thread priority: 10
```

PROGRAM-11

Aim: Illustrate Deadlock in multithreading.

Overview: Deadlock in Java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.
Synchronization keyword is used to make the class or method thread-safe which means only one thread can have lock of synchronized method and use it, other threads have to wait till the lock releases and anyone of them acquire that lock.

Source Code:

Demonstrating Deadlock Situation:

```
public class Program11a {
   public static Object Lock1 = new Object();
   public static Object Lock2 = new Object();
   public static void main(String args[]) {
      System.out.println("Deadlock Situation");
      ThreadDemo1 T1 = new ThreadDemo1();
      ThreadDemo2 T2 = new ThreadDemo2();
      T1.start();
      T2.start();
   }
   private static class ThreadDemo1 extends Thread {
      public void run() {
         synchronized (Lock1) {
            System.out.println("Thread 1: Holding lock 1...");
            try {
               Thread.sleep(10);
            } catch (InterruptedException e) {}
            System.out.println("Thread 1: Waiting for lock 2...");
            synchronized (Lock2) {
               System.out.println("Thread 1: Holding lock 1 & 2...");
            }
         }
      }
   }
   private static class ThreadDemo2 extends Thread {
      public void run() {
         synchronized (Lock2) {
            System.out.println("Thread 2: Holding lock 2...");
            try {
               Thread.sleep(10);
            } catch (InterruptedException e) {}
            System.out.println("Thread 2: Waiting for lock 1...");
            synchronized (Lock1) {
               System.out.println("Thread 2: Holding lock 1 & 2...");
            }
         }
      }
   }
}
```

Output:

```
Deadlock Situation
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 1: Waiting for lock 2...
Thread 2: Waiting for lock 1...
```

Source Code:

Deadlock Solution:

```
public class Program11b {
    public static Object lock1 = new Object();
    public static Object lock2 = new Object();
    public static void main(String[] args) {
        System.out.println("Deadlock Solution");
        new Thread1().start();
        new Thread2().start();
    }
    private static class Thread1 extends Thread {
        @Override
        public void run() {
            synchronized (lock1) {
                System.out.println("Thread-1 acquired lock1");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    System.out.println("Thread-1 interrupted.");
                }
                System.out.println("Thread-1 waiting for lock2");
                synchronized (lock2) {
                    System.out.println("Thread-1 acquired lock2");
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        System.out.println("Thread-1 interrupted.");
                    }
                }
                System.out.println("Thread-1 releases lock2");
            }
            System.out.println("Thread-1 releases lock1");
        }
    }
    private static class Thread2 extends Thread {
        @Override
        public void run() {
            synchronized (lock1) {
                System.out.println("Thread-2 acquired lock1");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    System.out.println("Thread-2 interrupted.");
                }
                System.out.println("Thread-2 waiting for lock2");
```

```
            synchronized (lock2) {
                System.out.println("Thread-2 acquired lock2");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    System.out.println("Thread-2 interrupted.");
                }
            }
            System.out.println("Thread-2 releases lock2");
        }
        System.out.println("Thread-2 releases lock1");
    }
  }
}
```

Output:

```
Deadlock Solution
Thread-1 acquired lock1
Thread-1 waiting for lock2
Thread-1 acquired lock2
Thread-1 releases lock2
Thread-1 releases lock1
Thread-2 acquired lock1
Thread-2 waiting for lock2
Thread-2 acquired lock2
Thread-2 releases lock2
Thread-2 releases lock1
```

Experiment – 12

Aim :- Implement Producer-Consumer Problem using multithreading.

Overview:

The Producer-Consumer problem epitomizes the challenge of coordinating concurrent threads sharing a bounded buffer. Producers generate items and deposit them, while consumers retrieve and process them. Synchronization mechanisms like semaphores or mutexes ensure orderly access to the buffer, preventing issues like overflow or underflow. Through this synchronization, threads collaborate efficiently, balancing resource utilization with system stability.

Code:

```
import java.util.LinkedList;

class ProducerConsumer {
  private final LinkedList<Integer> buffer = new LinkedList<>();
  private final int capacity = 5;
  private int producedCount = 0;
  private int consumedCount = 0;
  private final int totalIterations = 10; // Total number of iterations

  public void produce() throws InterruptedException {
    while (producedCount < totalIterations) {
      synchronized (this) {
        while (buffer.size() == capacity)
          wait();
        int value = producedCount++;
        System.out.println("Producer produced: " + value);
        buffer.add(value);
        notify();
      }
    }
  }

  public void consume() throws InterruptedException {
    while (consumedCount < totalIterations) {
      synchronized (this) {
        while (buffer.size() == 0)
          wait();
        int consumedValue = buffer.removeFirst();
        System.out.println("Consumer consumed: " + consumedValue);
        consumedCount++;
        notify();
      }
    }
  }
}

public class Exp12 {
```

```
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer();
        Thread producerThread = new Thread(() -> {
            try {
                pc.produce();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        Thread consumerThread = new Thread(() -> {
            try {
                pc.consume();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        producerThread.start();
        consumerThread.start();
    }
}
```

```
└─$ javac Exp12.java && java Exp12
Producer produced: 0
Producer produced: 1
Producer produced: 2
Producer produced: 3
Producer produced: 4
Consumer consumed: 0
Consumer consumed: 1
Consumer consumed: 2
Consumer consumed: 3
Consumer consumed: 4
Producer produced: 5
Producer produced: 6
Producer produced: 7
Producer produced: 8
Producer produced: 9
Consumer consumed: 5
Consumer consumed: 6
Consumer consumed: 7
Consumer consumed: 8
Consumer consumed: 9
```

Experiment – 13

Aim :- Illustrate Priorities in Multithreading via help of getPriority() and setPriority() method.

Overview:

Thread priorities in multithreading range from 1 to 10, with 1 being the lowest priority and 10 being the highest. Higher priority threads are scheduled for execution before lower priority threads. getPriority() retrieves a thread's priority, and setPriority() sets it. Use priorities judiciously, as they may not always have a significant impact and can lead to platform-specific behavior.

Code:

```
class Task extends Thread {

   public Task(String name) {
      super(name);
   }
   public void run() {
      for (int i = 0; i < 5; i++) {
         System.out.println(getName() + " is executing");
         try {
            Thread.sleep(100); // Simulate some work
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }  } }

public class Exp13 {
   public static void main(String[] args) {
      Task highPriorityThread = new Task("High Priority Thread");
      Task lowPriorityThread = new Task("Low Priority Thread");
      highPriorityThread.setPriority(Thread.MAX_PRIORITY);
      lowPriorityThread.setPriority(Thread.MIN_PRIORITY);
      highPriorityThread.start();
      lowPriorityThread.start();
   }
}
```

```
└─$ javac Exp13.java && java Exp13
High Priority Thread is executing
Low Priority Thread is executing
High Priority Thread is executing
Low Priority Thread is executing
Low Priority Thread is executing
High Priority Thread is executing
High Priority Thread is executing
Low Priority Thread is executing
Low Priority Thread is executing
High Priority Thread is executing
```

Experiment – 14

Aim :- Illustrate Deadlock in multithreading.

Overview: Deadlock in multithreading occurs when two or more threads are stuck indefinitely, each waiting for the other to release a resource. This results from improper synchronization, where threads hold resources needed by others. Prevention involves careful resource management and avoiding circular dependencies. Detection requires monitoring thread states, and resolution strategies include timeouts and avoiding resource contention. Proper design and synchronization are crucial to prevent deadlock in multithreaded applications.

Code:

```
public class Exp14 {
  private static final Object lock1 = new Object();
  private static final Object lock2 = new Object();
  public static void main(String[] args) {
    Thread thread1 = new Thread(() -> {
      synchronized (lock1) {
        System.out.println("Thread 1: Holding lock 1...");
        try {
          Thread.sleep(100); // Simulating some work
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        System.out.println("Thread 1: Waiting for lock 2...");
        synchronized (lock2) {
          System.out.println("Thread 1: Holding lock 1 and lock 2...");
        }
      }
    });
    Thread thread2 = new Thread(() -> {
      synchronized (lock2) {
        System.out.println("Thread 2: Holding lock 2...");
        try {
          Thread.sleep(100); // Simulating some work
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        System.out.println("Thread 2: Waiting for lock 1...");
        synchronized (lock1) {
          System.out.println("Thread 2: Holding lock 1 and lock 2...");
        }
      }
    });
    thread1.start();
    thread2.start();
  }
}
```

```
└─$ javac Exp14.java && java Exp14
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 2: Waiting for lock 1...
Thread 1: Waiting for lock 2...
^C
```

Experiment – 15

Aim :- Implement a program Java Bean to represent person details.

Overview: A Java Bean representing person details is a simple Java class that encapsulates information about a person. It typically includes private fields for attributes such as name, age, and address, along with getter and setter methods for accessing and modifying these attributes. This ensures data encapsulation and abstraction, allowing controlled access to the person's details while maintaining flexibility and ease of use within Java applications.

Code:

```
public class PersonExp15 {
  private String name;
  private int age;
  private String address;
  public PersonExp15(String name, int age, String address) {
    this.name = name;
    this.age = age;
    this.address = address;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  public String getAddress() {
    return address;
  }
  public void setAddress(String address) {
    this.address = address;
  }
  @Override
  public String toString() {
    return (
      "Person [name=" + name + ", age=" + age + ", address=" + address + "]"
    );
  }
  public static void main(String[] args) {
    PersonExp15 person = new PersonExp15("John Doe", 30, "123 Main St");
    System.out.println(person);
  }
}
```

```
└─$ javac PersonExp15.java && java PersonExp15
Person [name=John Doe, age=30, address=123 Main St]
```

Experiment – 16

Aim :- Write a program in java to demonstrate encapsulation in java beans.

Overview: Encapsulation in Java beans is a fundamental concept in object-oriented programming (OOP). It refers to the practice of bundling the data (attributes) and methods (functions) that operate on the data into a single unit or entity, known as a bean. Encapsulation helps in hiding the internal state of an object and only exposing necessary operations through methods.

Code:

```
public class Exp16 {
   private String name;
   private int age;
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      if (age >= 0) {
         this.age = age;
      } else {
         System.out.println("Age cannot be negative.");
      }
   }
   public static void main(String[] args) {
      Exp16 person = new Exp16();
     person.setName("Random Person xyz");
      person.setAge(45);
      System.out.println("Name: " + person.getName());
      System.out.println("Age: " + person.getAge());
      person.setAge(-5);
   }
}
```

```
└─$ javac Exp16.java && java Exp16
Name: Random Person xyz
Age: 45
Age cannot be negative.
```

Experiment – 17

Aim :- Create a database in MySQL using JSP and perform insertion and retrieval operations.

Overview: Creating a database in MySQL using JSP involves several steps. First, you need to set up the MySQL server and create a database using the MySQL command-line interface or a graphical tool like phpMyAdmin. Next, you can connect your JSP application to the MySQL database using JDBC (Java Database Connectivity) by including the MySQL JDBC driver in your project. In your JSP code, you can write SQL queries to insert data into the database using INSERT statements and retrieve data using SELECT statements. Finally, you can display the retrieved data on your JSP pages. This integration allows for dynamic content generation and interaction with a persistent data store.

Code:

SQL Commands:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL
);
```

JSP Files: index.jsp: This page contains a form to insert user data into the database.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Insert User</title>
</head>
<body>
  <h2>Insert User</h2>
  <form action="insert.jsp" method="post">
    Username: <input type="text" name="username"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

index.jsp: This page inserts user data into the database.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

```
<%@ page import="java.sql.*" %>
<%@ page import="java.io.PrintWriter" %>
<%@ page import="java.io.IOException" %>
<%
String username = request.getParameter("username");
String email = request.getParameter("email");
try {
   Class.forName("com.mysql.jdbc.Driver");
   Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase",
"root", "");
   Statement stmt = con.createStatement();
   String sql = "INSERT INTO users (username, email) VALUES (?, ?)";
   PreparedStatement pstmt = con.prepareStatement(sql);
   pstmt.setString(1, username);
   pstmt.setString(2, email);
   pstmt.executeUpdate();
   out.println("User inserted successfully.");
   pstmt.close();
   con.close();
} catch (Exception e) {
   out.println("Error: " + e.getMessage()); }
%>
```

retrieve.jsp: This page retrieves user data from the database and displays it.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<%@ page import="java.sql.*" %>
<%@ page import="java.io.PrintWriter" %>
<%@ page import="java.io.IOException" %>
<%
try {
   Class.forName("com.mysql.jdbc.Driver");
   Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase",
"root", "");
   Statement stmt = con.createStatement();
   String sql = "SELECT * FROM users";
   ResultSet rs = stmt.executeQuery(sql);
   out.println("<h2>User List</h2>");
   out.println("<table border='1'><tr><th>ID</th><th>Username</th><th>Email</th></tr>");
   while (rs.next()) {
      out.println("<tr><td>" + rs.getInt("id") + "</td><td>" + rs.getString("username") +
"</td><td>" + rs.getString("email") + "</td></tr>");
   }
   out.println("</table>");
   rs.close();
   stmt.close();
```

Name

Name

Email

Email

Phone

Phone

Password

Password

Submit

Account Already Exists. Login Here

Experiment – 18

Aim :- Create a Java JSP login and Sign Up form with Session using MySQL.

Overview: A Java JSP login and sign-up form with session management using MySQL provides secure user authentication and registration. JSP pages handle user input, connecting to MySQL for data storage. Session management ensures user persistence across pages. Security measures like hashing passwords prevent data breaches. This system enhances web application functionality and security.

Code:

```
DOCTYPE html>
<html>
<head>
  <title>Form Validation with Regular Expressions</title>
</head>
<body>
<h2>User Registration</h2>
<form method="post" action="processData.jsp" onsubmit="return validateForm();">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required>
  <br>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>
  <br>
  <label for="password">Password:</label>
  <input type="password" id="password" name="password" required>
  <br>
  <input type="submit" value="Submit">
</form>
<script>
  function validateForm() {
    var username = document.getElementById("username").value;
    var email = document.getElementById("email").value;
    var password = document.getElementById("password").value;
    var usernamePattern = /^[a-zA-Z0-9_]{5,}$/; // Alphanumeric and underscore, minimum
length 5
    var emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/; // Email pattern
    var passwordPattern = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$/; // Minimum 8
characters, at least one uppercase letter, one lowercase letter, and one number
    if (!usernamePattern.test(username)) {
      alert("Username must be alphanumeric and at least 5 characters long.");
      return false;
    }
    if (!emailPattern.test(email)) {
      alert("Please enter a valid email address.");
      return false;
    }
    if (!passwordPattern.test(password)) {
      alert("Password must be at least 8 characters long, contain at least one uppercase letter,
one lowercase letter, and one number.");
      return false;
    }
```

```
    return true;
  }
</script>
</body>
</html>
```

User Registration

Username:

Email:

Password:

Submit

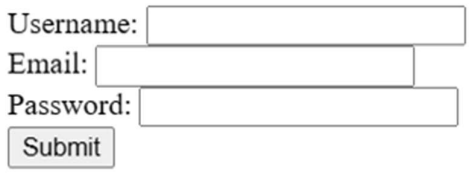
Experiment – 19

Aim :- Implement Regular Expressions validation before submitting data in JSP.

Overview: Regular Expressions validation in JSP involves verifying input data against predefined patterns before submission. Utilizing regex patterns, it ensures data conformity to desired formats, such as email addresses or alphanumeric strings. Implementation typically occurs on the server-side, enhancing security by preventing malicious or erroneous data from reaching backend systems. This validation process helps maintain data integrity and improves user experience by prompting corrective actions for invalid inputs.

Code:

```
import java.util.regex.*;

public class FormValidationServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String username = request.getParameter("username");
    String email = request.getParameter("email");
    String usernameRegex = "[a-zA-Z0-9_]+";
    String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-
zA-Z]{2,7}$";
    Pattern usernamePattern = Pattern.compile(usernameRegex);
    Pattern emailPattern = Pattern.compile(emailRegex);
    boolean isValidUsername = usernamePattern.matcher(username).matches();
    boolean isValidEmail = emailPattern.matcher(email).matches();
    if (isValidUsername && isValidEmail) {
      response.sendRedirect("success.jsp");
    } else {
      request.setAttribute("error", "Invalid username or email format");
      request.getRequestDispatcher("/form.jsp").forward(request, response);
    }
  }
}
```



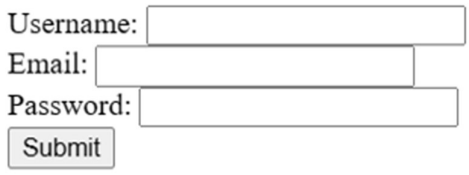
Experiment – 20

Aim :- Implement Customizable adapter class in a registration form in JSP

Overview: A customizable adapter class for a registration form in JSP acts as an intermediary between the JSP page and the database. It encapsulates the logic for user registration, including validation and database interaction. The adapter class typically provides methods to validate user input, connect to a database using JDBC, and execute SQL queries to store user information. By using an adapter class, the registration process becomes modular and easier to manage

Code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
public class RegistrationAdapter {
  private static final String DB_URL = "jdbc:mysql://localhost:3306/mydatabase";
  private static final String DB_USER = "root";
  private static final String DB_PASSWORD = "password";
  public boolean registerUser(String username, String password, String email) {
    boolean isSuccess = false;
    try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
      String query = "INSERT INTO users (username, password, email) VALUES (?, ?, ?)";
      try (PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString(1, username);
        stmt.setString(2, password);
        stmt.setString(3, email);
        int rowsInserted = stmt.executeUpdate();
        if (rowsInserted > 0) {
          isSuccess = true;
        }
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return isSuccess;
  }
}
```



Experiment – 21

Aim :- 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.

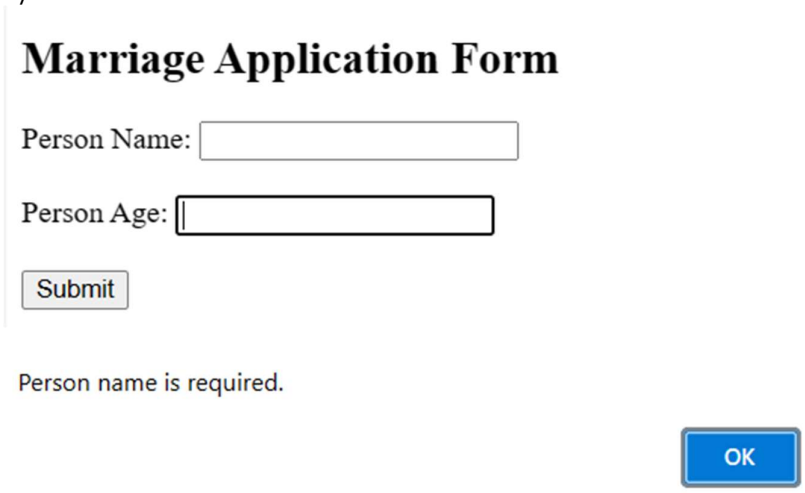
3. Personage is required.

4. Person age must be a numeric value. 5. Personage must be there between 1 to 125

Code:

```
DOCTYPE html>
<html>
<head>
  <title>Marriage Application Form</title>
  <script>
    function validateForm() {
      var name = document.forms["marriageForm"]["personName"].value;
      var age = document.forms["marriageForm"]["personAge"].value;
      if (name == "") {
        alert("Person name is required.");
        return false;
      }
      if (name.length < 5) {
        alert("Person's name must have a minimum of 5 characters.");
        return false;
      }
      if (age == "") {
        alert("Person age is required.");
        return false;
      }
      if (isNaN(age)) {
        alert("Person age must be a numeric value.");
        return false;
      }
      if (age < 1 || age > 125) {
        alert("Person age must be between 1 to 125.");
        return false;
      }
      return true;
    }
  </script>
</head>
<body>
  <h2>Marriage Application Form</h2>
  <form name="marriageForm" onsubmit="return validateForm()">
    <label for="personName">Person Name:</label>
    <input type="text" id="personName" name="personName"><br><br>
    <label for="personAge">Person Age:</label>
    <input type="text" id="personAge" name="personAge"><br><br>
    <input type="submit" value="Submit">
  </form>
```

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


Marriage Application Form
Person Name:
Person Age:
Submit

Person name is required.
OK

Experiment – 22

Aim :- Design Servlet Login and Logout using Cookies

Overview: Servlets can be used to implement login and logout functionalities by managing user sessions with cookies. When a user logs in, a cookie can be created to store a unique session identifier. This identifier can be used to track the user's session and maintain their logged-in state across requests. For logout, the cookie can be invalidated or expired to end the session. Servlets can retrieve the cookie information from the request to verify the user's session during subsequent requests. By using cookies, the login and logout processes become stateful, allowing for personalized user experiences and secure session management in web applications.

Code:

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>
<h1>Welcome to Login App by Cookie</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>
</body>
</html>
```

File: link.html

```
<a href="login.html">Login</a> |
<a href="LogoutServlet">Logout</a> |
<a href="ProfileServlet">Profile</a>
<hr>
```

File: login.html

```
<form action="LoginServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: LoginServlet.java

```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
```

```
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);
    String name=request.getParameter("name");
    String password=request.getParameter("password");
    if(password.equals("admin123")){
      out.print("You are successfully logged in!");
      out.print("<br>Welcome, "+name);
      Cookie ck=new Cookie("name",name);
      response.addCookie(ck);
    }else{
      out.print("sorry, username or password error!");
      request.getRequestDispatcher("login.html").include(request, response);
    }
    out.close();
  }
}
```

File: LogoutServlet.java

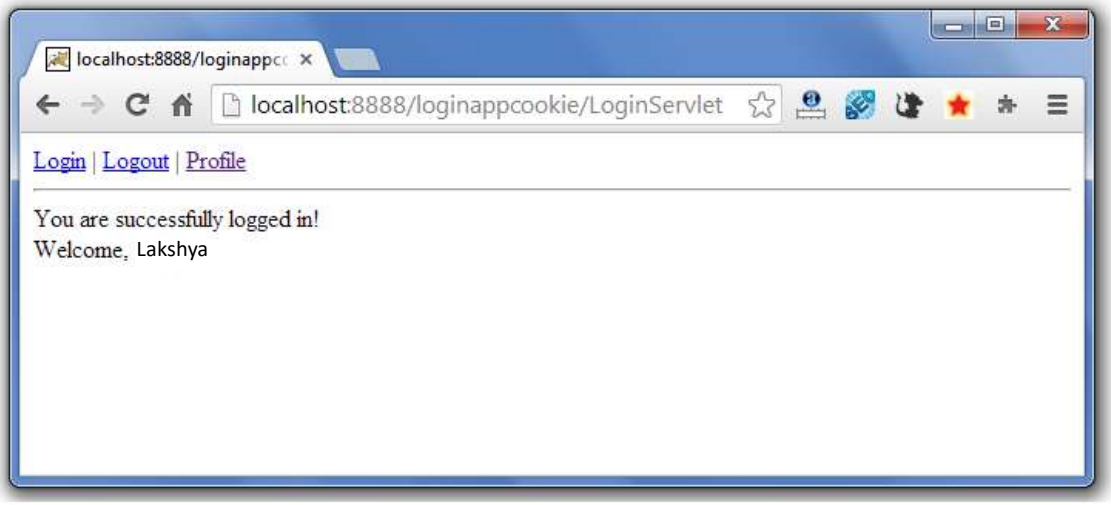
```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LogoutServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);
    Cookie ck=new Cookie("name","");
    ck.setMaxAge(0);
    response.addCookie(ck);
    out.print("you are successfully logged out!");
  }
}
```

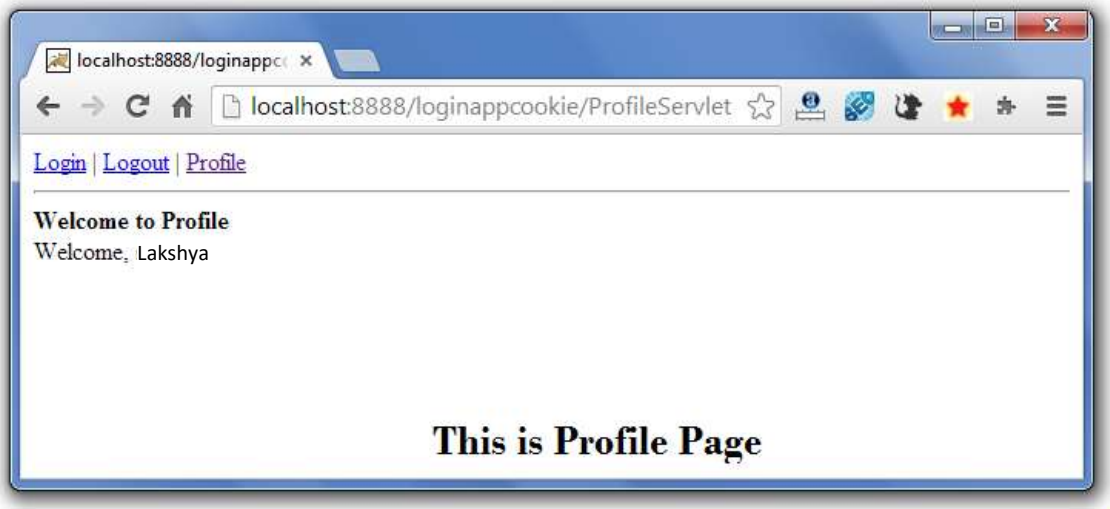
File: ProfileServlet.java

```
package com.javatpoint;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ProfileServlet extends HttpServlet {
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();

      request.getRequestDispatcher("link.html").include(request, response);

      Cookie ck[]=request.getCookies();
      if(ck!=null){
       String name=ck[0].getValue();
      if(!name.equals("")||name!=null){
         out.print("<b>Welcome to Profile</b>");
         out.print("<br>Welcome, "+name);
      }
      }else{
         out.print("Please login first");
         request.getRequestDispatcher("login.html").include(request, response);
      }
      out.close();
   }
}
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <servlet>
    <description></description>
    <display-name>LoginServlet</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.javatpoint.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>ProfileServlet</display-name>
    <servlet-name>ProfileServlet</servlet-name>
    <servlet-class>com.javatpoint.ProfileServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ProfileServlet</servlet-name>
    <url-pattern>/ProfileServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>LogoutServlet</display-name>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>com.javatpoint.LogoutServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
```

```
  <url-pattern>/LogoutServlet</url-pattern>
 </servlet-mapping>
</web-app>
```




localhost:8888/loginappc
localhost:8888/loginappcookie/ProfileServlet
Login | Logout | Profile
Welcome to Profile
Welcome, Lakshya
This is Profile Page


localhost:8888/loginappc
localhost:8888/loginappcookie/LogoutServlet
Login | Logout | Profile
you are successfully logged out!
This is logout or signout page

Experiment – 23

Aim :- Create a servlet that prints all the request headers it receives, along with their associated values.

Overview: A servlet can retrieve and display request headers using the `HttpServletRequest` object's `getHeaderNames()` method, which returns an enumeration of all the header names in the current request. By iterating over this enumeration and retrieving each header's value using the `getHeader()` method, the servlet can print each header and its associated value. This approach allows developers to inspect incoming HTTP headers, providing insights into the client's request and facilitating server-side processing tailored to the specific client or user agent interacting with the servlet.

Code:

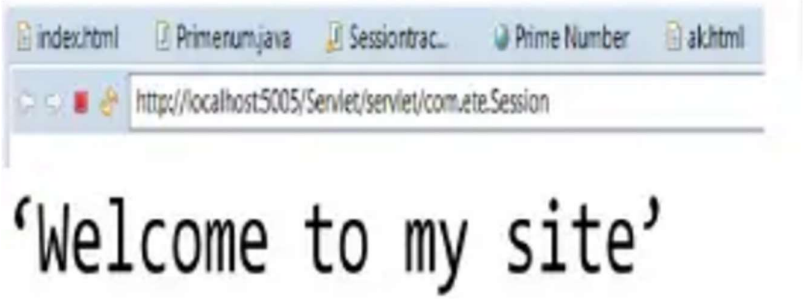
```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collections;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RequestHeadersServlet")
public class RequestHeadersServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><head><title>Request Headers</title></head><body>");
      out.println("<h2>Request Headers:</h2>");
      out.println("<ul>");
      Enumeration<String> headerNames = request.getHeaderNames();
      while (headerNames.hasMoreElements()) {
         String headerName = headerNames.nextElement();
         out.println("<li><strong>" + headerName + ":</strong> " +
request.getHeader(headerName) + "</li>");
      }
      out.println("</ul>");
      out.println("</body></html>");
   }
}
```

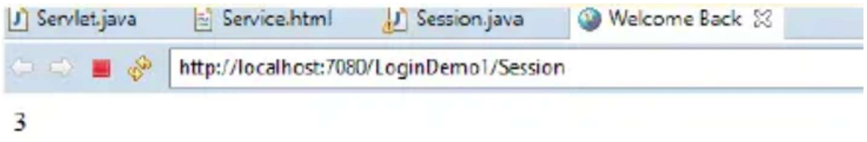
Experiment – 24

Aim :- 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".

Code:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/WelcomeServlet")
public class WelcomeServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      boolean isFirstTimeVisitor = true;
      Cookie[] cookies = request.getCookies();
      if (cookies != null) {
         for (Cookie cookie : cookies) {
            if ("visitedBefore".equals(cookie.getName())) {
               isFirstTimeVisitor = false;
               break;
            }
         }
      }
      Cookie visitedCookie = new Cookie("visitedBefore", "true");
      response.addCookie(visitedCookie);
      out.println("<html><head><title>Welcome</title></head><body>");
      if (isFirstTimeVisitor) {
         out.println("<h2>Welcome, you are visiting for the first time</h2>");
      } else {
         out.println("<h2>Welcome Back</h2>");
      }
      out.println("</body></html>");
   }
}
```





3

'WELCOME BACK'

Experiment – 25

Aim :- Create User Registration using Jsp, Servlet and Jdbc.

Code:

Step 1: Create database table for member

```
CREATE TABLE `member` (
`uname` varchar(45) NOT NULL,
`password` varchar(45) DEFAULT NULL,
`email` varchar(45) DEFAULT NULL,
`phone` varchar(45) DEFAULT NULL,
PRIMARY KEY (`uname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;</code>
```

Step 2: Create a memberRegister.jsp for the user registration

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1" %>
<!DOCTYPE html>
<html>
<head>
  <meta charset="ISO-8859-1">
  <title>Insert title here</title>
</head>
<body>
  <form action="Register" method="post">
    <table>
      <tr>
        <td>User Name</td>
        <td><input type="text" name="uname"></td>
      </tr>
      <tr>
        <td>Password</td>
        <td><input type="password" name="password"></td>
      </tr>
      <tr>
        <td>Email</td>
        <td><input type="text" name="email"></td>
      </tr>
      <tr>
        <td>Phone</td>
        <td><input type="text" name="phone"></td>
      </tr>
      <tr>
        <td>Submit</td>
        <td><input type="submit" value="register"></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

Step 3: Create a dto class Member.java

```
public class Member {
  private String uname, password, email, phone;
  public Member() {
    super();
  }
  public Member(String uname, String password, String email, String phone) {
    super();
    this.uname = uname;
    this.password = password;
    this.email = email;
    this.phone = phone;
  }
  public String getUname() {
    return uname;
  }
  public void setUname(String uname) {
    this.uname = uname;
  }
  public String getPassword() {
    return password;
  }
  public void setPassword(String password) {
    this.password = password;
  }
  public String getEmail() {
    return email;
  }
  public void setEmail(String email) {
    this.email = email;
  }
  public String getPhone() {
    return phone;
  }
  public void setPhone(String phone) {
    this.phone = phone;
  }
}
```

Step 4: Create a Servlet named Register.java

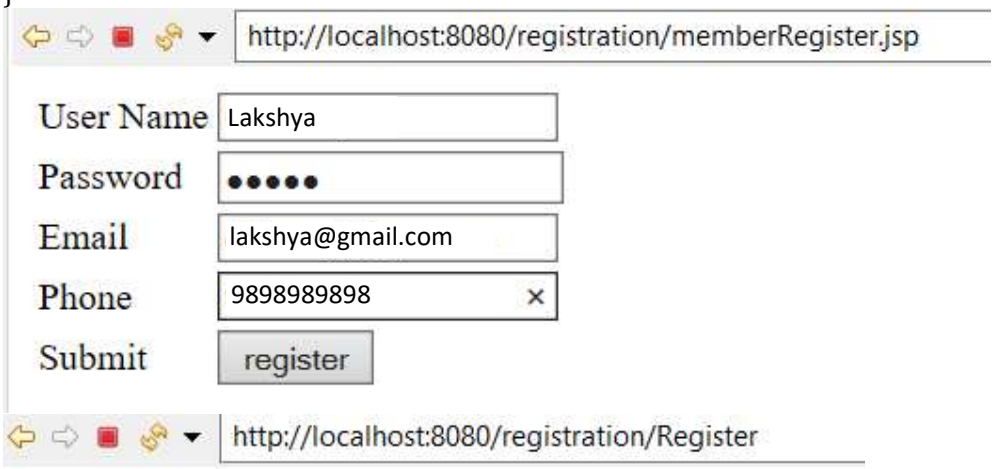
```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/Register")
public class Register extends HttpServlet {
  private static final long serialVersionUID = 1L;
  public Register() {
    super();
  }
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.getWriter().append("Served at: ").append(request.getContextPath());
  }
  protected void doPost(HttpServletRequest request,HttpServletResponse response) throws
ServletException, IOException {
    String uname = request.getParameter("uname");
    String password = request.getParameter("password");
    String email = request.getParameter("email");
    String phone = request.getParameter("phone");
    Member member = new Member(uname, password, email, phone);
    RegisterDao rdao = new RegisterDao();
    String result = rdao.insert(member);
    response.getWriter().println(result);
  }
}
```

Step 5: Create a Dao class RegisterDao.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class RegisterDao {
  private String dburl = "jdbc:mysql://localhost:3306/userdb";
  private String dbuname = "root";
  private String dbpassword = "mysql";
  private String dbdriver = "com.mysql.jdbc.Driver";
  public void loadDriver(String dbDriver) {
    try {
      Class.forName(dbDriver);
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
  public Connection getConnection() {
    Connection con = null;
    try {
      con = DriverManager.getConnection(dburl, dbuname, dbpassword);
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return con;
  }
  public String insert(Member member) {
    loadDriver(dbdriver);
    Connection con = getConnection();
    String sql = "insert into member values(?,?,?,?)";
    String result = "Data Entered Successfully";
    try {
      PreparedStatement ps = con.prepareStatement(sql);
      ps.setString(1, member.getUname());
      ps.setString(2, member.getPassword());
      ps.setString(3, member.getEmail());
      ps.setNString(4, member.getPhone());
```

```
    ps.executeUpdate();
   } catch (SQLException e) {
    result = "Data Not Entered Successfully";
    e.printStackTrace();
   }
   return result;
  }
}
```



Experiment – 26

Aim :- Create Employee Registration Form using a combination of JSP, Servlet, JDBC and MySQL Database

Overview:

Thread priorities in multithreading range from 1 to 10, with 1 being the lowest priority and 10 being the highest. Higher priority threads are scheduled for execution before lower priority threads. getPriority() retrieves a thread's priority, and setPriority() sets it. Use priorities judiciously, as they may not always have a significant impact and can lead to platform-specific behavior.

Code:

MySQL Database Setup

```
CREATE TABLE `employee` (
  `id` int(3) NOT NULL,
  `first_name` varchar(20) DEFAULT NULL,
  `last_name` varchar(20) DEFAULT NULL,
  `username` varchar(250) DEFAULT NULL,
  `password` varchar(20) DEFAULT NULL,
  `address` varchar(45) DEFAULT NULL,
  `contact` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Create a JavaBean - Employee.java:

```
import java.io.Serializable;
public class Employee implements Serializable {
   private static final long serialVersionUID = 1 L;
   private String firstName;
   private String lastName;
   private String username;
   private String password;
   private String address;
   private String contact;
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;  }
   public String getLastName() {
      return lastName; }
   public void setLastName(String lastName) {
      this.lastName = lastName; }
   public String getUsername() {
      return username;  }
   public void setUsername(String username) {
      this.username = username;  }
   public String getPassword() {
      return password;  }
```

```
   public void setPassword(String password) {
      this.password = password;  }
   public String getAddress() {
      return address;  }
   public void setAddress(String address) {
      this.address = address;  }
   public String getContact() {
      return contact; }
   public void setContact(String contact) {
      this.contact = contact;
 }
}
```

Create an EmployeeDao.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import net.javaguides.registration.model.Employee;
public class EmployeeDao {
   public int registerEmployee(Employee employee) throws ClassNotFoundException {
      String INSERT_USERS_SQL = "INSERT INTO employee" +
         "  (id, first_name, last_name, username, password, address, contact) VALUES " +
         " (?, ?, ?, ?, ?,?,?);";
      int result = 0;
      Class.forName("com.mysql.jdbc.Driver");
      try (Connection connection = DriverManager
         .getConnection("jdbc:mysql://localhost:3306/demo?useSSL=false", "root", "root");
         PreparedStatement preparedStatement =
connection.prepareStatement(INSERT_USERS_SQL)) {
         preparedStatement.setInt(1, 1);
         preparedStatement.setString(2, employee.getFirstName());
         preparedStatement.setString(3, employee.getLastName());
         preparedStatement.setString(4, employee.getUsername());
         preparedStatement.setString(5, employee.getPassword());
         preparedStatement.setString(6, employee.getAddress());
         preparedStatement.setString(7, employee.getContact());
         System.out.println(preparedStatement);
         result = preparedStatement.executeUpdate();
      } catch (SQLException e) {
         printSQLException(e);
      }
      return result; }
   private void printSQLException(SQLException ex) {
      for (Throwable e: ex) {
         if (e instanceof SQLException) {
            e.printStackTrace(System.err);
            System.err.println("SQLState: " + ((SQLException) e).getSQLState());
            System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
            System.err.println("Message: " + e.getMessage());
            Throwable t = ex.getCause();
            while (t != null) {
               System.out.println("Cause: " + t);
               t = t.getCause();
            } } } } }
```

Create an EmployeeServlet.java

```
package net.javaguides.employeemanagement.web;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.javaguides.employeemanagement.dao.EmployeeDao;
import net.javaguides.employeemanagement.model.Employee;
@WebServlet("/register")
public class EmployeeServlet extends HttpServlet {
   private static final long serialVersionUID = 1 L;
   private EmployeeDao employeeDao;
   public void init() {
      employeeDao = new EmployeeDao();  }
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      String firstName = request.getParameter("firstName");
      String lastName = request.getParameter("lastName");
      String username = request.getParameter("username");
      String password = request.getParameter("password");
      String address = request.getParameter("address");
      String contact = request.getParameter("contact");
      Employee employee = new Employee();
      employee.setFirstName(firstName);
      employee.setLastName(lastName);
      employee.setUsername(username);
      employee.setPassword(password);
      employee.setContact(contact);
      employee.setAddress(address);
      try {
         employeeDao.registerEmployee(employee);
      } catch (Exception e) {
         e.printStackTrace();  }
      response.sendRedirect("employeedetails.html"); } }
```

Create an employeeregister.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="ISO-8859-1">
   <title>Insert title here</title>
</head>
<body>
   <div align="center">
      <h1>Employee Register Form</h1>
      <form action="register" method="post">
         <table style="with: 80%">
            <tr> <td>First Name</td>
               <td><input type="text" name="firstName" /></td> </tr>
            <tr><td>Last Name</td>
               <td><input type="text" name="lastName" /></td></tr>
```

```
        <tr><td>UserName</td>
        <td><input type="text" name="username" /></td></tr>
        <tr><td>Password</td>
        <td><input type="password" name="password" /></td>   </tr>
        <tr><td>Address</td>
          <td><input type="text" name="address" /></td></tr>

        <tr> <td>Contact No</td>
          <td><input type="text" name="contact" /></td></tr>
      </table>
      <input type="submit" value="Submit" />
    </form>
  </div>
</body>
</html>
```

Create an employeedetail.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 <h1>Employee successfully registered !</h1>
</body>
</html>
```

Lakshya