GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Java Programming Lab

LAB MANUAL

SUBJECT CODE: BCS23209TP

Prepared by:-

RUBI KALITA

Laboratory Instructor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEING

GUWAHATI-781017

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

BCS23209L	JavaProgramming Lab	L	T	P	C
		0	0	2	1

Pre-requisite:ConceptofOOPusingC++

Course Objectives:

1. Applytheconceptsof Javato solve problems.
2. Understandtheprinciplesofinheritance andpolymorphism.
3. Implementationofpackages and interfaces.
4. Understandthe conceptsofexceptionhandlingandmultithreading.
5. ImplementationofthedesignofGraphicalUserInterfaceusingappletsandswing controls.
6. Understandtheconcepts ofJDBC,JSP.

CourseOutcomes: Aftersuccessfulcompletionofthecourse,thestudentswilllearn:

CO1:**Develop**Javaprogramsusingfundamentalandobject-orientedconceptssuchasclasses, objects, inheritance, and interfaces to solve real-world problems. (Cognitive Level: Creating)
CO2: **Apply** exception handling and multithreading techniques to create robust and efficient Java applications. (Cognitive Level: Applying)
CO3:**Design**graphicaluserinterfaces(GUI)withAWTandSwing,integratingevent-handling mechanisms and layout management. (Cognitive Level: Creating)
CO4:**Construct**database-drivenanddynamicweb-basedapplicationsusingJDBCandJSP. (Cognitive Level: Creating)

Module1:Basicsof JavaProgramming — **6 hours**

1. WriteaJavaprogramtodemonstratetheusageofdatatypes,variables,constants, and type casting.
2. Implementaprogramtoillustrateconditionalstatements(if,switch)andloops (for, while, do-while).
3. Developaprogramtoperformarrayoperationssuchasinsertion,deletion, and searching.
4. Create a Java program to define a class, create objects, and demonstrate theusage of constructors and this keyword.
5. Writeaprogramtodemonstratemethodoverloadingandconstructoroverloading.

Module2:Object-OrientedProgramming Concepts	**6 hours**
6. Implementsingle,multilevel andhierarchicalinheritanceinJava. 7. Writeaprogramtodemonstratetheusageofsuperandfinalkeywordsin inheritance. 8. Developaprogramtoillustratemethodoverriding. 9. Create a Java program to define interfaces, implement them, and access them through interface references.	

Module3:ExceptionHandlingand Multithreading	**6 hours**
11. Writeaprogramtohandlecheckedanduncheckedexceptionsusingtry, catch, throw, throws, and finally. 12. Developauser-definedexception classanduseitina program. 13. Createamultithreadedprogramtodemonstratethreadlifecycleandpriorities. 14. Implementthreadsynchronizationandinter-threadcommunication(e.g.,Producer-Consumer problem).	
Module4:GUIProgrammingandEventHandling	**6 hours**
15. CreateaJavaapplettodemonstratetheappletlifecycleandparameterpassing. 16. DesignaGUIapplicationusingAWTtohandlebasiceventslikebutton clicks. 17. DevelopaSwing-based applicationtodemonstratetheuseofJButton,JLabel, JTextField, and JTextArea. 18. Implement layout management using BorderLayout, GridLayout, and FlowLayout. 19. Writeaprogramto handlemouseand keyboard eventsusingadapter classes.	
Module5:DatabaseandWebProgramming	**6 hours**
20. WriteaJavaprogram to connectto adatabaseusingJDBC. 21. DevelopaprogramtousedifferenttypesofJDBCstatements(Statement, PreparedStatement, CallableStatement). 22. CreateasimpledynamicwebapplicationusingJSPtodisplay andprocess user inputs.	
TotalLecturehours	**30 hours**

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Module 1: Basics of Java Programming

Experiment 1.

Write a Java program to demonstrate the usage of data types, variables, constants, and type casting.

Program:

```
public class DataTypesDemo {

    public static void main(String[] args) {
        // Primitive data types
        int integerVar = 10;
        double doubleVar = 20.5;
        char charVar = 'A';
        boolean booleanVar = true;
        long longVar = 1234567890123L;
        float floatVar = 3.14f;
        short shortVar = 100;
        byte byteVar = 127;


        // Constants
        final int CONSTANT_VALUE = 100;

        // Type casting
        // Implicit casting (widening)
        double intToDouble = integerVar;

        // Explicit casting (narrowing)
        int doubleToInt = (int) doubleVar;
```

```
        // Printing values
        System.out.println("Integer: " + integerVar);
        System.out.println("Double: " + doubleVar);
        System.out.println("Character: " + charVar);
        System.out.println("Boolean: " + booleanVar);
        System.out.println("Long: " + longVar);
        System.out.println("Float: " + floatVar);
        System.out.println("Short: " + shortVar);
        System.out.println("Byte: " + byteVar);
        System.out.println("Constant: " + CONSTANT_VALUE);
        System.out.println("Integer to Double (implicit): " + intToDouble);
        System.out.println("Double to Integer (explicit): " + doubleToInt);

        // String example
        String message = "Hello, Java!";
        System.out.println(message);
    }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 2.

Implement a program to illustrate conditional statements (if, switch) and loops (for, while, do-while).

Program:

```
#include <stdio.h>

int main() {
   int num = 7;

   // If-else statement
   if (num > 0) {
      printf("%d is positive.\n", num);
   } else if (num < 0) {
      printf("%d is negative.\n", num);
   } else {
      printf("%d is zero.\n", num);
   }

   // Switch statement
   switch (num) {
      case 1:
      case 2:
      case 3:
         printf("%d is a small number.\n", num);
         break;
      case 4:
      case 5:
      case 6:
         printf("%d is a medium number.\n", num);
         break;
      default:
         printf("%d is a large number or not in range 1-6.\n", num);
         break;
```

```
    }

    // For loop
    printf("For loop:\n");
    for (int i = 1; i <= 5; i++) {
        printf("%d ", i);
    }
    printf("\n");

    // While loop
    printf("While loop:\n");
    int i = 1;
    while (i <= 5) {
        printf("%d ", i);
        i++;
    }
    printf("\n");

    // Do-while loop
    printf("Do-while loop:\n");
    i = 1;
    do {
        printf("%d ", i);
        i++;
    } while (i <= 5);
    printf("\n");

    return 0;
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 3.

Develop a program to perform array operations such as insertion, deletion, and searching.

Program:

```
public class ArrayOperations {

    public static void main(String[] args) {
        int[] array = new int[10]; // Initializing an array of size 10
        int size = 0; // Current size of the array

        // Insertion
        size = insertElement(array, size, 10, 0);
        size = insertElement(array, size, 20, 1);
        size = insertElement(array, size, 30, 2);
        size = insertElement(array, size, 40, 3);
        size = insertElement(array, size, 50, 4);

        System.out.print("Array after insertion: ");
        printArray(array, size);

        // Search
        int index = searchElement(array, size, 30);
        if (index != -1) {
            System.out.println("Element 30 found at index: " + index);
        } else {
            System.out.println("Element 30 not found in the array");
        }

        // Deletion
        size = deleteElement(array, size, 2);
         System.out.print("Array after deletion: ");
        printArray(array, size);
    }

    // Insertion operation
    public static int insertElement(int[] array, int size, int element, int position) {
        if (position < 0 || position > size || size >= array.length) {
            System.out.println("Insertion is not possible.");
            return size;
        }
```

```
        for (int i = size; i > position; i--) {
            array[i] = array[i - 1];
        }
        array[position] = element;
        return size + 1;
    }

    // Deletion operation
    public static int deleteElement(int[] array, int size, int position) {
        if (position < 0 || position >= size) {
            System.out.println("Deletion is not possible.");
            return size;
        }
        for (int i = position; i < size - 1; i++) {
            array[i] = array[i + 1];
        }
        return size - 1;
    }

    // Search operation (Linear search)
    public static int searchElement(int[] array, int size, int element) {
        for (int i = 0; i < size; i++) {
            if (array[i] == element) {
                return i; // Return the index if element is found
            }
        }
        return -1; // Return -1 if element is not found
    }

    // Utility method to print the array
    public static void printArray(int[] array, int size) {
        for (int i = 0; i < size; i++) {
            System.out.print(array[i] + " ");
        }
        System.out.println();
    }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 4.

Create a Java program to define a class, create objects, and demonstrate the usage of constructors and this keyword.

Program:

```
public class Student {
    private String name;
    private int age;
    private String studentId;

    // Default constructor
    public Student() {
        this("Unknown", 18, "N/A");
    }

    // Parameterized constructor
    public Student(String name, int age, String studentId) {
        this.name = name;
        this.age = age;
        this.studentId = studentId;
    }
    // Method to display student information
    public void displayInfo() {
        System.out.println("Name: " + this.name);
        System.out.println("Age: " + this.age);
        System.out.println("Student ID: " + this.studentId);
    }
    public static void main(String[] args) {
        // Creating objects using different constructors
        Student student1 = new Student();
        Student student2 = new Student("Alice Smith", 20, "S12345");

        // Displaying student information
        System.out.println("Student 1 Information:");
        student1.displayInfo();
```

```
        System.out.println("\nStudent 2 Information:");
        student2.displayInfo();
    }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 5.

Write a program to demonstrate method overloading and constructor overloading.

Program:

```
class Box {
   double width, height, depth;

   // constructor used when all dimensions
   // specified
   Box(double w, double h, double d)
   {
      width = w;
      height = h;
      depth = d;
   }
   // constructor used when no dimensions
   // specified
   Box() { width = height = depth = 0; }

   // constructor used when cube is created
   Box(double len) { width = height = depth = len; }

   // compute and return volume
   double volume() { return width * height * depth; }
```

```
}

// Driver code
public class Test {
    public static void main(String args[])
    {
        // create boxes using the various
        // constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;

        // get volume of first box
        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);

        // get volume of second box
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        // get volume of cube
        vol = mycube.volume();
```

```
        System.out.println("Volume of mycube is " + vol);
    }
}
```

Module 2: Object-Oriented Programming Concepts

Experiment 6.

Implement single, multilevel and hierarchical inheritance in Java.

Program:

```
/ Single Inheritance
class Animal {
    String name;
    void eat() {
        System.out.println("Animal is eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking");
    }
}

// Multilevel Inheritance
class Vehicle {
    String model;
}

class Car extends Vehicle {
    String color;
}

class ElectricCar extends Car {
    int batteryCapacity;
}

// Hierarchical Inheritance
class Shape {
    void draw() {
        System.out.println("Drawing a shape");
    }
}
```

```
class Circle extends Shape {
   void drawCircle() {
      System.out.println("Drawing a circle");
   }
}

class Square extends Shape {
   void drawSquare() {
      System.out.println("Drawing a square");
   }
}

public class Main {
   public static void main(String[] args) {
      // Single Inheritance
      Dog myDog = new Dog();
      myDog.name = "Buddy";
      myDog.eat();
      myDog.bark();

      // Multilevel Inheritance
      ElectricCar myElectricCar = new ElectricCar();
      myElectricCar.model = "Tesla Model 3";
      myElectricCar.color = "Red";
      myElectricCar.batteryCapacity = 75;

      // Hierarchical Inheritance
      Circle myCircle = new Circle();
      myCircle.draw();
      myCircle.drawCircle();

      Square mySquare = new Square();
      mySquare.draw();
      mySquare.drawSquare();
   }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 7.

Write a program to demonstrate the usage of super and final keywords in inheritance.

Program:

```
class Animal {
   String name;

   Animal(String name) {
      this.name = name;
   }

   void eat() {
      System.out.println(name + " is eating.");
   }

   final void makeSound() {
      System.out.println("Generic animal sound.");
   }
}

class Dog extends Animal {
   String breed;

   Dog(String name, String breed) {
      super(name); // Calling the constructor of the superclass (Animal)
      this.breed = breed;
   }

   @Override
   void eat() {
      super.eat(); // Calling the eat() method of the superclass
      System.out.println(name + " is eating bone.");
   }

   //Attempting to override final method will cause compilation error
   // void makeSound() {
   //     System.out.println("Woof!");
   // }
}
```

```
final class Cat extends Animal {
    Cat(String name) {
        super(name);
    }

    @Override
    void eat() {
        System.out.println(name + " is eating fish.");
    }
}

// Attempting to inherit from final class will cause compilation error
// class Persian extends Cat {
// }

public class InheritanceDemo {
    public static void main(String[] args) {
        Dog dog = new Dog("Buddy", "Golden Retriever");
        dog.eat();
        dog.makeSound();

        Cat cat = new Cat("Whiskers");
        cat.eat();
        cat.makeSound();
    }
}



class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }

    void eat() {
        System.out.println(name + " is eating.");
    }

    final void makeSound() {
```

```
        System.out.println("Generic animal sound.");
    }
}

class Dog extends Animal {
    String breed;

    Dog(String name, String breed) {
        super(name); // Calling the constructor of the superclass (Animal)
        this.breed = breed;
    }

    @Override
    void eat() {
        super.eat(); // Calling the eat() method of the superclass
        System.out.println(name + " is eating bone.");
    }

    //Attempting to override final method will cause compilation error
    // void makeSound() {
    //     System.out.println("Woof!");
    // }
}

final class Cat extends Animal {
    Cat(String name) {
        super(name);
    }

    @Override
    void eat() {
        System.out.println(name + " is eating fish.");
    }
}

// Attempting to inherit from final class will cause compilation error
// class Persian extends Cat {
// }

public class InheritanceDemo {
    public static void main(String[] args) {
        Dog dog = new Dog("Buddy", "Golden Retriever");
```

```
        dog.eat();
        dog.makeSound();

        Cat cat = new Cat("Whiskers");
        cat.eat();
        cat.makeSound();
    }
}
class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }

    void eat() {
        System.out.println(name + " is eating.");
    }

    final void makeSound() {
        System.out.println("Generic animal sound.");
    }
}

class Dog extends Animal {
    String breed;

    Dog(String name, String breed) {
        super(name); // Calling the constructor of the superclass (Animal)
        this.breed = breed;
    }

    @Override
    void eat() {
        super.eat(); // Calling the eat() method of the superclass
        System.out.println(name + " is eating bone.");
    }

    //Attempting to override final method will cause compilation error
    // void makeSound() {
    //     System.out.println("Woof!");
    // }
```

```
}

final class Cat extends Animal {
    Cat(String name) {
        super(name);
    }

    @Override
    void eat() {
        System.out.println(name + " is eating fish.");
    }
}

// Attempting to inherit from final class will cause compilation error
// class Persian extends Cat {
// }

public class InheritanceDemo {
    public static void main(String[] args) {
        Dog dog = new Dog("Buddy", "Golden Retriever");
        dog.eat();
        dog.makeSound();

        Cat cat = new Cat("Whiskers");
        cat.eat();
        cat.makeSound();
    }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 8. Develop a program to illustrate method overriding.

Program:

```
class Animal {
    public void makeSound() {
        System.out.println("Generic animal sound");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow!");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Dog myDog = new Dog();
        Cat myCat = new Cat();

        myAnimal.makeSound(); // Output: Generic animal sound
        myDog.makeSound();    // Output: Woof!
        myCat.makeSound();    // Output: Meow!
    }
}
```

Experiment 9.

Create a Java program to define interfaces, implement them, and access them through interface references.

Program:

```
// Define an interface
interface Shape {
    // Abstract methods (no implementation)
    double getArea();
    double getPerimeter();
}

// Implement the Shape interface in the Circle class
class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
}

// Implement the Shape interface in the Rectangle class
class Rectangle implements Shape {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
```

```
    @Override
    public double getArea() {
        return length * width;
    }

    @Override
    public double getPerimeter() {
        return 2 * (length + width);
    }
}

// Main class to test the implementation
public class Main {
    public static void main(String[] args) {
        // Create objects of the classes
        Circle circle = new Circle(5);
        Rectangle rectangle = new Rectangle(4, 6);

        // Access objects through interface references
        Shape shape1 = circle;
        Shape shape2 = rectangle;

        // Call methods using interface references
        System.out.println("Circle Area: " + shape1.getArea());
        System.out.println("Circle Perimeter: " + shape1.getPerimeter());

        System.out.println("Rectangle Area: " + shape2.getArea());
        System.out.println("Rectangle Perimeter: " + shape2.getPerimeter());
    }
}
```

Experiment 10.

Demonstrate the creation and usage of packages and importing user-defined packages.

Program:

In this example, we'll create a user-defined package and function to print a message for the users.

```
// Java program to create a user-defined
// package and function to print
// a message for the users
package example;

// Class
public class A {

   public void show()
   {
      System.out.println("Hello!! How are you?");
   }

   public static void main(String args[])
   {
      A obj = new A();
      obj.show();
   }
}
```

In the below-mentioned example, we'll import the user-defined package "example" created in the above example. And use the function to print messages.

```
import example A;

public class GFG {
   public static void main(String args[])
   {
      A obj = new A();
      obj.show();
   }
}
```

Module 3: Exception Handling and Multithreading

Experiment 11:

Write a program to handle checked and unchecked exceptions using try, catch, throw, throws, and finally.

Program:

```
import java.io.IOException;

public class ExceptionHandling {

    // Method that declares it might throw a checked exception
    public void checkedExceptionMethod() throws IOException {
        // Simulate a checked exception (e.g., file not found)
        if (true) {
            throw new IOException("File not found");
        }
    }

    // Method that might throw an unchecked exception
    public void uncheckedExceptionMethod(int num) {
        // Simulate an unchecked exception (e.g., division by zero)
        int result = 10 / num;
        System.out.println("Result: " + result);
    }

    public static void main(String[] args) {
        ExceptionHandling eh = new ExceptionHandling();

        // Handling checked exception using try-catch
        try {
            eh.checkedExceptionMethod();
        } catch (IOException e) {
            System.err.println("Caught checked exception: " + e.getMessage());
        }
```

```
        // Handling unchecked exception using try-catch
        try {
            eh.uncheckedExceptionMethod(0); // Will cause ArithmeticException
        } catch (ArithmeticException e) {
            System.err.println("Caught unchecked exception: " + e.getMessage());
        }

        // Using finally block
        try {
            int[] arr = new int[5];
            System.out.println(arr[10]); // This will throw ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Caught exception: " + e.getMessage());
        } finally {
            System.out.println("Finally block executed");
        }

        // Throwing a custom exception
        try {
            validateAge(15);
        } catch (InvalidAgeException e) {
            System.err.println("Caught custom exception: " + e.getMessage());
        }
    }

    // Method to demonstrate "throws" keyword
    public static void validateAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or older.");
        }
        System.out.println("Age is valid.");
    }
}

// Custom exception class
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
```

```
        super(message);
    }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 12:

Develop a user-defined exception class and use it in a program.

Program:

```
public class Main {
   static void validateAge(int age) throws InvalidAgeException {
      if (age < 0) {
         throw new InvalidAgeException("Age cannot be negative");
      } else {
         System.out.println("Age is valid: " + age);
      }
   }

   public static void main(String[] args) {
      try {
         validateAge(-5);
      } catch (InvalidAgeException e) {
         System.out.println("Caught exception: " + e.getMessage());
      }

      try {
         validateAge(30);
      } catch (InvalidAgeException e) {
         System.out.println("Caught exception: " + e.getMessage());
      }
   }
}
```

Experiment 13:

Create a multithreaded program to demonstrate thread life cycle and priorities.

Program:

```
class MyThread extends Thread {
    public MyThread(String name) {
        super(name);
    }

    @Override
    public void run() {
        System.out.println(getName() + " - State: " + getState());
        for (int i = 0; i < 5; i++) {
            System.out.println(getName() + " - Count: " + i);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                System.out.println(getName() + " interrupted.");
            }
        }
        System.out.println(getName() + " - State: " + getState());
    }
}

public class ThreadDemo {
    public static void main(String[] args) throws InterruptedException {
        MyThread thread1 = new MyThread("Thread-1");
        MyThread thread2 = new MyThread("Thread-2");
        MyThread thread3 = new MyThread("Thread-3");

        thread1.setPriority(Thread.MIN_PRIORITY);
        thread2.setPriority(Thread.NORM_PRIORITY);
        thread3.setPriority(Thread.MAX_PRIORITY);

        System.out.println(thread1.getName() + " - State: " + thread1.getState() + " - Priority: " +
thread1.getPriority());
```

```
        System.out.println(thread2.getName() + " - State: " + thread2.getState() + " - Priority: " +
thread2.getPriority());
        System.out.println(thread3.getName() + " - State: " + thread3.getState() + " - Priority: " +
thread3.getPriority());

        thread1.start();
        thread2.start();
        thread3.start();

        thread1.join();
        thread2.join();
        thread3.join();

        System.out.println(thread1.getName() + " - State: " + thread1.getState());
        System.out.println(thread2.getName() + " - State: " + thread2.getState());
        System.out.println(thread3.getName() + " - State: " + thread3.getState());
    }
}
```

Experiment 14:

Implement thread synchronization and inter-thread communication (e.g., Producer-Consumer problem).

Program:

```
import java.util.LinkedList;
import java.util.Queue;

class SharedBuffer {
    private Queue<Integer> buffer;
    private int capacity;

    public SharedBuffer(int capacity) {
        this.buffer = new LinkedList<>();
        this.capacity = capacity;
    }

    public synchronized void produce(int item) throws InterruptedException {
        while (buffer.size() == capacity) {
            wait(); // Wait if buffer is full
        }
        buffer.add(item);
        System.out.println("Produced: " + item);
        notifyAll(); // Notify consumers
    }

    public synchronized int consume() throws InterruptedException {
        while (buffer.isEmpty()) {
            wait(); // Wait if buffer is empty
        }
        int item = buffer.remove();
        System.out.println("Consumed: " + item);
        notifyAll(); // Notify producers
        return item;
    }
}
```

```
class Producer implements Runnable {
    private SharedBuffer buffer;
    private int id;

    public Producer(SharedBuffer buffer, int id) {
        this.buffer = buffer;
        this.id = id;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            try {
                buffer.produce(id * 10 + i);
                Thread.sleep(100); // Simulate production time
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Consumer implements Runnable {
    private SharedBuffer buffer;
    private int id;

    public Consumer(SharedBuffer buffer, int id) {
        this.buffer = buffer;
        thisentre id = id;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            try {
                buffer.consume();
```

```
            Thread.sleep(200); // Simulate consumption time
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
  }
}

public class ProducerConsumer {
  public static void main(String[] args) {
    SharedBuffer buffer = new SharedBuffer(3);

    Thread producer1 = new Thread(new Producer(buffer, 1));
    Thread producer2 = new Thread(new Producer(buffer, 2));
    Thread consumer1 = new Thread(new Consumer(buffer, 1));
    Thread consumer2 = new Thread(new Consumer(buffer, 2));

    producer1.start();
    producer2.start();
    consumer1.start();
    consumer2.start();
  }
}
```

Module 4: GUI Programming and Event Handling

Experiment 15:

Create a Java applet to demonstrate the applet life cycle and parameter passing.

Program:

```
import java.applet.Applet;
import java.awt.Graphics;

/*
<applet code="AppletLifeCycleDemo" width="300" height="150">
    <param name="message" value="Hello from HTML!" />
</applet>
*/
public class AppletLifeCycleDemo extends Applet {
    private String message;

    // Called when the applet is first loaded
    public void init() {
        System.out.println("init() method called: Applet is initializing");
        message = getParameter("message");
        if (message == null) {
            message = "Default message";
        }
    }

    // Called after init() and when the applet becomes active
    public void start() {
        System.out.println("start() method called: Applet is starting");
    }

    // Called when the applet is being stopped (e.g., when the user navigates away)
    public void stop() {
        System.out.println("stop() method called: Applet is stopping");
    }

    // Called before the applet is destroyed (e.g., when the browser is closed)
```

```
public void destroy() {
   System.out.println("destroy() method called: Applet is being destroyed");
}

// Called to paint the applet's content
public void paint(Graphics g) {
   System.out.println("paint() method called: Applet is painting");
   g.drawString(message, 50, 50);
}
```

Experiment 16:

Design a GUI application using AWT to handle basic events like button clicks.

Program:

```
import java.awt.*;
import java.awt.event.*;

public class ButtonClickExample extends Frame {
    private Button button;
    private Label label;

    public ButtonClickExample() {
        setTitle("Button Click Example");
        setLayout(new FlowLayout());

        // Create the button
        button = new Button("Click Me");
        add(button);

        // Create a label to display messages
        label = new Label("Click the button!");
        add(label);

        // Add an action listener to the button
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // This method will be called when the button is clicked
                label.setText("Button Clicked!");
            }
        };

        setSize(300, 150);
        setVisible(true);
    }
```

```
    public static void main(String[] args) {
        new ButtonClickExample();
    }
}
```

Experiment 17:

Develop a Swing-based application to demonstrate the use of JButton, JLabel, JTextField, and JTextArea.

Program:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SwingDemo extends JFrame implements ActionListener {

    private JLabel labelName;
    private JTextField textName;
    private JButton buttonSubmit;
    private JTextArea textAreaResult;

    public SwingDemo() {
        setTitle("Swing Demo Application");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setLayout(new FlowLayout());

        labelName = new JLabel("Enter your name:");
        textName = new JTextField(20);
        buttonSubmit = new JButton("Submit");
        textAreaResult = new JTextArea(10, 30);
        textAreaResult.setEditable(false);

        buttonSubmit.addActionListener(this);

        add(labelName);
        add(textName);
        add(buttonSubmit);
        add(new JScrollPane(textAreaResult));

        setVisible(true);
    }

    @Override
```

```
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == buttonSubmit) {
            String name = textName.getText();
            textAreaResult.append("Hello, " + name + "!\n");
            textName.setText("");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(SwingDemo::new);
    }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 18:

Implement layout management using BorderLayout, GridLayout, and FlowLayout.

Program:

```
import javax.swing.*;
import java.awt.*;

public class LayoutExample extends JFrame {

    public LayoutExample() {
        setTitle("Layout Example");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 300);
        setLocationRelativeTo(null); // Center the frame

        JPanel mainPanel = new JPanel(new BorderLayout());

        // BorderLayout example
        JPanel borderPanel = new JPanel(new BorderLayout());
        borderPanel.add(new JButton("North"), BorderLayout.NORTH);
        borderPanel.add(new JButton("South"), BorderLayout.SOUTH);
        borderPanel.add(new JButton("East"), BorderLayout.EAST);
        borderPanel.add(new JButton("West"), BorderLayout.WEST);
        borderPanel.add(new JButton("Center"), BorderLayout.CENTER);
        mainPanel.add(borderPanel, BorderLayout.NORTH);

        // GridLayout example
        JPanel gridPanel = new JPanel(new GridLayout(2, 3, 5, 5)); // 2 rows, 3 columns, 5px gaps
        gridPanel.add(new JButton("1"));
        gridPanel.add(new JButton("2"));
        gridPanel.add(new JButton("3"));
        gridPanel.add(new JButton("4"));
        gridPanel.add(new JButton("5"));
        gridPanel.add(new JButton("6"));
```

```
        mainPanel.add(gridPanel, BorderLayout.CENTER);

        // FlowLayout example
        JPanel flowPanel = new JPanel(new FlowLayout());
        flowPanel.add(new JButton("Button 1"));
        flowPanel.add(new JButton("Button 2"));
        flowPanel.add(new JButton("Button 3"));
         mainPanel.add(flowPanel, BorderLayout.SOUTH);

        add(mainPanel);
        setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(LayoutExample::new);
    }
}
```

Experiment 19:

Write a program to handle mouse and keyboard events using adapter classes.

Program:

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.BorderLayout;

public class EventHandlingWithAdapters {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Handling with Adapters");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        JPanel panel = new JPanel();
        JLabel mouseLabel = new JLabel("Mouse Events: ");
        JLabel keyLabel = new JLabel("Key Events: ");

        panel.add(mouseLabel);
        panel.add(keyLabel);

        // MouseAdapter to handle mouse events
        panel.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                mouseLabel.setText("Mouse Clicked at: " + e.getX() + ", " + e.getY());
            }
        });

        // KeyAdapter to handle key events
        frame.addKeyListener(new KeyAdapter() {
            @Override
```

```
        public void keyPressed(KeyEvent e) {
            keyLabel.setText("Key Pressed: " + KeyEvent.getKeyText(e.getKeyCode()));
        }
    });

    frame.add(panel, BorderLayout.CENTER);
    frame.setVisible(true);
    frame.setFocusable(true); // Make sure the frame is focusable for key events
    frame.requestFocusInWindow(); // Request focus so key events are received
  }
}
```

Module 5: Database and Web Programming

Experiment 20:

Write a Java program to connect to a database using JDBC.

Program:

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

public class DatabaseConnection {

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/your_database_name";
        String user = "your_username";
        String password = "your_password";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("MySQL JDBC Driver not found.");
            return;
        }

        try (Connection connection = DriverManager.getConnection(url, user, password)) {
            if (connection != null) {
                System.out.println("Connected to the database!");
            } else {
                System.out.println("Failed to connect to the database.");
            }
        } catch (SQLException e) {
            System.err.println("Database connection error: " + e.getMessage());
        }
    }
}
```

Experiment 21:

Develop a program to use different types of JDBC statements (Statement, PreparedStatement, CallableStatement).

Program:

```
import java.sql.*;

public class JDBCStatements {

    static final String DB_URL = "jdbc:mysql://localhost/your_database";
    static final String USER = "your_username";
    static final String PASS = "your_password";

    public static void main(String[] args) {
        try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASS)) {

            // Statement Example
            System.out.println("Creating statement...");
            try (Statement stmt = conn.createStatement()) {
                String sql = "SELECT id, name FROM employees";
                ResultSet rs = stmt.executeQuery(sql);
                System.out.println("Statement Results:");
                while (rs.next()) {
                    System.out.println("ID: " + rs.getInt("id") + ", Name: " + rs.getString("name"));
                }
            }

            // PreparedStatement Example
            System.out.println("\nCreating prepared statement...");
            String insertSql = "INSERT INTO employees (name) VALUES (?)";
            try (PreparedStatement pstmt = conn.prepareStatement(insertSql)) {
                pstmt.setString(1, "John Doe");
                int affectedRows = pstmt.executeUpdate();
                System.out.println("PreparedStatement: " + affectedRows + " row(s) inserted.");
            }
```

```
        // CallableStatement Example
        System.out.println("\nCalling stored procedure...");
        String callSql = "{call get_employee_count(?)}";
        try (CallableStatement cstmt = conn.prepareCall(callSql)) {
            cstmt.registerOutParameter(1, Types.INTEGER);
            cstmt.execute();
            int employeeCount = cstmt.getInt(1);
             System.out.println("CallableStatement: Total employees - " + employeeCount);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Experiment 22:

Create a simple dynamic web application using JSP to display and process user inputs.

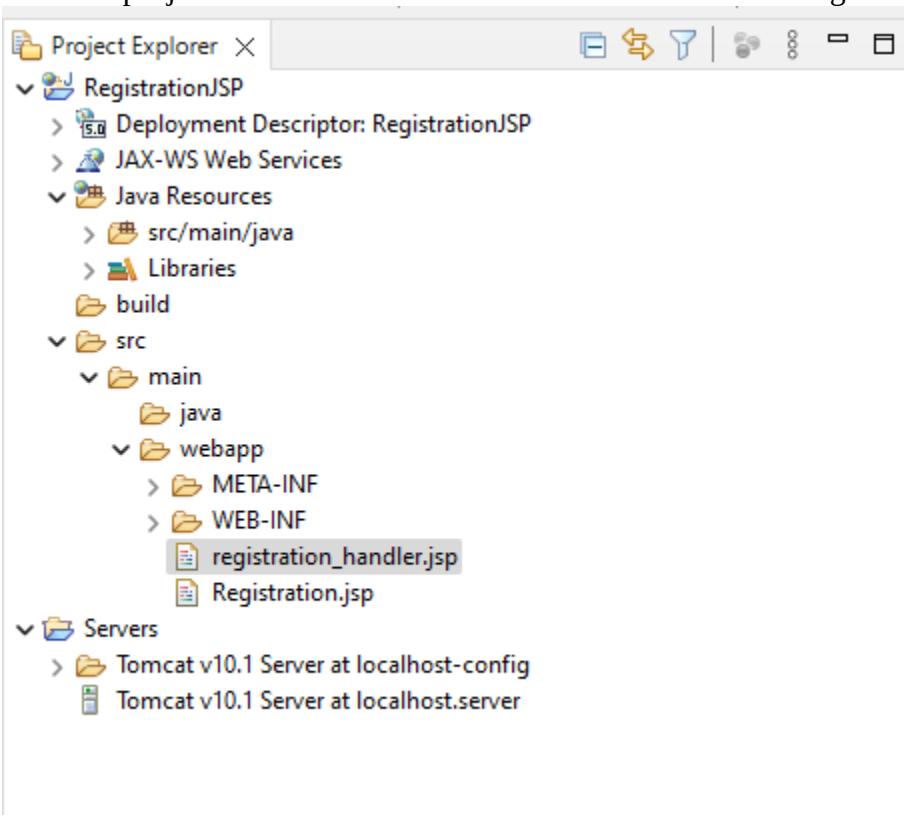
Program:

Steps to Generate Registration Form in JSP

We can develop the simple registration JSP form with basic details of the user after then processing from the server then response back display the data to the client as the result.

Step 1: Create the dynamic web project using Eclipse and it is named as the **RegistrationJSP** Open the project into the eclipse.

Once the project then the file structure looks like the below image.



Step 2: Create the new JSP file and it named as the **Registration.jsp** and this file can be used to create the registration form to gather data from the user.

Go to **src > webapp > Registration.jsp** and put the below code.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Registration Form</title>
   <style>
      /* CSS styles for the registration form container */
      .registration-container {
         width: 600px; /* Set the width of the container */
         margin: 0 auto; /* Center the container horizontally */
         padding: 20px; /* Add some padding inside the container */
         border: 2px solid #ccc; /* Add a border */
         border-radius: 10px; /* Add some border radius */
         background-color: #f9f9f9; /* Background color */
      }

      /* CSS styles for the heading */
      .registration-heading {
         text-align: center; /* Center the heading text */
      }

      /* CSS styles for input fields */
      .input-field {
         width: 100%;
         padding: 8px;
         margin: 5px 0;
         box-sizing: border-box;
      }
   </style>
</head>
<body>

<div class="registration-container">
   <h2 class="registration-heading">Registration Form</h2>
   <form action="registration_handler.jsp" method="post">
```

```
        <label for="firstname">First Name:</label>
        <input type="text" id="firstname" name="firstname" class="input-field" required><br>

        <label for="lastname">Last Name:</label>
        <input type="text" id="lastname" name="lastname" class="input-field" required><br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" class="input-field" required><br>

        <label for="mobile">Mobile Number:</label>
        <input type="tel" id="mobile" name="mobile" class="input-field" required><br>

        <label for="address">Address:</label>
        <textarea id="address" name="address" class="input-field" required></textarea><br>

        <label for="branch">Branch:</label>
        <input type="text" id="branch" name="branch" class="input-field" required><br>

        <label for="year">Year:</label>
        <input type="text" id="year" name="year" class="input-field" required><br>

        <label for="percentage">Percentage:</label>
        <input type="text" id="percentage" name="percentage" class="input-field" required><br>

        <input type="submit" value="Register">
    </form>
</div>

</body>
</html>
```

Step 3: Create the new JSP file and it named as the **registration_handler.jsp** and this file can be used to processing the user data from the server and response back to the client.

Go to **src > webapp > registration_handler.jsp** and put the below code.

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

```
    <meta charset="UTF-8">
    <title>Registration Handler</title>
    <style>
        /* CSS styles for the registration form container */
        .registration-container {

            width: 600px; /* Set the width of the container */
            margin: 20px auto; /* Center the container horizontally with some top margin */
            padding: 20px; /* Add some padding inside the container */
            border: 2px solid #ccc; /* Add a border */
            border-radius: 10px; /* Add some border radius */
            background-color: #f9f9f9; /* Background color */
        }

        /* CSS styles for the heading */
        .registration-heading {
            text-align: center; /* Center the heading text */
        }

        /* CSS styles for input fields */
        .input-field {
            width: 100%;
            padding: 8px;
            margin: 5px 0;
            box-sizing: border-box;
        }
    </style>
</head>
<body>

<div class="registration-container">
    <h2 class="registration-heading">Registration Details</h2>
    <%
        // Retrieve form data
        String firstName = request.getParameter("firstname");
        String lastName = request.getParameter("lastname");
        String email = request.getParameter("email");
        String mobile = request.getParameter("mobile");
        String address = request.getParameter("address");
        String branch = request.getParameter("branch");
        String year = request.getParameter("year");
        String percentage = request.getParameter("percentage");
```

```
   %>
   <p><strong>First Name:</strong> <%= firstName %></p>
   <p><strong>Last Name:</strong> <%= lastName %></p>
   <p><strong>Email:</strong> <%= email %></p>
   <p><strong>Mobile Number:</strong> <%= mobile %></p>
   <p><strong>Address:</strong> <%= address %></p>
   <p><strong>Branch:</strong> <%= branch %></p>
   <p><strong>Year:</strong> <%= year %></p>
   <p><strong>Percentage:</strong> <%= percentage %></p>
</div>

</body>
</html>
```

Step 4: Once the project completes and it runs as the tomcat sever then it will run on the port 8082. Refer the blow images for the better understanding.

GIRIJANANDACHOWDHURYUNIVERSITY

Hathkhowapara,Azara,Guwahati-781017, Assam

Output:

Registration Form: