JAVA LAB

OBCA-DS-454

MANAV RACHNA INTERNATIONAL INSTITUTE OF RESEARCH AND STUDIES

Submitted by	
Student name	**Krishaank Shukla**
Roll number	23/SCA/BCA(DS&BDA)/ **017**
Programme	BCA (DS-BDA)
Semester	4th
Section	BCA IV E
Department	School of Computer Applications
Batch	2023-2026
Submitted to	
Faculty name	**Dr. Priyanka Sharma**

MANAV RACHNA
Vidyanatariksha

SCHOOL OF COMPUTER APPLICATIONS

Sr. No.	Title	Date	Signature
1	Write a program to find the average and sum of the N numbers using Command line argument.		
2	Write a program to demonstrate type casting.		
3.	Write a program to generate prime numbers between 1 & given number		
4.	Write a program to design a class account using the inheritance and static members which show all functions of a bank (Withdrawl, deposit)		
5.	Write a program to create a simple class to find out the area and perimeter of rectangle using super and this keyword.		
6.	Write a program to find the factorial of a given number using recursion.		
7.	Write a program to design a class using abstract methods and abstract classes.		
8.	Write a program to count the number of objects created for a class using static member function		
9.	Write a program to demonstrate the use of function overloading.		
10.	Write a program to demonstrate the use of multiple inheritance.		
11.	Write a program that show the partial implementation of Interface		
12.	Write a program to design a string class that perform string method(Equal, Reverse the string, change case).		
13.	Write a program to handle the exception using try and multiple catch block.		

14.	Write a program to create a package that access the member of External class as well as same package.		
15.	Write a program that import the user define package and access the Member variable of classes that contained by package.		
16.	Write a program to handle the user defined exception using throw keyword.		
17.	Write a program to create a class component that shows controls and event handling on that controls.		
18.	(mathcalc).		
	Write a program to draw the line, Rectangle, oval, text using the graphics method.		
19.	Write a program to create a menu using the frame.		
20.	Write a program to create a menu using the frame.		
21.	Write a program to implement the flow layout and border layout.		
22.	Write a program to imp Write a program to create a dialogbox. lement the gridLayout, cardLayout.		
23.	Write a program to implement the gridLayout, cardLayout.		
24.	Write a program to create Frame that display the student information		

Experiment No: 1

```
public class SumAverage {
   public static void main(String[] args) {
      if (args.length == 0) {
         System.out.println("Please provide numbers as command-line arguments.");
         return;
      }

      int sum = 0;
      for (String arg : args) {
         try {
            int num = Integer.parseInt(arg);
            sum += num;
         } catch (NumberFormatException e) {
            System.out.println(arg + " is not a valid integer.");
            return;
         }
      }

      double average = (double) sum / args.length;

      System.out.println("Sum = " + sum);
      System.out.println("Average = " + average);
   }
}
```

Result/Output:

Compile:

```
bash

javac SumAverage.java
```

Run (example with 5 numbers):

```
bash

java SumAverage 10 20 30 40 50
```

Experiment No: 2

```
public class TypeCastingDemo {
   public static void main(String[] args) {
      // Implicit type casting (widening)
      int intValue = 100;
      double doubleValue = intValue; // int to double
      System.out.println("Implicit Type Casting:");
      System.out.println("Integer Value: " + intValue);
      System.out.println("Double Value after implicit casting: " + doubleValue);

      // Explicit type casting (narrowing)
      double anotherDoubleValue = 9.78;
      int anotherIntValue = (int) anotherDoubleValue; // double to int
      System.out.println("\nExplicit Type Casting:");
      System.out.println("Double Value: " + anotherDoubleValue);
      System.out.println("Integer Value after explicit casting: " + anotherIntValue);
   }
}
```

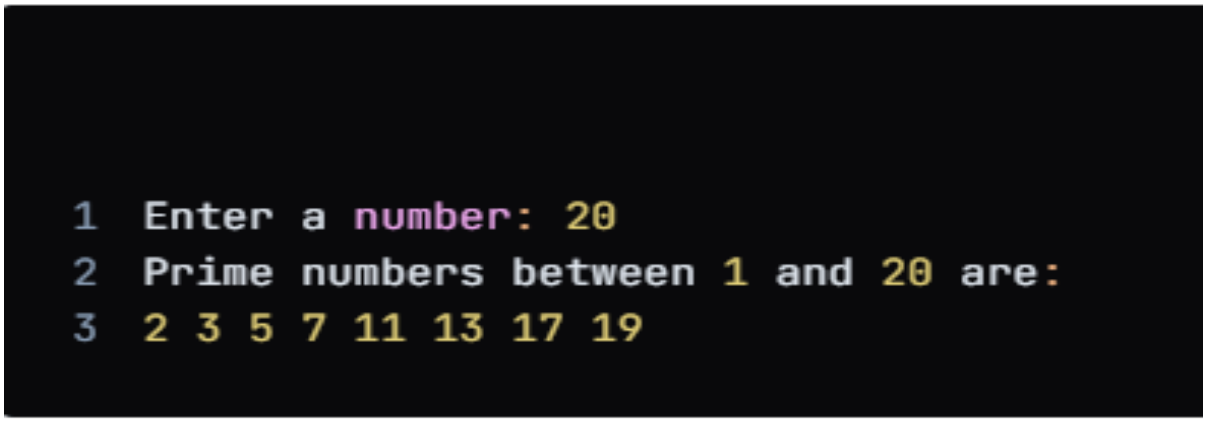
Result/Output:

```
Implicit Type Casting:
Integer Value: 100
Double Value after implicit casting: 100.0

Explicit Type Casting:
Double Value: 9.78
Integer Value after explicit casting: 9
```

Experiment No: 3

```
import java.util.Scanner;

public class PrimeNumberGenerator {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter a number: ");
      int limit = scanner.nextInt();

      System.out.println("Prime numbers between 1 and " + limit + " are:");
      for (int num = 2; num <= limit; num++) {
         if (isPrime(num)) {
            System.out.print(num + " ");
         }
      }
         scanner.close();
   }
    public static boolean isPrime(int number) {
      if (number <= 1) {
         return false;
      }
      for (int i = 2; i <= Math.sqrt(number); i++) {
         if (number % i == 0) {
            return false;
         }
      }
      return true;
   }
}
```

Result/Output:



Experiment No: 4

```
class Account {
   private static int accountCount = 0; // Static member to keep track of the number of
accounts
   protected double balance; // Protected member to allow access in subclasses

   public Account() {
      accountCount++;
      balance = 0.0;
   }
   public void deposit(double amount) {
      if (amount > 0) {
         balance += amount;
         System.out.println("Deposited: $" + amount);
      } else {
         System.out.println("Deposit amount must be positive.");
      }   }
   public void withdraw(double amount) {
      if (amount > 0 && amount <= balance) {
         balance -= amount;
         System.out.println("Withdrew: $" + amount);
      } else {
         System.out.println("Insufficient balance or invalid amount.");
      }
   }
    public double getBalance() {
      return balance;
   }
    public static int getAccountCount() {
      return accountCount;
   }}
 class SavingsAccount extends Account {
   private double interestRate;
    public SavingsAccount(double interestRate) {
      super(); // Call the constructor of the superclass
      this.interestRate = interestRate;
   }
    public void applyInterest() {
      double interest = balance * interestRate / 100;
      deposit(interest);
      System.out.println("Interest applied: $" + interest);
   }}
public class Bank {
   public static void main(String[] args) {
      SavingsAccount myAccount = new SavingsAccount(5.0); // 5% interest rate
```

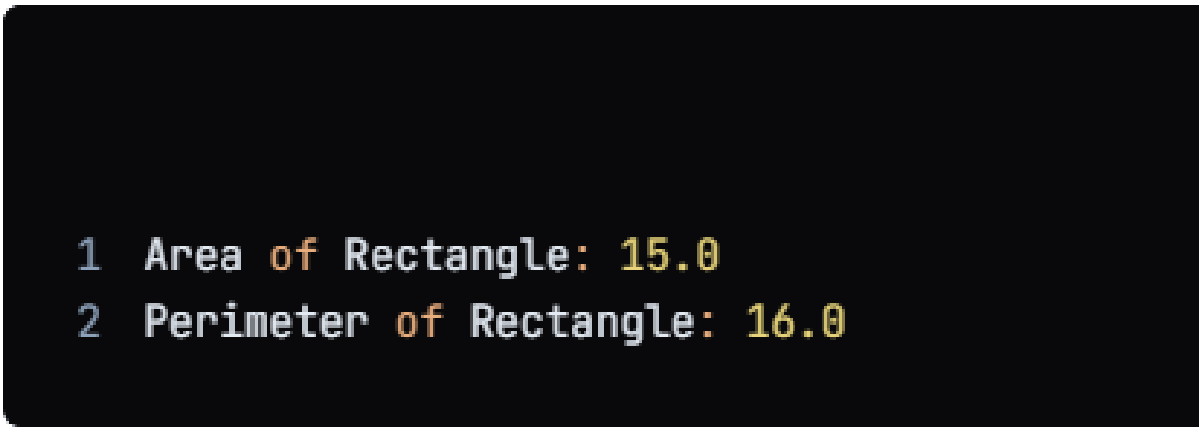
```
        myAccount.deposit(1000);
        myAccount.withdraw(200);
        myAccount.applyInterest();
         System.out.println("Current Balance: $" + myAccount.getBalance());
        System.out.println("Total Accounts Created: " + Account.getAccountCount());
    }
}
```

Result/Output:

Experiment No: 5

```
class Shape {
   protected double length;
   protected double width;

   public Shape(double length, double width) {
      this.length = length;
      this.width = width;
   }}
 class Rectangle extends Shape {
   public Rectangle(double length, double width) {
      super(length, width); // Call the constructor of the superclass
   }
    public double area() {
      return length * width; // Calculate area
   }
   public double perimeter() {
      return 2 * (length + width); // Calculate perimeter
   }}
public class RectangleDemo {
   public static void main(String[] args) {
      Rectangle rectangle = new Rectangle(5.0, 3.0); // Create a rectangle with length 5.0
and width 3.0

      System.out.println("Area of Rectangle: " + rectangle.area());
      System.out.println("Perimeter of Rectangle: " + rectangle.perimeter());
   }}
```

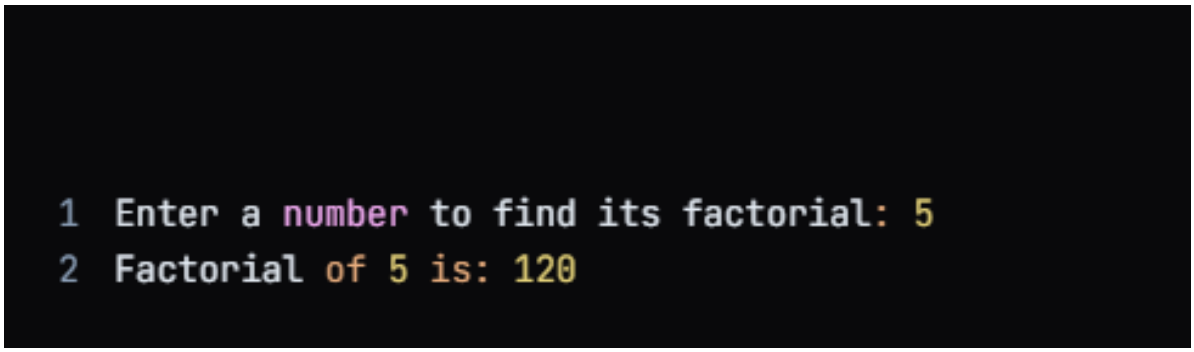
Result/Output:



Experiment No: 6

```
import java.util.Scanner;

public class FactorialCalculator {

   // Recursive method to calculate factorial
   public static long factorial(int n) {
      if (n == 0) {
         return 1; // Base case: 0! = 1
      } else {
         return n * factorial(n - 1); // Recursive case
      }
   }
    public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter a number to find its factorial: ");
      int number = scanner.nextInt();

      if (number < 0) {
         System.out.println("Factorial is not defined for negative numbers.");
      } else {
         long result = factorial(number);
         System.out.println("Factorial of " + number + " is: " + result);
      }
            scanner.close();
   }}
```

Result/Output:



Experiment No: 7

abstract class Shape { // Abstract method to calculate area abstract double area(); }

class Circle extends Shape { private double radius;

```
public Circle(double radius) {
    this.radius = radius;
}@Override
double area() {
    return Math.PI * radius * radius; // Area of circle = πr²
}
```

}class Rectangle extends Shape

{ private double length; private double width;

```
public Rectangle(double length, double width) {
    this.length = length;
    this.width = width;
}@Override
double area() {
    return length * width; // Area of rectangle = length * width
}}
```

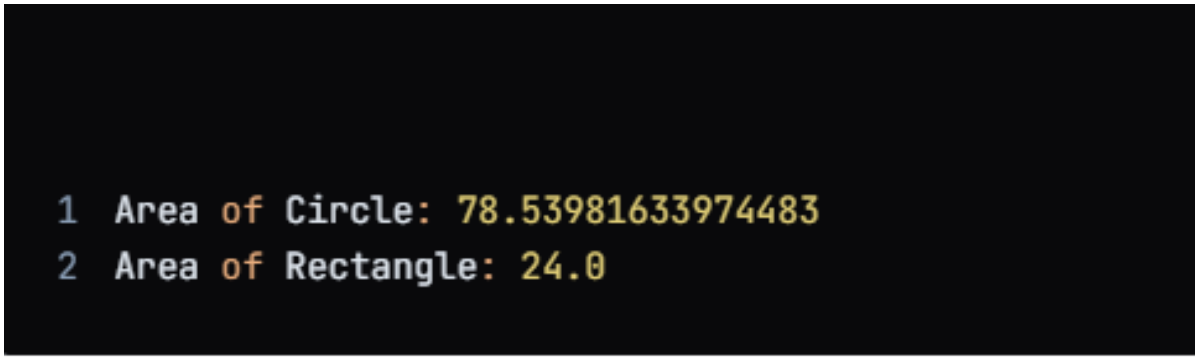
public class ShapeDemo

{ public static void main(String[] args)

{ Shape circle = new Circle(5.0); // Create a Circle object Shape rectangle = new Rectangle(4.0, 6.0); // Create a Rectangle object

```
   System.out.println("Area of Circle: " + circle.area());
    System.out.println("Area of Rectangle: " + rectangle.area());
}}
```

Result/Output:



Experiment No: 8

```
class ObjectCounter {
   // Static member to keep track of the number of objects created
   private static int objectCount = 0;

   // Constructor increments the object count
   public ObjectCounter() {
      objectCount++;
   }

   // Static method to get the current object count
   public static int getObjectCount() {
      return objectCount;
   }
}
 public class ObjectCounterDemo {
   public static void main(String[] args) {
      ObjectCounter obj1 = new ObjectCounter(); // First object
      ObjectCounter obj2 = new ObjectCounter(); // Second object
      ObjectCounter obj3 = new ObjectCounter(); // Third object

      // Display the number of objects created
      System.out.println("Number of objects created: " +
ObjectCounter.getObjectCount());
   }}
```

Result/Output:

Experiment No: 9

```
class MathOperations {

   // Method to add two integers
   public int add(int a, int b) {
      return a + b;
   }
    // Method to add three integers
   public int add(int a, int b, int c) {
      return a + b + c;
   }
    // Method to add two double values
   public double add(double a, double b) {
      return a + b;
   }}
 public class FunctionOverloadingDemo {
   public static void main(String[] args) {
      MathOperations mathOps = new MathOperations();
       // Calling the overloaded add methods
      int sum1 = mathOps.add(5, 10); // Calls the first method
      int sum2 = mathOps.add(5, 10, 15); // Calls the second method
      double sum3 = mathOps.add(5.5, 10.5); // Calls the third method
       // Displaying the results
      System.out.println("Sum of two integers: " + sum1);
      System.out.println("Sum of three integers: " + sum2);
      System.out.println("Sum of two doubles: " + sum3);
   }}
```

Result/Output:

```
1  Sum of two integers: 15
2  Sum of three integers: 30
3  Sum of two doubles: 16.0
```

Experiment No: 10

```
// First interface
interface Animal {
   void eat();
}
// Second interface
interface Pet {
   void play();
}
// Class that implements both interfaces
class Dog implements Animal, Pet {
   @Override
   public void eat() {
      System.out.println("Dog is eating.");
   }
    @Override
   public void play() {
      System.out.println("Dog is playing.");
   }
}
 public class MultipleInheritanceDemo {
   public static void main(String[] args) {
      Dog dog = new Dog(); // Create a Dog object
      dog.eat(); // Call the eat method
      dog.play(); // Call the play method
   }}
```

Result/Output:

Experiment No: 11

```
// Interface with a default method
interface Animal {
   void eat(); // Abstract method

   // Default method with a partial implementation
   default void sleep() {
      System.out.println("This animal is sleeping.");
   }}
 // Class that implements the Animal interface
class Dog implements Animal {
   @Override
   public void eat() {
      System.out.println("Dog is eating.");
   }}
 class Cat implements Animal {
   @Override
   public void eat() {
      System.out.println("Cat is eating.");
   }}
public class PartialImplementationDemo {
   public static void main(String[] args) {
      Dog dog = new Dog(); // Create a Dog object
      Cat cat = new Cat(); // Create a Cat object
       // Call methods on Dog
      dog.eat(); // Calls the implemented method
      dog.sleep(); // Calls the default method
       // Call methods on Cat
      cat.eat(); // Calls the implemented method
      cat.sleep(); // Calls the default method
   }}
```

Result/Output:

```
1 Dog is eating.
2 This animal is sleeping.
3 Cat is eating.
4 This animal is sleeping.
```

Experiment No: 12

```
class MyString {
   private String str;

   // Constructor to initialize the string
   public MyString(String str) {
      this.str = str;
   }
    // Method to check equality with another string
   public boolean equals(MyString other) {
      return this.str.equals(other.str);
   }
    // Method to reverse the string
   public String reverse() {
      return new StringBuilder(str).reverse().toString();
   }
    // Method to change the case of the string
   public String changeCase() {
      StringBuilder changedCase = new StringBuilder();
      for (char c : str.toCharArray()) {
         if (Character.isUpperCase(c)) {
            changedCase.append(Character.toLowerCase(c));
         } else {
            changedCase.append(Character.toUpperCase(c));
         }
       }
      return changedCase.toString();
   }
    // Method to get the original string
   public String getString() {
      return str;
   }}
 public class StringClassDemo {
   public static void main(String[] args) {
      MyString str1 = new MyString("Hello World");
      MyString str2 = new MyString("hello world");
       // Check equality
      System.out.println("Are the strings equal? " + str1.equals(str2));
       // Reverse the string
      System.out.println("Reversed string: " + str1.reverse());
       // Change case of the string
      System.out.println("Changed case: " + str1.changeCase());
   }
}
```

Result/Output:

```
1 Are the strings equal? false
2 Reversed string: dlroW olleH
3 Changed case: hELLO wORLD
```

Experiment No: 13

```
import java.util.Scanner;

public class ExceptionHandlingDemo {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      try {
         System.out.print("Enter a numerator: ");
         String numeratorInput = scanner.nextLine();
         int numerator = Integer.parseInt(numeratorInput);
 // May throw NumberFormatException

         System.out.print("Enter a denominator: ");
         String denominatorInput = scanner.nextLine();
         int denominator = Integer.parseInt(denominatorInput);
 // May throw NumberFormatException

         // Attempt to perform division

         int result = numerator / denominator;
 // May throw ArithmeticException
         System.out.println("Result: " + result);
      } catch (ArithmeticException e) {
         System.out.println("Error: Cannot divide by zero.");
      } catch (NumberFormatException e) {
         System.out.println("Error: Please enter valid integers.");
      } catch (Exception e) {
         System.out.println("An unexpected error occurred: " + e.getMessage());
      } finally {
         scanner.close();

 // Close the scanner resource
      }
   }
}
```

Result/Output:

1. Valid Input:

```
Enter a numerator: 10
Enter a denominator: 2
Result: 5
```

2. Division by Zero:

```
Enter a numerator: 10
Enter a denominator: 0
Error: Cannot divide by zero.
```

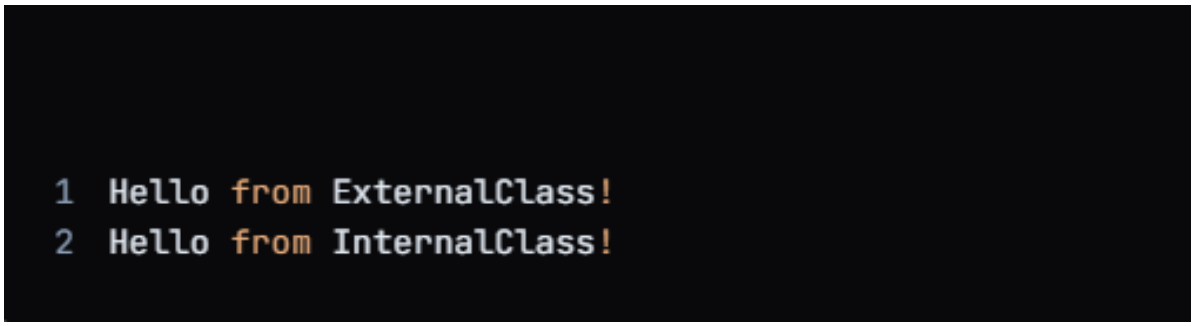
3. Invalid Input:

```
Enter a numerator: ten
Error: Please enter valid integers.
```

Experiment No: 14

```
import mypackage.ExternalClass;
import mypackage.InternalClass;

public class MainClass {
   public static void main(String[] args) {
      // Create an instance of ExternalClass
      ExternalClass external = new ExternalClass();
      external.display(); // Accessing method from ExternalClass

      // Create an instance of InternalClass
      InternalClass internal = new InternalClass();
      internal.show();
 // Accessing method from InternalClass
   }
}
```

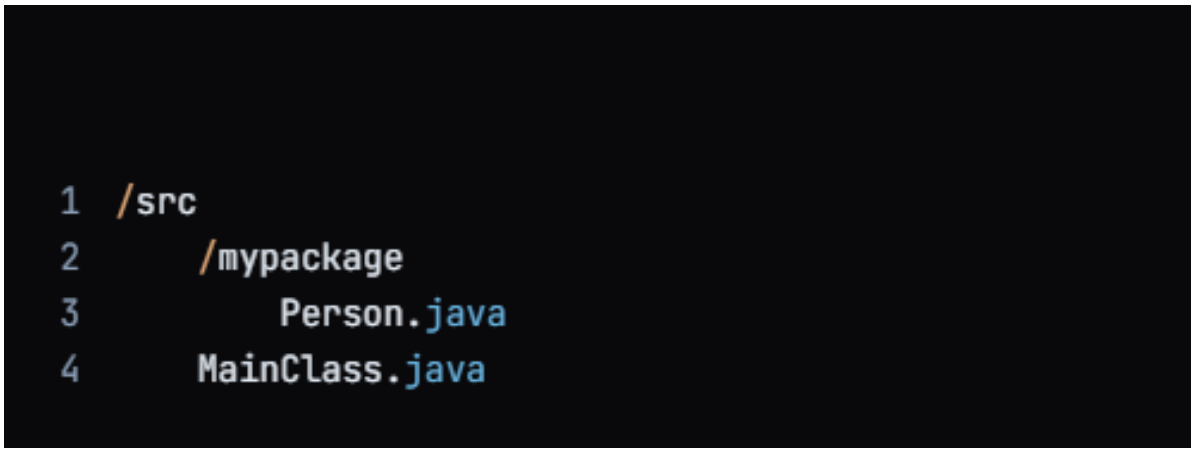
Result/Output:



Experiment No: 15

```
import mypackage.Person;

public class MainClass {
   public static void main(String[] args) {
      // Create an instance of Person
      Person person = new Person("Alice");

      // Accessing the member variable
      System.out.println("Person's name: " + person.name);
   }
}
```

Result/Output:



Experiment No: 16

```
// Custom Exception Class
class InvalidAgeException extends Exception {
   public InvalidAgeException(String message) {
      super(message);

   }
}

// Class to demonstrate the use of the custom exception
public class AgeValidator {
   // Method to validate age
   public static void validateAge(int age) throws InvalidAgeException {
      if (age < 18) {
         throw new InvalidAgeException("Age must be 18 or older.");
      } else {
         System.out.println("Age is valid: " + age);
      }
   }

   public static void main(String[] args) {
      try {
         validateAge(15); // This will throw an exception
      } catch (InvalidAgeException e) {
         System.out.println("Caught Exception: " + e.getMessage());
      }

      try {

         validateAge(20); // This will not throw an exception
      } catch (InvalidAgeException e) {
         System.out.println("Caught Exception: " + e.getMessage());
      }
   }
}
```

Result/Output:

2. Compile the program using the command:

```
1  javac AgeValidator.java
```

3. Run the program using the command:

```
1  java AgeValidator
```

Experiment No: 17

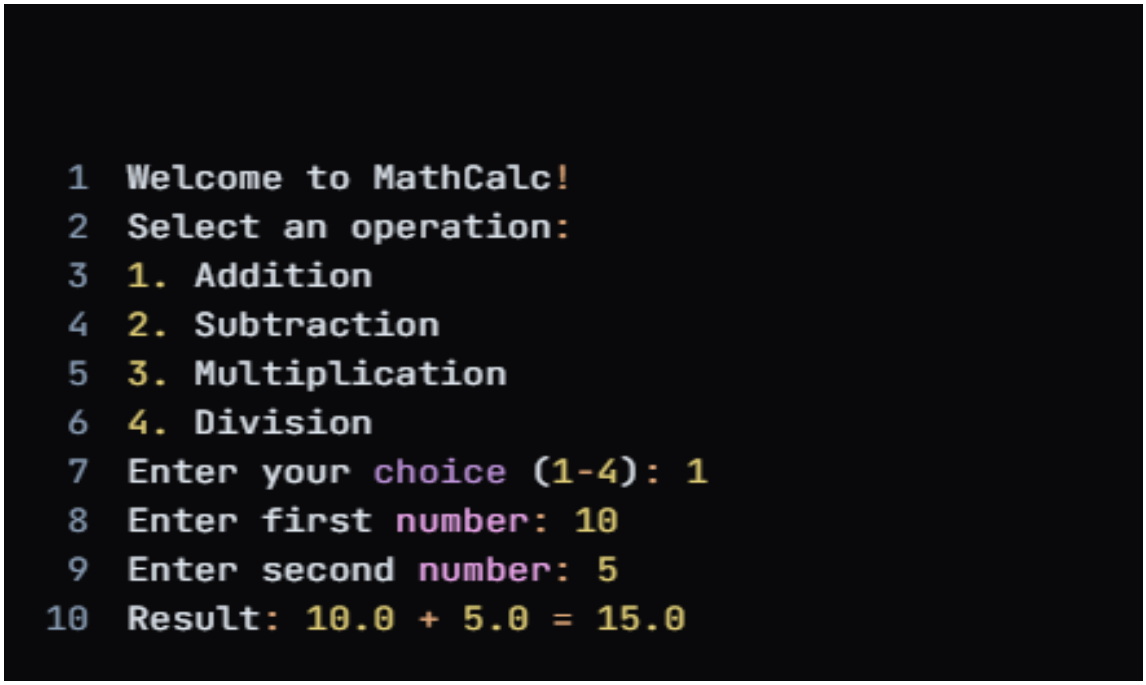
```
import java.awt.*;
import java.awt.event.*;

class EventHandling extends Frame implements ActionListener
{
    TextField textField;

    EventHandling ()
    {
        textField = new TextField ();
        textField.setBounds (60, 50, 170, 20);
        Button button = new Button ("Show");
        button.setBounds (90, 140, 75, 40);

        button.addActionListener (this);
        add (button);
        add (textField);
        setSize (250, 250);
        setLayout (null);
        setVisible (true);
    }


    public void actionPerformed (ActionEvent e)
    {
        textField.setText ("Hello World");
    }

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

Result/Output:

Experiment No: 18

```
import java.util.Scanner;

public class MathCalc {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Welcome to MathCalc!");
        System.out.println("Select an operation:");
        System.out.println("1. Addition");
        System.out.println("2. Subtraction");
        System.out.println("3. Multiplication");
        System.out.println("4. Division");
                System.out.print("Enter your choice (1-4): ");
        int choice = scanner.nextInt();
             System.out.print("Enter first number: ");
        double num1 = scanner.nextDouble();
               System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();
                double result = 0;
        switch (choice) {
            case 1:
                result = num1 + num2;
                System.out.println("Result: " + num1 + " + " +
num2 + " = " + result);
                break;
            case 2:
                result = num1 - num2;
                System.out.println("Result: " + num1 + " - " +
num2 + " = " + result);
                break;
            case 3:
                result = num1 * num2;
                System.out.println("Result: " + num1 + " * " +
num2 + " = " + result);
                break;
            case 4:
                if (num2 != 0) {
                    result = num1 / num2;
                    System.out.println("Result: " + num1 + " / " +
num2 + " = " + result);
                } else {
                    System.out.println("Error: Division by zero is
not allowed.");
                }
                break;
            default:
                System.out.println("Invalid choice. Please select
a valid operation.");
        }
        scanner.close();    }
}
```

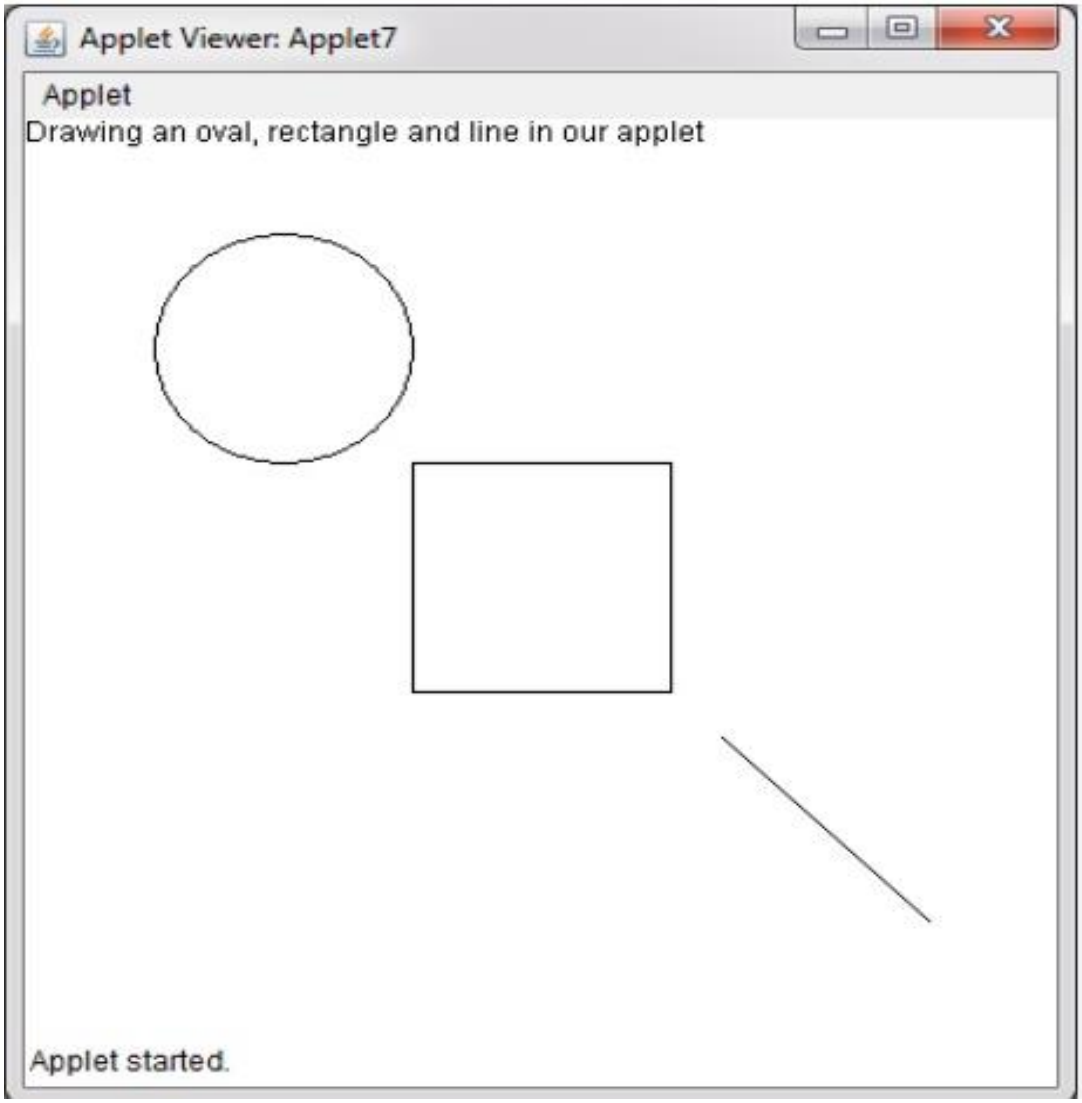
Result/Output:



Experiment No: 19

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

public class ShapeDrawing extends JPanel {

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    // Draw a line

    g.drawLine(50, 50, 200, 50);

    // Draw a rectangle

    g.drawRect(50, 70, 150, 100);
    // Draw an oval

    g.drawOval(50, 180, 150, 100);

    // Draw text
    g.drawString("Hello, Graphics!", 50, 300);
}

public static void main(String[] args) {
    JFrame frame = new JFrame("Shape Drawing Example");
    ShapeDrawing panel = new ShapeDrawing();
    frame.add(panel);

    frame.setSize(300, 400);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);

}
}
```

Result/Output:

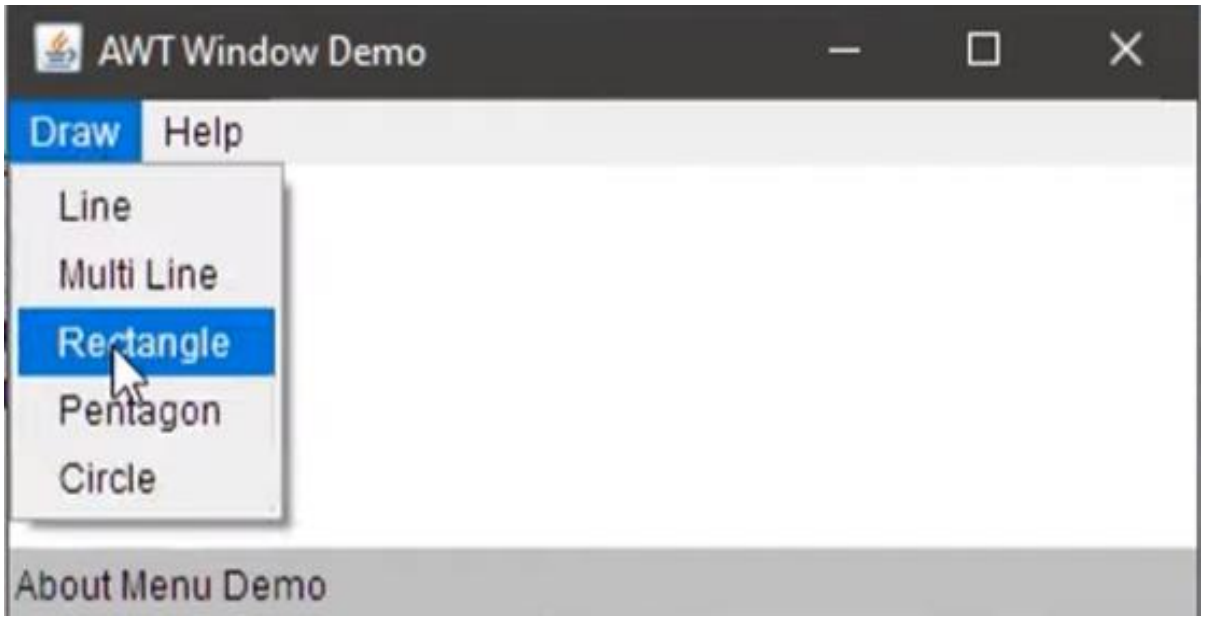


Experiment No: 20

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

public class AWTMenuExample extends Frame implements ActionListener {
    AWTMenuExample() {
      // Create menu bar
      MenuBar menuBar = new MenuBar();
       // Create "Draw" menu
      Menu drawMenu = new Menu("Draw");
      MenuItem line = new MenuItem("Line");
      MenuItem multiLine = new MenuItem("Multi Line");
      MenuItem rectangle = new MenuItem("Rectangle");
      MenuItem pentagon = new MenuItem("Pentagon");
      MenuItem circle = new MenuItem("Circle");
       // Add items to "Draw"
      drawMenu.add(line);
      drawMenu.add(multiLine);
      drawMenu.add(rectangle);
      drawMenu.add(pentagon);
      drawMenu.add(circle);
       // Add action listeners
      line.addActionListener(this);
      multiLine.addActionListener(this);
      rectangle.addActionListener(this);
      pentagon.addActionListener(this);
      circle.addActionListener(this);
       // Create "Help" menu
      Menu helpMenu = new Menu("Help");
       // Add menus to the menu bar
      menuBar.add(drawMenu);
      menuBar.add(helpMenu);
       // Set menu bar to the frame
      setMenuBar(menuBar);
       // Frame settings
      setTitle("AWT Window Demo");
      setSize(400, 300);
      setLayout(null);
      setVisible(true);
       // Add window close handler
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            dispose();
         }
      });
   }
```

```
    public void actionPerformed(ActionEvent e) {
      System.out.println(e.getActionCommand() + " selected");
    }
    public static void main(String[] args) {
      new AWTMenuExample();
    }
}
```

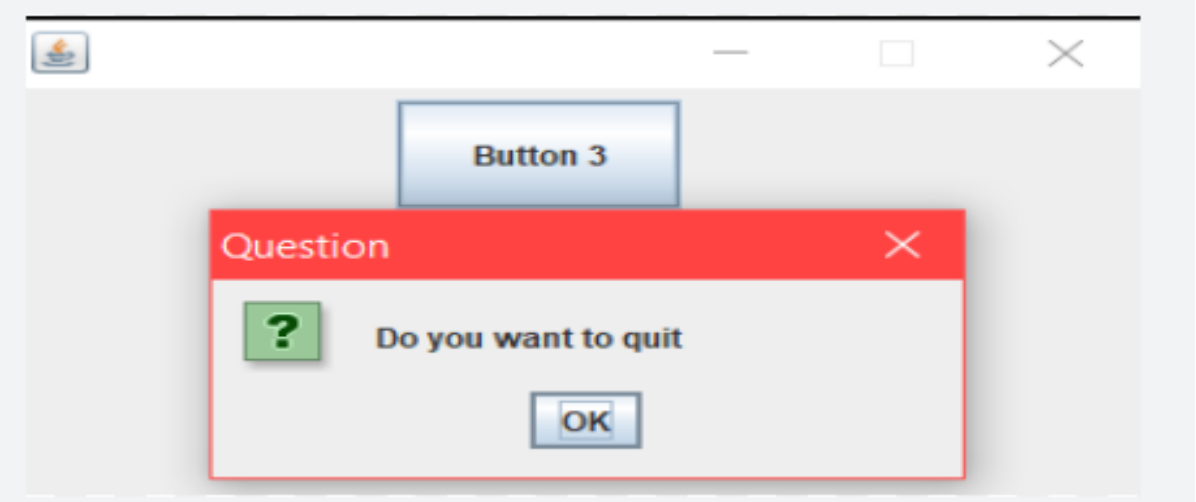
Result/Output:



Experiment No: 21

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

public class DialogBoxExample { public static void main(String[] args) { // Create frame
JFrame frame = new JFrame("Dialog Demo"); frame.setSize(300, 200);
frame.setLayout(null);

   // Create button
    JButton button = new JButton("Button 3");
    button.setBounds(100, 60, 100, 30);
    // Add action listener
    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(
                frame,
                "Do you want to quit",
                "Question",
                JOptionPane.QUESTION_MESSAGE
            );
        }
    });
    // Add button and frame settings
    frame.add(button);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}}
```

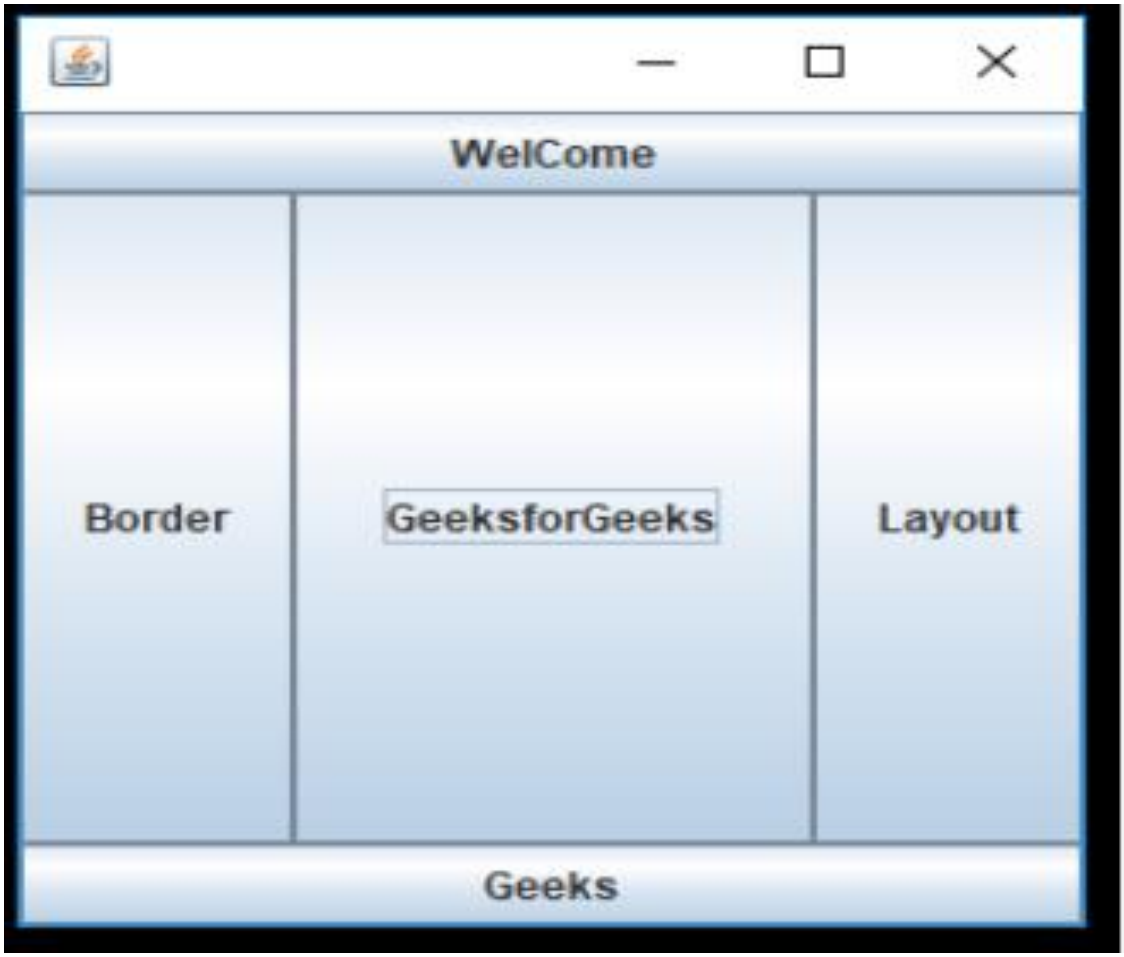
Result/Output:



Experiment No: 22

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

public class LayoutExample extends JFrame {

public LayoutExample() {
    // Set up the frame
    setTitle("FlowLayout and BorderLayout Example");
    setSize(400, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // Create a panel with FlowLayout
    JPanel flowPanel = new JPanel();
    flowPanel.setLayout(new FlowLayout());
     // Add buttons to the FlowLayout panel
    flowPanel.add(new JButton("Button 1"));
    flowPanel.add(new JButton("Button 2"));
    flowPanel.add(new JButton("Button 3"));
    // Create a panel with BorderLayout
    JPanel borderPanel = new JPanel();
    borderPanel.setLayout(new BorderLayout());
    // Add buttons to the BorderLayout panel
    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);

    // Add both panels to the frame
    setLayout(new BorderLayout());
    add(flowPanel, BorderLayout.NORTH);
    add(borderPanel, BorderLayout.CENTER);
}

public static void main(String[] args) {
    // Create and display the GUI
    LayoutExample layoutExample = new LayoutExample();
    layoutExample.setVisible(true);
}


}
```

Result/Output:

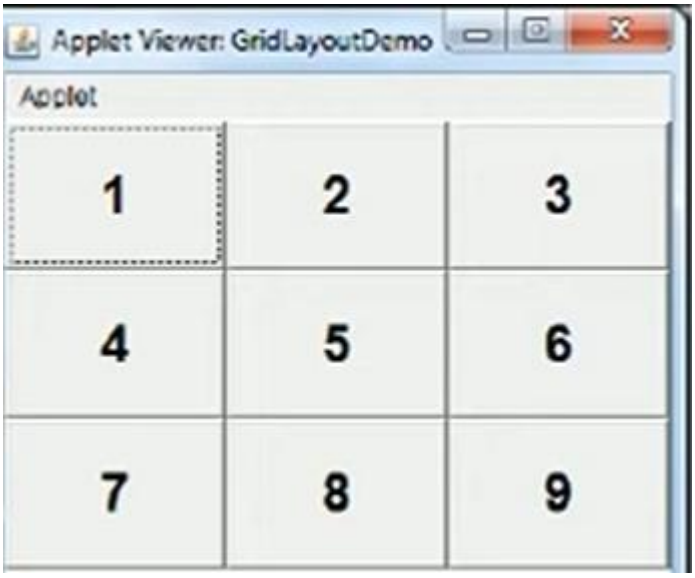

WelCome
Border
GeeksforGeeks
Layout
Geeks

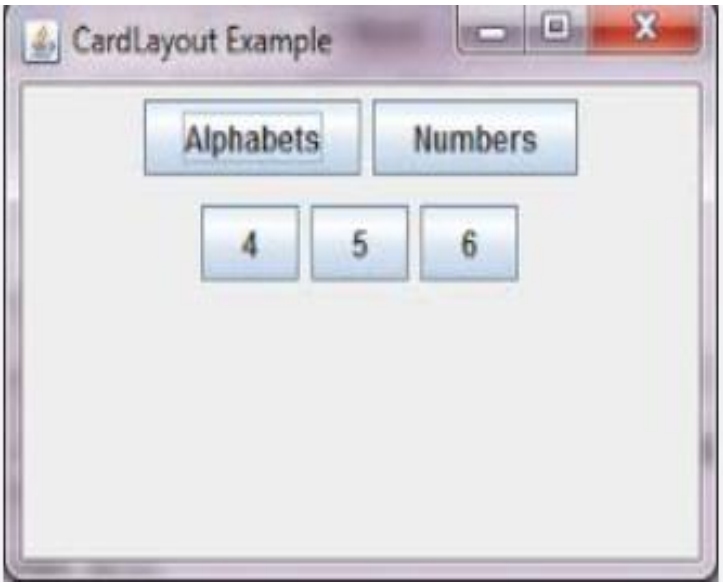
Experiment No: 23

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

public class CombinedLayoutExample extends JFrame implements ActionListener
{ CardLayout cardLayout; JPanel cardPanel;

public CombinedLayoutExample() {
    setTitle("CardLayout + GridLayout Example");
    setSize(400, 300);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLayout(new BorderLayout());
    // Top buttons to switch cards
    JPanel buttonPanel = new JPanel();
    JButton btnAlphabet = new JButton("Alphabets");
    JButton btnNumbers = new JButton("Numbers");
    buttonPanel.add(btnAlphabet);
    buttonPanel.add(btnNumbers);
    // Card panel with CardLayout
    cardLayout = new CardLayout();
    cardPanel = new JPanel(cardLayout);
    // Alphabet card using GridLayout (3x1)
    JPanel alphabetPanel = new JPanel(new GridLayout(1, 3, 10,
10));
    alphabetPanel.add(new JButton("A"));
    alphabetPanel.add(new JButton("B"));
    alphabetPanel.add(new JButton("C"));
    // Number card using GridLayout (3x3)
    JPanel numberPanel = new JPanel(new GridLayout(3, 3, 10, 10));
    for (int i = 1; i <= 9; i++) {
        numberPanel.add(new JButton(String.valueOf(i)));
    }
    // Add both cards
    cardPanel.add(alphabetPanel, "Alphabets");
    cardPanel.add(numberPanel, "Numbers");
    // Add action listeners
    btnAlphabet.addActionListener(this);
    btnNumbers.addActionListener(this);
    // Add panels to frame
    add(buttonPanel, BorderLayout.NORTH);
    add(cardPanel, BorderLayout.CENTER);
    setVisible(true);
}

public void actionPerformed(ActionEvent e) {
    cardLayout.show(cardPanel, e.getActionCommand());
}
public static void main(String[] args) {
```

```
    new CombinedLayoutExample();
}}
```

Result/Output:





Experiment No: 24

import javax.swing.*; import javax.swing.table.DefaultTableModel; import java.awt.*; import java.awt.event.*;

public class StudentRecordForm extends JFrame implements ActionListener { // Form components JTextField rollNoField, nameField, classField, sectionField; JTextArea addressArea; JTable table; DefaultTableModel model; JButton insertBtn, viewBtn, clearBtn, exitBtn;

```
public StudentRecordForm() {
    setTitle("School Record Management System");
    setSize(700, 500);
    setLayout(null);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    // Title label
    JLabel title = new JLabel("School Record Management System",
JLabel.CENTER);
    title.setFont(new Font("Serif", Font.BOLD | Font.ITALIC, 20));
    title.setBounds(150, 10, 400, 30);
    add(title);
    // Labels and text fields
    JLabel rollLabel = new JLabel("Roll No:");
    rollLabel.setBounds(50, 60, 100, 25);
    add(rollLabel);
    rollNoField = new JTextField();
    rollNoField.setBounds(150, 60, 100, 25);
    add(rollNoField);
    JLabel classLabel = new JLabel("Class:");
    classLabel.setBounds(400, 60, 50, 25);
    add(classLabel);
    classField = new JTextField();
    classField.setBounds(460, 60, 50, 25);
    add(classField);
    JLabel nameLabel = new JLabel("Name:");
    nameLabel.setBounds(50, 100, 100, 25);
    add(nameLabel);
    nameField = new JTextField();
    nameField.setBounds(150, 100, 200, 25);
    add(nameField);
    JLabel sectionLabel = new JLabel("Section:");
    sectionLabel.setBounds(400, 100, 60, 25);
    add(sectionLabel);
    sectionField = new JTextField();
    sectionField.setBounds(460, 100, 50, 25);
    add(sectionField);
    JLabel addressLabel = new JLabel("Address:");
    addressLabel.setBounds(50, 140, 100, 25);
```

```
    add(addressLabel);
    addressArea = new JTextArea();
    JScrollPane scrollPane = new JScrollPane(addressArea);
    scrollPane.setBounds(150, 140, 400, 60);
    add(scrollPane);
    // Table
    model = new DefaultTableModel(new String[]{"Roll No.", "Name",
"Class", "Section", "Address"}, 0);
    table = new JTable(model);
    JScrollPane tablePane = new JScrollPane(table);
    tablePane.setBounds(50, 220, 600, 100);
    add(tablePane);
    // Buttons
    insertBtn = new JButton("Insert");
    insertBtn.setBounds(50, 340, 100, 30);
    insertBtn.addActionListener(this);
    add(insertBtn);
    viewBtn = new JButton("View data");
    viewBtn.setBounds(180, 340, 100, 30);
    viewBtn.addActionListener(this);
    add(viewBtn);
    clearBtn = new JButton("Clear");
    clearBtn.setBounds(310, 340, 100, 30);
    clearBtn.addActionListener(this);
    add(clearBtn);
    exitBtn = new JButton("Exit");
    exitBtn.setBounds(440, 340, 100, 30);
    exitBtn.addActionListener(this);
    add(exitBtn);
    setVisible(true);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == insertBtn) {
        String roll = rollNoField.getText();
        String name = nameField.getText();
        String cls = classField.getText();
        String section = sectionField.getText();
        String address = addressArea.getText();
        if (!roll.isEmpty() && !name.isEmpty()) {
            model.addRow(new Object[]{roll, name, cls, section,
address});
            JOptionPane.showMessageDialog(this, "Record Added
Successfully", "Message", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(this, "Please fill in
Roll No and Name", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    } else if (e.getSource() == clearBtn) {
        rollNoField.setText("");
```

```
        nameField.setText("");
        classField.setText("");
        sectionField.setText("");
        addressArea.setText("");
    } else if (e.getSource() == exitBtn) {
        System.exit(0);
    } else if (e.getSource() == viewBtn) {
        // View logic already handled by JTable model
        JOptionPane.showMessageDialog(this, "Table updated with
current records.", "Info", JOptionPane.INFORMATION_MESSAGE);
    }
public static void main(String[] args) {
    new StudentRecordForm();
}}
```

Result/Output: