DRONACHARYA
College of Engineering

LABORATORY MANUAL

B.Tech. Semester- V

JAVA LAB
Subject code: LC-CSE-327G

Prepared by:	**Checked by:**	**Approved by:**
Dr. Ashima Mehta	Dr. Ashima Mehta	Name : Prof. (Dr.) Isha Malhotra
Sign.: ……………………..	**Sign.: ………………….**	**Sign.: ………………….**

TABLE OF CONTENTS

VISION AND MISSION OF THE INSTITUTE

Vision:

“Empowering human values and advanced technical education to navigate and address global challenges with excellence.”

Mission:

- M1: Seamlessly integrate human values with advanced technical education.
- M2: Supporting the cultivation of a new generation of innovators who are not only skilled but also ethically responsible.
- M3: Inspire global citizens who are equipped to create positive and sustainable impact, driving progress towards a more inclusive and harmonious world.

VISION AND MISSION OF THE DEPARTMENT

Vision:

“Steering the future of computer science through innovative advancements, fostering ethical values and principles through technical education.”

Mission:

M1: Directing future innovations in computer science through revolutionary progress.

M2: Instilling a foundation of ethical values and principles in every technologist.

M3: Offering a comprehensive technical education to equip individuals for a meaningful and influential future.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOS)

PEO1: Apply the technical competence in Computer Science and Engineering for solving problems in the real world.

PEO2: Carry out research and develop solutions on problems of social applications.

PEO3: Work in a corporate environment, demonstrating team skills, work morals, flexibility and lifelong learning.

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Exhibit design and programming skills to develop and mechanize business solutions using revolutionary technologies.

PSO2: Learn strong theoretical foundation leading to brilliance and enthusiasm towards research, to provide well-designed solutions to complicated problems.

PSO3: Work effectively with diverse Engineering fields as a team to design, build and develop system applications.

UNIVERSITY SYLLABUS

1. Write a java program to check whether given alphabet is vowel or not.
2. Write a java program to implement method overloading.
3. Write a java program to implement method riding.
4. Write a java program to solve Fibonacci series using recursive method.
5. Write a java program to create a class circle and initialize and display its variables center and radius.
6. Write a java program to implement parameterized constructor.
7. Write a java program to implement stack operations using array.
8. Write a java program to perform addition of two matrices.
9. Write a java program to implement exception handling.
10. Write a java program extend thread class.
11. Write a java program to implement ArrayList using collection framework.

COURSE OUTCOMES (COs)

Upon successful completion of the course, the students will:

1. Gain knowledge of the structure and model of the Java programming language, (knowledge)
2. Use the Java programming language for various programming technologies (understanding)
3. Develop software in the Java programming language

CO-PO Mapping:

	PSO1	PSO2	PSO3	PSO4	PSO5	PSO6	PSO7	PSO8	PSO9	PSO10	PSO11	PSO12
C327.1	3	3	2	2	2	1	-	1	-	1	1	3
C327.2	3	3	3	2	3	1	-	1	-	1	1	3
C327.3	3	3	2	1	2	1	-	1	-	1	1	3

CO-PSO Mapping:

	PSO1	PSO2	PSO3
C327.1	3	2	1
C327.2	3	3	2
C327.3	3	2	3

COURSE OVERVIEW

A Java lab course is a practical component accompanying a Java programming course that focuses on hands-on experience with the Java programming language. The course covers topics such as setting up the Java development environment, basic syntax, object-oriented programming (OOP) concepts, Java classes and packages, exception handling, file handling, and input/output operations. Students gain practical experience by working on programming exercises and projects, applying their knowledge to solve real-world problems. The course emphasizes writing clean, efficient, and well-structured code, and provides students with the skills and confidence to develop Java applications. By the end of the course, students should be able to design and implement Java programs, effectively utilize OOP principles, handle exceptions, and work with files and data input/output.

LIST OF EXPERIMENTS MAPPED WITH COs

S.No	Experiment	Course Outcome	Page No.
1	Write a java program to check whether given alphabet is vowel or not.	C327.1	1
2	Write a java program to implement method overloading.	C327.1	4
3	Write a java program to implement method riding.	C327.1, C327.2	8
4	Write a java program to solve Fibonacci series using recursive method.	C327.1, C327.2	12
5	Write a java program to create a class circle and initialize and display its variables center and radius.	C327.1	15
6	Write a java program to implement parameterized constructor.	C327.3	18
7	Write a java program to implement stack operations using array.	C327.1, C327.2	21
8	Write a java program to perform addition of two matrices.	C327.1	27
9	Write a java program to implement exception handling.	C327.3	31
10	Write a java program extend thread class.	C327.3	35
11	Write a java program to implement ArrayList using collection framework.	C327.3	39

DOs and DON'Ts

DOs

1. Login-on with your username and password.

2. Log off the Computer every time when you leave the Lab.

3. Arrange your chair properly when you are leaving the lab.

4. Put your bags in the designated area.

5. Ask permission to print.

DON'Ts

1. Do not share your username and password.

2. Do not remove or disconnect cables or hardware parts.

3. Do not personalize the computer setting.

4. Do not run programs that continue to execute after you log off.

5. Do not download or install any programs, games or music on computer in Lab.

6. Personal Internet use chat room for Instant Messaging (IM) and Sites is strictly prohibited.

7. No Internet gaming activities allowed.

8. Tea, Coffee, Water & Eatables are not allowed in the Computer Lab.

GENERAL SAFETY PRECAUTIONS

Precautions (In case of Injury or Electric Shock)

1. To break the victim with live electric source, use an insulator such as fire wood or plastic to break the contact. Do not touch the victim with bare hands to avoid the risk of electrifying yourself.

2. Unplug the risk of faulty equipment. If main circuit breaker is accessible, turn the circuit off.

3. If the victim is unconscious, start resuscitation immediately, use your hands to press the chest in and out to continue breathing function. Use mouth-to-mouth resuscitation if necessary.

4. Immediately call medical emergency and security. Remember! Time is critical; be best.

Precautions (In case of Fire)

1. Turn the equipment off. If power switch is not immediately accessible, take plug off.

2. If fire continues, try to curb the fire, if possible, by using the fire extinguisher or by covering it with a heavy cloth if possible isolate the burning equipment from the other surrounding equipment.

3. Sound the fire alarm by activating the nearest alarm switch located in the hallway.

4. Call security and emergency department immediately:

Emergency : 200 (Reception)

Security : 248 (Gate No.1)

GUIDELINES TO STUDENTS FOR REPORT PREPARATION

All students are required to maintain a record of the experiments conducted by them.

Guidelines for its preparation are as follows:-

1) All files must contain a title page followed by an index page. ***The files will not be signed by the faculty without an entry in the index page.***

2) Student's Name, Roll number and date of conduction of experiment must be written on all pages.

3) For each experiment, the record must contain the following

(i) Aim/Objective of the experiment

(ii) Pre-experiment work (as given by the faculty)

(iii) Lab assignment questions and their solutions

(iv) Test Cases (if applicable to the course)

(v) Results/ output

Note:

1. Students must bring their lab record along with them whenever they come for the lab.

2. Students must ensure that their lab record is regularly evaluated.

LAB ASSESSMENT CRITERIA

An estimated 10 lab classes are conducted in a semester for each lab course. These lab classes are assessed continuously. Each lab experiment is evaluated based on 5 assessment criteria as shown in following table. Assessed performance in each experiment is used to compute CO attainment as well as internal marks in the lab course.

Grading Criteria	**Exemplary (4)**	**Competent (3)**	**Needs Improvement (2)**	**Poor (1)**
AC1: Pre-Lab written work (this may be assessed through viva)	Complete procedure with underlined concept is properly written	Underlined concept is written but procedure is incomplete	Not able to write concept and procedure	Underlined concept is not clearly understood
AC2: Program Writing/ Modeling	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/ tools are applied, Program/solution written is readable	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/ tools are applied	Assigned problem is properly analyzed & correct solution designed	Assigned problem is properly analyzed
AC3:	Able to identify	Able to identify	Is dependent	Unable to

Identification & Removal of errors/ bugs	errors/ bugs and remove them	errors/ bugs and remove them with little bit of guidance	totally on someone for identification of errors/ bugs and their removal	understand the reason for errors/ bugs even after they are explicitly pointed out
AC4:Execution & Demonstration	All variants of input /output are tested, Solution is well demonstrated and implemented concept is clearly explained	All variants of input /output are not tested, However, solution is well demonstrated and implemented concept is clearly explained	Only few variants of input /output are tested, Solution is well demonstrated but implemented concept is not clearly explained	Solution is not well demonstrated and implemented concept is not clearly explained
AC5:Lab Record Assessment	All assigned problems are well recorded with objective, design constructs and solution along with Performance analysis using all variants of input	More than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done	Less than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done	Less than 40 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done

	and output	with all variants of input and output	with all variants of input and output	with all variants of input and output

LAB EXPERIMENTS

LAB EXPERIMENT 1

OBJECTIVE:

Write a java program to check whether given alphabet is vowel or not.

PRE-EXPERIMENT QUESTIONS:

1. How can you check whether a given alphabet is a vowel or not in Java?
2. What are the common approaches to implement vowel checking in a programming language?

BRIEF DISCUSSION AND EXPLANATION:

```
import java.util.Scanner;

public class VowelChecker {

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter an alphabet: ");

    char alphabet = scanner.next().charAt(0);



    // Convert the alphabet to lowercase for case-insensitive comparison

    alphabet = Character.toLowerCase(alphabet);



    if (isVowel(alphabet)) {

      System.out.println(alphabet + " is a vowel.");

    } else {
            System.out.println(alphabet + " is not a vowel.");
        }
    }


    public static boolean isVowel(char alphabet) {
        // Check if the alphabet is one of the vowels (a, e, i, o, u)
        return alphabet == 'a' || alphabet == 'e' || alphabet == 'i' || alphabet == 'o' || alphabet == 'u';
    }
}
```

1. The program starts by importing the `Scanner` class from the `java.util` package. This class allows us to read user input from the console.

2. In the `main` method, a `Scanner` object named `scanner` is created to read input from the user.

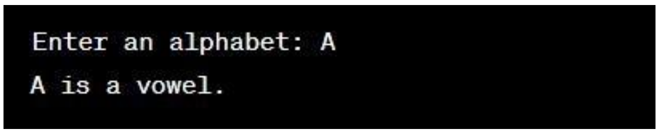
3. The program prompts the user to enter an alphabet by using `System.out.print("Enter an alphabet: ");`. The input is read as a string using `scanner.next()`, and then the first character is extracted using `charAt(0)` to obtain the alphabet.

4. The alphabet is converted to lowercase using `Character.toLowerCase(alphabet)` to make the comparison case-insensitive.

5. The program then calls the `isVowel` method, passing the alphabet as an argument. This method checks if the given alphabet is a vowel by comparing it with the lowercase vowels ('a', 'e', 'i', 'o', 'u'). It returns `true` if the alphabet is a vowel and `false` otherwise.

6. Based on the result of `isVowel`, the program prints the appropriate message to the console.

Output:



```
Enter an alphabet: A
A is a vowel.
```

POST EXPERIMENT QUESTIONS:

1. Explain the structure of the VowelCheck Java program.
2. What is the purpose of the isVowel method in the program? How does it check for vowels?

LAB EXPERIMENT 2

OBJECTIVE:

Write a java program to implement method overloading.

PRE-EXPERIMENT QUESTIONS:

1. What is method overloading in Java?
2. What are the requirements for method overloading?

BRIEF DISCUSSION AND EXPLANATION:

```
public class MethodOverloadingExample {

   public static void main(String[] args) {

      MethodOverloadingExample obj = new MethodOverloadingExample();



      obj.add(5, 10);

      obj.add(3.7, 2.5);

      obj.add("Hello", "World");

   }



   // Method to add two integers

   public void add(int a, int b) {

      int sum = a + b;

      System.out.println("Sum of integers: " + sum);
```

```
    }

    // Method to add two doubles
    public void add(double a, double b) {
        double sum = a + b;
        System.out.println("Sum of doubles: " + sum);
    }

    // Method to concatenate two strings
    public void add(String a, String b) {
        String concat = a + " " + b;
        System.out.println("Concatenation of strings: " + concat);
    }
}
```

Discussion:

1. The program defines a class called `MethodOverloadingExample`.

2. In the `main` method, an instance of the class is created using `MethodOverloadingExample obj = new MethodOverloadingExample();`.

3. The program demonstrates method overloading by defining multiple methods with the same name (`add`) but different parameter types.

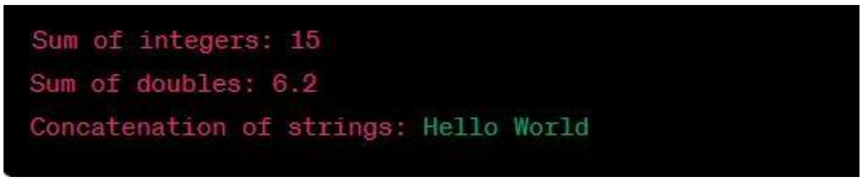
4. The first `add` method takes two integer parameters and calculates their sum. It then prints the result.

5. The second `add` method takes two double parameters and calculates their sum. It then prints the result.

6. The third `add` method takes two string parameters and concatenates them with a space in between. It then prints the result.

7. Inside the `main` method, the `add` method is called three times with different arguments to demonstrate method overloading.

Output:



```
Sum of integers: 15
Sum of doubles: 6.2
Concatenation of strings: Hello World
```

In this sample output, the `add` method is called three times with different arguments.

- In the first call, the `add` method with integer parameters is invoked, and the sum of 5 and 10 is printed.

- In the second call, the `add` method with double parameters is invoked, and the sum of 3.7 and 2.5 is printed.

- In the third call, the `add` method with string parameters is invoked, and the concatenation of "Hello" and "World" is printed.

POST EXPERIMENT QUESTIONS:

1. Explain the structure of the **MethodOverloadingExample** Java program.
2. What is the purpose of the **add** method in the program?

LAB EXPERIMENT 3

OBJECTIVE:

Write a java program to implement method riding.

PRE-EXPERIMENT QUESTIONS:

1. What is method overriding in Java?
2. What are the requirements for method overriding?

BRIEF DISCUSSION AND EXPLANATION:

```
class Vehicle {

    public void sound() {

        System.out.println("Vehicle makes a sound");

    }

}



class Car extends Vehicle {

    @Override

    public void sound() {

        System.out.println("Car goes vroom");

    }

}
```

```
class Motorcycle extends Vehicle {
    @Override
    public void sound() {
        System.out.println("Motorcycle goes vroom vroom");
    }
}


public class MethodOverridingExample {
    public static void main(String[] args) {
        Vehicle vehicle1 = new Vehicle();
        Vehicle vehicle2 = new Car();
        Vehicle vehicle3 = new Motorcycle();
        vehicle1.sound(); // Output: Vehicle makes a sound
        vehicle2.sound(); // Output: Car goes vroom
        vehicle3.sound(); // Output: Motorcycle goes vroom vroom
    }
}
```

Discussion:

1. The program defines a base class `Vehicle` with a `sound` method that prints "Vehicle makes a sound".

2. The classes `Car` and `Motorcycle` inherit from the `Vehicle` class and override the `sound` method with their own implementations.

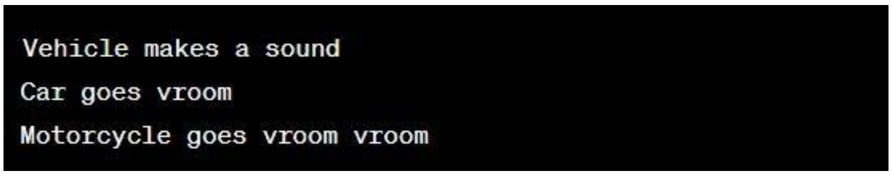
3. The `Car` class overrides the `sound` method to print "Car goes vroom" instead of the default message.

4. The `Motorcycle` class overrides the `sound` method to print "Motorcycle goes vroom vroom" instead of the default message.

5. In the `main` method, objects of `Vehicle`, `Car`, and `Motorcycle` classes are created.

6. The `sound` method is called on each object, and the output demonstrates the method overriding behavior.

Sample Output:



```
Vehicle makes a sound
Car goes vroom
Motorcycle goes vroom vroom
```

In this sample output, the `sound` method is called on objects of the `Vehicle`, `Car`, and `Motorcycle` classes.

- When `sound` is called on the `Vehicle` object, the base class implementation is invoked, and the default message "Vehicle makes a sound" is printed.

- When `sound` is called on the `Car` object, the overridden method in the `Car` class is invoked, and "Car goes vroom" is printed.

- When `sound` is called on the `Motorcycle` object, the overridden method in the `Motorcycle` class is invoked, and "Motorcycle goes vroom vroom" is printed.

This demonstrates method overriding, where the behavior of the method is determined by the actual type of the object at runtime.

POST EXPERIMENT QUESTIONS:

1. What is the output of the program? Explain the behavior of the overridden methods.
2. How can you modify the program to include additional subclasses that override the **makeSound** method?

LAB EXPERIMENT 4

OBJECTIVE:

Write a java program to solve Fibonacci series using recursive method.

PRE-EXPERIMENT QUESTIONS:

1. What is the Fibonacci series?
2. How can you solve the Fibonacci series using a recursive approach?

BRIEF DISCUSSION AND EXPLANATION:

```
public class FibonacciRecursive {

    public static void main(String[] args) {

        int n = 10; // Number of Fibonacci numbers to generate

        System.out.println("Fibonacci Series:");

        for (int i = 0; i < n; i++) {

            System.out.print(fibonacci(i) + " ");

        }

    }



    public static int fibonacci(int n) {

        if (n <= 1) {

            return n;

        }
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Discussion:

1. The program defines a class named `FibonacciRecursive`.

2. In the `main` method, an integer variable `n` is initialized with the number of Fibonacci numbers to generate. In this example, we generate 10 Fibonacci numbers.

3. The program uses a `for` loop to iterate `n` times and print the Fibonacci numbers.

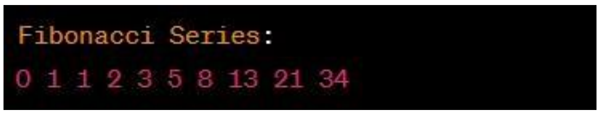
4. Inside the loop, the `fibonacci` method is called with the current index `i` as an argument, and the result is printed.

5. The `fibonacci` method is implemented using recursion. It takes an integer `n` as an argument and returns the `n`th Fibonacci number.

6. In the recursive implementation, if `n` is less than or equal to 1, the method returns `n` as it is (base case).

7. Otherwise, the method calls itself recursively with `n - 1` and `n - 2` as arguments, and adds the results of the two recursive calls to calculate the Fibonacci number for `n`.

Sample Output:



```
Fibonacci Series:
0 1 1 2 3 5 8 13 21 34
```

In this sample output, the program generates the first 10 Fibonacci numbers using recursion.

- The `main` method initializes `n` as 10, indicating that we want to generate 10 Fibonacci numbers.

- The program enters a loop that iterates from 0 to 9 (inclusive).

- For each iteration, the `fibonacci` method is called with the current index as an argument.

- The Fibonacci numbers are calculated recursively using the formula `fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)`.

- The Fibonacci numbers are printed sequentially, separated by spaces.

The output shows the generated Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.

POST EXPERIMENT QUESTIONS:

1. Explain the structure of the **FibonacciSeries** Java program.
2. What is the purpose of the **fibonacci** method in the program?

LAB EXPERIMENT 5

OBJECTIVE:

Write a java program to create a class circle and initialize and display its variables center and radius.

PRE-EXPERIMENT QUESTIONS:

1. What is a class in Java?
2. What are instance variables? How are they defined and used in Java classes?

BRIEF DISCUSSION AND EXPLANATION:

```
class Circle {

   private double centerX;

   private double centerY;

   private double radius;

   public Circle(double centerX, double centerY, double radius) {

      this.centerX = centerX;

      this.centerY = centerY;

      this.radius = radius;

   }



   public void display() {

      System.out.println("Center: (" + centerX + ", " + centerY + ")");
```

```
        System.out.println("Radius: " + radius);
    }
}

public class CircleExample {
    public static void main(String[] args) {
        Circle circle = new Circle(3.0, 4.0, 2.5);
        circle.display();
    }
}
```

Discussion:

1. The program defines a class named `Circle` to represent a circle.

2. The `Circle` class has three private instance variables: `centerX` and `centerY` for the coordinates of the center and `radius` for the radius of the circle.

3. The `Circle` class has a constructor that takes the center coordinates (`centerX` and `centerY`) and the radius as parameters to initialize the object's variables.

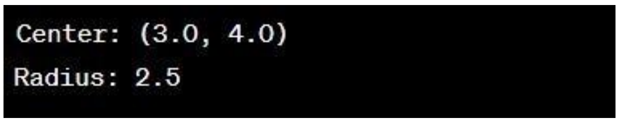
4. The `display` method in the `Circle` class prints the values of the center and radius.

5. The `CircleExample` class contains the `main` method where the program is executed.

6. Inside the `main` method, a `Circle` object named `circle` is created with center coordinates (3.0, 4.0) and a radius of 2.5.

7. The `display` method is called on the `circle` object to print the values of the center and radius.

Sample Output:



```
Center: (3.0, 4.0)
Radius: 2.5
```

In this sample output, the program creates a `Circle` object with center coordinates (3.0, 4.0) and a radius of 2.5. Then, the `display` method is called on the `circle` object, which prints the values of the center and radius. The output shows the center as (3.0, 4.0) and the radius as 2.5.

POST EXPERIMENT QUESTIONS:

1. How can you modify the program to add additional methods or variables to the **Circle** class?

2. What is the scope of the **center** and **radius** variables in the **Circle** class?

LAB EXPERIMENT 6

OBJECTIVE:

Write a java program to implement parameterized constructor.

PRE-EXPERIMENT QUESTIONS:

1. What is a constructor in Java?
2. What is the purpose of a parameterized constructor?

BRIEF DISCUSSION AND EXPLANATION:

```
class Person {

   private String name;

   private int age;

   public Person(String name, int age) {

      this.name = name;

      this.age = age;

   }



   public void display() {

      System.out.println("Name: " + name);

      System.out.println("Age: " + age);

   }

}
```

```
public class ParameterizedConstructorExample {
    public static void main(String[] args) {
        Person person = new Person("John Doe", 30);
        person.display();
    }
}
```

Discussion:

1. The program defines a class named `Person` to represent a person.

2. The `Person` class has two private instance variables: `name` to store the person's name and `age` to store the person's age.

3. The `Person` class has a parameterized constructor that takes `name` and `age` as parameters to initialize the object's variables.

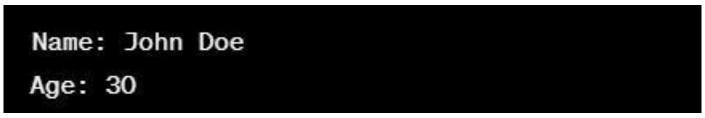
4. The `display` method in the `Person` class prints the values of the name and age.

5. The `ParameterizedConstructorExample` class contains the `main` method where the program is executed.

6. Inside the `main` method, a `Person` object named `person` is created using the parameterized constructor, with the name set as "John Doe" and age as 30.

7. The `display` method is called on the `person` object, which prints the values of the name and age.

Output:



```
Name: John Doe
Age: 30
```

In this sample output, the program creates a `Person` object using the parameterized constructor with the name "John Doe" and age 30. Then, the `display` method is called on the `person` object, which prints the values of the name and age. The output shows the name as "John Doe" and the age as 30.

POST EXPERIMENT QUESTIONS:

1. Explain the structure of the **ParameterizedConstructorExample** Java program.
2. What is the purpose of the **ParameterizedConstructorExample** class?

LAB EXPERIMENT 7

OBJECTIVE:

Write a java program to implement stack operations using array.

PRE-EXPERIMENT QUESTIONS:

1. What is a stack in computer science? How does it work?
2. What are the common operations performed on a stack?

BRIEF DISCUSSION AND EXPLANATION:

```
class Stack {

    private int maxSize;

    private int[] stackArray;

    private int top;



    public Stack(int size) {

        maxSize = size;

        stackArray = new int[maxSize];

        top = -1;

    }

    public void push(int value) {

        if (top == maxSize - 1) {

            System.out.println("Stack is full. Cannot push " + value);
        } else {

            stackArray[++top] = value;

            System.out.println("Pushed " + value + " onto the stack");

        }

    }



    public int pop() {

        if (top == -1) {

            System.out.println("Stack is empty. Cannot pop");

            return -1;

        } else {

            int value = stackArray[top--];

            System.out.println("Popped " + value + " from the stack");

            return value;

        }

    }



    public int peek() {

        if (top == -1) {

            System.out.println("Stack is empty");
            return -1;

        } else {

            int value = stackArray[top];

            System.out.println("Top element of the stack is " + value);

            return value;

        }

    }



    public boolean isEmpty() {

        return top == -1;

    }

}



public class StackExample {

    public static void main(String[] args) {

        Stack stack = new Stack(5);



        stack.push(10);

        stack.push(20);

        stack.push(30);
        stack.peek();

        stack.pop();

        stack.pop();

        stack.pop();

        stack.pop();

        stack.isEmpty();

    }

}
```

Discussion:

1. The program defines a class named `Stack` to implement stack operations using an array.

2. The `Stack` class has private instance variables: `maxSize` to store the maximum capacity of the stack, `stackArray` to store the elements of the stack, and `top` to keep track of the index of the top element.

3. The `Stack` class has a constructor that takes the size as a parameter and initializes the stack with the given size.

4. The `push` method pushes an element onto the stack. It first checks if the stack is full, and if not, increments `top` and assigns the value to `stackArray[top]`.

5. The `pop` method pops the top element from the stack. It checks if the stack is empty, and if not, returns the top element by decrementing `top` and accessing `stackArray[top]`.

6. The `peek` method returns the top element of the stack without removing it. It checks if the stack is empty and returns the top element by accessing `stackArray[top]`.

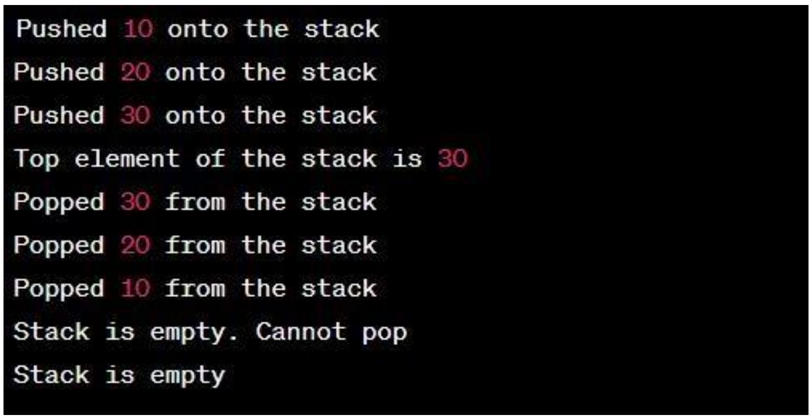
7. The `isEmpty` method checks if the stack is empty by comparing `top` with -1.

8. The `StackExample` class contains the `main` method where the program is executed.

9. Inside the `main` method, a `Stack` object named `stack` is created with a maximum size of 5.

10. Stack operations are performed on the `stack` object, such as pushing elements, peeking the top element, and popping elements.

Output:



```
Pushed 10 onto the stack
Pushed 20 onto the stack
Pushed 30 onto the stack
Top element of the stack is 30
Popped 30 from the stack
Popped 20 from the stack
Popped 10 from the stack
Stack is empty. Cannot pop
Stack is empty
```

In this sample output, the program demonstrates stack operations using an array.

- The elements 10, 20, and 30 are pushed onto the stack using the `push` method.

- The `peek` method is called to retrieve the top element, which is 30.

- The elements are popped from the stack using the `pop` method, starting from the top. The values 30, 20, and 10 are displayed as they are popped.

- After all elements are popped, the `isEmpty` method is called to verify that the stack is empty.

The output shows the results of the various stack operations performed.

POST EXPERIMENT QUESTIONS:

1. What is a stack in computer science? How does it work?
2. What are the common operations performed on a stack?

LAB EXPERIMENT 8

OBJECTIVE:

Write a java program to perform addition of two matrices.

PRE-EXPERIMENT QUESTIONS:

1. What are matrices in mathematics and computer science?
2. How can you represent matrices in Java programs?

BRIEF DISCUSSION AND EXPLANATION:

```
public class MatrixAddition {

    public static void main(String[] args) {

        int[][] matrix1 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

        int[][] matrix2 = { { 9, 8, 7 }, { 6, 5, 4 }, { 3, 2, 1 } };



        int[][] result = addMatrices(matrix1, matrix2);



        displayMatrix(result);

    }

    public static int[][] addMatrices(int[][] matrix1, int[][] matrix2) {

        int rows = matrix1.length;

        int columns = matrix1[0].length;
        int[][] result = new int[rows][columns];


        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < columns; j++) {

                result[i][j] = matrix1[i][j] + matrix2[i][j];

            }

        }


        return result;

    }


    public static void displayMatrix(int[][] matrix) {

        int rows = matrix.length;

        int columns = matrix[0].length;


        System.out.println("Matrix Addition Result:");


        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < columns; j++) {

                System.out.print(matrix[i][j] + " ");
```

~~~~
            }

            System.out.println();

        }

    }

}

```
~~~~

Discussion:

1. The program defines a class named `MatrixAddition` to perform matrix addition.

2. The program uses two-dimensional arrays (`matrix1` and `matrix2`) to represent the matrices to be added.

3. The `main` method initializes the input matrices (`matrix1` and `matrix2`), calls the `addMatrices` method to perform the addition, and displays the result using the `displayMatrix` method.

4. The `addMatrices` method takes two matrices as input and returns the result of their addition.

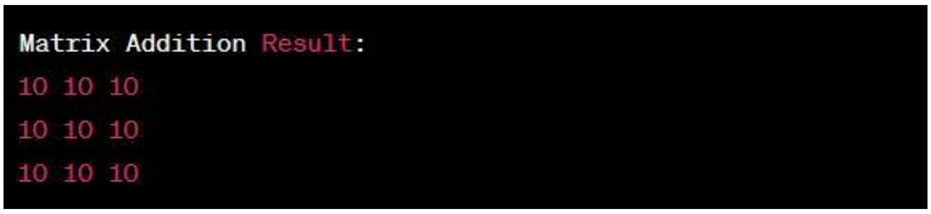
5. The method determines the number of rows and columns in the matrices (`matrix1.length` and `matrix1[0].length`) and creates a new array called `result` to store the sum.

6. The nested `for` loops iterate through each element of the matrices and calculate the sum by adding corresponding elements from `matrix1` and `matrix2`. The sum is stored in the `result` array.

7. The `displayMatrix` method takes a matrix as input and displays its elements row by row.

8. Inside the `displayMatrix` method, the number of rows and columns are determined, and then the elements are printed using nested `for` loops.

Output:



```
Matrix Addition Result:
10 10 10
10 10 10
10 10 10
```

In this sample output, the program performs the addition of two matrices.

- `matrix1` is initialized as a 3x3 matrix with elements 1, 2, 3, 4, 5, 6, 7, 8, and 9.

- `matrix2` is initialized as another 3x3 matrix with elements 9, 8, 7, 6, 5, 4, 3, 2, and 1.

- The `addMatrices` method adds the corresponding elements from `matrix1` and `matrix2` and returns the result as a new matrix called `result`.

- The `displayMatrix` method is called with `result` as an argument, which displays the elements of the resulting matrix.

- The output shows the result of the matrix addition, which is a 3x3 matrix with elements 10, 10, 10, 10, 10, 10, 10, 10, and 10.

POST EXPERIMENT QUESTIONS:

1. Explain the structure of the **MatrixAddition** Java program.
2. How are the input matrices (**matrix1** and **matrix2**) defined and initialized?

LAB EXPERIMENT 9

OBJECTIVE:

Write a java program to implement exception handling.

PRE-EXPERIMENT QUESTIONS:

1. What is exception handling in Java?
2. What is the purpose of using try-catch blocks in exception handling?

BRIEF DISCUSSION AND EXPLANATION:

```
import java.util.InputMismatchException;

import java.util.Scanner;

public class ExceptionHandlingExample {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);



        try {

            System.out.print("Enter an integer: ");

            int num = scanner.nextInt();

            int result = 10 / num;

            System.out.println("Result: " + result);

        } catch (InputMismatchException e) {

            System.out.println("Invalid input! Please enter an integer.");
        } catch (ArithmeticException e) {

            System.out.println("Cannot divide by zero!");

        } finally {

            scanner.close();

            System.out.println("Program execution completed.");

        }

    }

}
```

Discussion:

1. The program imports `InputMismatchException` and `Scanner` classes from the `java.util` package for exception handling and user input.

2. The `main` method prompts the user to enter an integer.

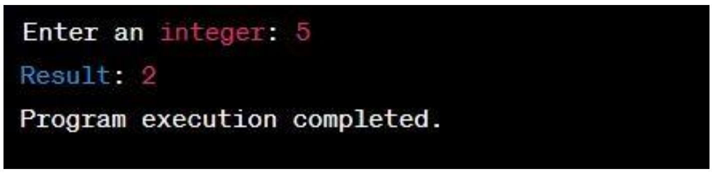
3. Inside the `try` block, the program reads an integer from the user using the `nextInt` method of the `Scanner` class. It then performs a division by the user-provided number and displays the result.

4. If an `InputMismatchException` occurs, it means the user did not enter a valid integer. The program catches this exception and displays an error message.

5. If an `ArithmeticException` occurs due to division by zero, the program catches this exception and displays an appropriate error message.

6. The `finally` block is executed regardless of whether an exception occurred or not. It closes the `Scanner` object and prints a completion message.

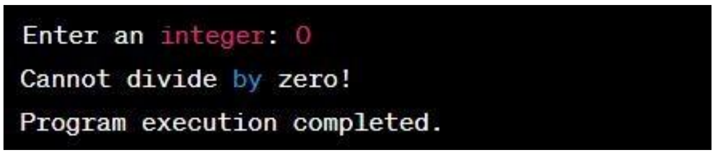
Output 1 (Valid Input - Non-zero Integer):



```
Enter an integer: 5
Result: 2
Program execution completed.
```

In this sample output, the user enters the integer 5. The program successfully performs the division and displays the result as 2. The program execution completes, and the closing message is printed.

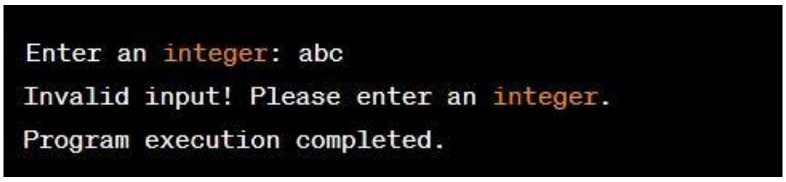
Output 2 (Valid Input - Zero):



```
Enter an integer: 0
Cannot divide by zero!
Program execution completed.
```

In this sample output, the user enters the integer 0. Since division by zero is not possible, an `ArithmeticException` occurs. The program catches the exception, displays an error message, and then completes execution.

Output 3 (Invalid Input - Non-integer):



```
Enter an integer: abc
Invalid input! Please enter an integer.
Program execution completed.
```

In this sample output, the user enters a non-integer input ("abc"). Since it is not a valid integer, an `InputMismatchException` occurs. The program catches the exception, displays an error message, and then completes execution.

Note: The program demonstrates exception handling for specific exceptions (`InputMismatchException` and `ArithmeticException`). Depending on your requirements, you can modify the program to handle other exceptions as needed.

POST EXPERIMENT QUESTIONS:

1. What is the purpose of the **Scanner** object in the Java program?
2. What exceptions are expected to be thrown in the code within the try block? Why?

LAB EXPERIMENT 10

OBJECTIVE:

Write a java program extend thread class.

PRE-EXPERIMENT QUESTIONS:

1. What is multithreading in Java?
2. How can you create a new thread in Java?

BRIEF DISCUSSION AND EXPLANATION:

```
class CustomThread extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Custom Thread: " + i);
            try {
                Thread.sleep(1000); // Sleep for 1 second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
public class ThreadExtensionExample {

    public static void main(String[] args) {

        CustomThread customThread = new CustomThread();

        customThread.start();



        for (int i = 1; i <= 5; i++) {

            System.out.println("Main Thread: " + i);

            try {

                Thread.sleep(1000); // Sleep for 1 second

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

}
```

Discussion:

1. The program defines a class named `CustomThread` that extends the `Thread` class to create a custom thread.

2. The `CustomThread` class overrides the `run` method, which is the entry point for the custom thread's execution.

3. Inside the `run` method, a loop is used to print the numbers from 1 to 5 with a delay of 1 second between each iteration using `Thread.sleep(1000)`.

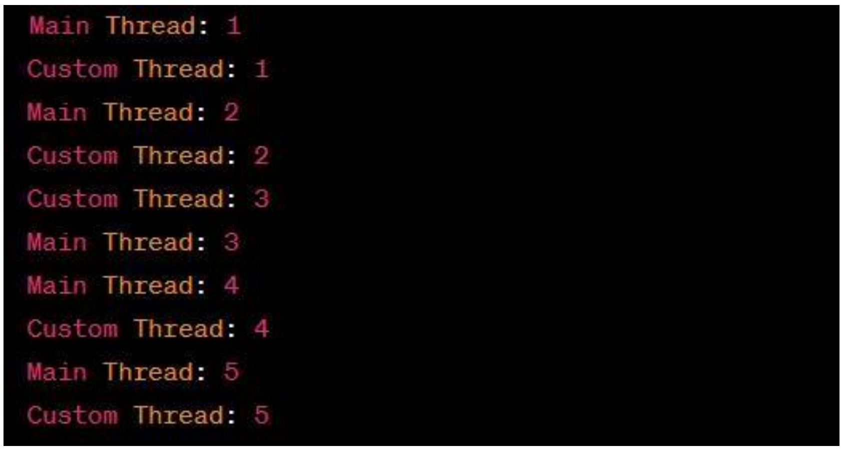
4. The `ThreadExtensionExample` class contains the `main` method where the program is executed.

5. Inside the `main` method, an instance of the `CustomThread` class named `customThread` is created.

6. The `start` method is called on `customThread` to start the execution of the custom thread.

7. Another loop in the `main` method is used to print the numbers from 1 to 5 by the main thread with a delay of 1 second between each iteration.

Output:



```
Main Thread: 1
Custom Thread: 1
Main Thread: 2
Custom Thread: 2
Custom Thread: 3
Main Thread: 3
Main Thread: 4
Custom Thread: 4
Main Thread: 5
Custom Thread: 5
```

In this sample output, the program demonstrates the execution of the custom thread and the main thread.

- The custom thread (`CustomThread`) is created as an instance of the `CustomThread` class and started by calling the `start` method.

- The custom thread runs concurrently with the main thread. Both threads print numbers from 1 to 5 using separate loops.

- The custom thread and the main thread take turns executing their respective code blocks with a delay of 1 second between each iteration.

- The output shows the interleaved execution of the custom thread and the main thread, each printing their respective numbers.

Note: The exact interleaving of the custom thread and the main thread may vary in different program runs due to the nature of thread scheduling.

POST EXPERIMENT QUESTIONS:

1. How is a custom thread created by extending the **Thread** class in the Java program?
2. What is the purpose of the **run** method in the **CustomThread** class?

LAB EXPERIMENT 11

OBJECTIVE:

Write a java program to implement ArrayList using collection framework.

PRE-EXPERIMENT QUESTIONS:

1. What is an ArrayList in Java?
2. What advantages does the ArrayList class offer compared to regular arrays?

BRIEF DISCUSSION AND EXPLANATION:

```
import java.util.ArrayList;

import java.util.List;

public class ArrayListExample {

  public static void main(String[] args) {

    // Create an ArrayList

    List<String> fruits = new ArrayList<>();



    // Add elements to the ArrayList

    fruits.add("Apple");

    fruits.add("Banana");

    fruits.add("Orange");



    // Access elements of the ArrayList
```

```
System.out.println("Fruits: " + fruits);


// Get the size of the ArrayList

int size = fruits.size();

System.out.println("Size: " + size);


// Check if the ArrayList is empty

boolean isEmpty = fruits.isEmpty();

System.out.println("Is Empty? " + isEmpty);


// Retrieve an element by index

String firstFruit = fruits.get(0);

System.out.println("First Fruit: " + firstFruit);


// Update an element at a specific index

fruits.set(1, "Mango");

System.out.println("Updated Fruits: " + fruits);


// Remove an element from the ArrayList

fruits.remove(2);
```

```
        System.out.println("Updated Fruits: " + fruits);

    }

}
```

Discussion:

1. The program imports the necessary classes from the `java.util` package, including `ArrayList` and `List`.

2. The `ArrayListExample` class contains the `main` method where the program is executed.

3. Inside the `main` method, an `ArrayList` named `fruits` is created, which can hold elements of type `String`.

4. Elements ("Apple", "Banana", "Orange") are added to the `fruits` ArrayList using the `add` method.

5. The `println` statement is used to display the contents of the ArrayList.

6. The `size` method is called to retrieve the number of elements in the ArrayList.

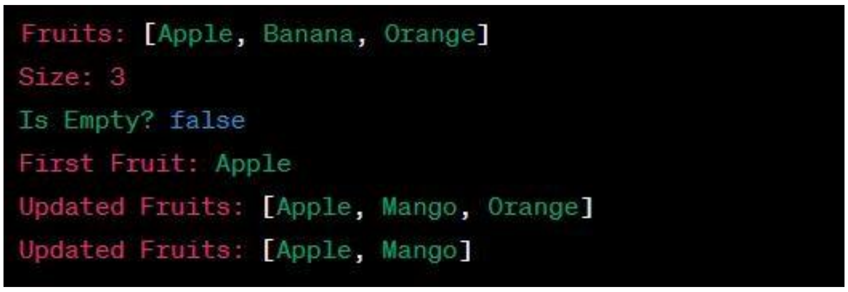
7. The `isEmpty` method is called to check if the ArrayList is empty.

8. The `get` method is used to retrieve an element from the ArrayList based on its index.

9. The `set` method is used to update an element at a specific index in the ArrayList.

10. The `remove` method is used to remove an element from the ArrayList based on its index.

Output:



```
Fruits: [Apple, Banana, Orange]
Size: 3
Is Empty? false
First Fruit: Apple
Updated Fruits: [Apple, Mango, Orange]
Updated Fruits: [Apple, Mango]
```

In this sample output, the program demonstrates the implementation of an ArrayList using the Collection framework.

- Elements ("Apple", "Banana", "Orange") are added to the `fruits` ArrayList using the `add` method.

- The `println` statement displays the contents of the ArrayList, showing all the fruits.

- The `size` method is called to retrieve the number of elements in the ArrayList, which is 3.

- The `isEmpty` method is called to check if the ArrayList is empty, which returns `false`.

- The `get` method is used to retrieve the first fruit ("Apple") from the ArrayList based on its index (0).

- The `set` method is used to update the second fruit from "Banana" to "Mango" at index 1.

- The `remove` method is used to remove the third fruit ("Orange") from the ArrayList at index 2.

The output demonstrates the various ArrayList operations, including adding elements, accessing elements by index, updating elements, and removing elements from the ArrayList.

POST EXPERIMENT QUESTIONS:

1. How is an ArrayList initialized and created in the Java program?
2. How do you add elements to an ArrayList? What method is used?

This lab manual has been updated by

Dr. Ashima Mehta

(ashima.mehta@ggnindia.dronacharya.info)

Crosschecked By

HOD CSE

Please spare some time to provide your valuable feedback.