Practice 1: Learning to work with Java IDE and Writing Simple Conversion Programs

AIM

Write a program that converts temperature from Fahrenheit to Celsius using an IDE.
Formula: **Celsius = (Fahrenheit - 32) × 5/9**

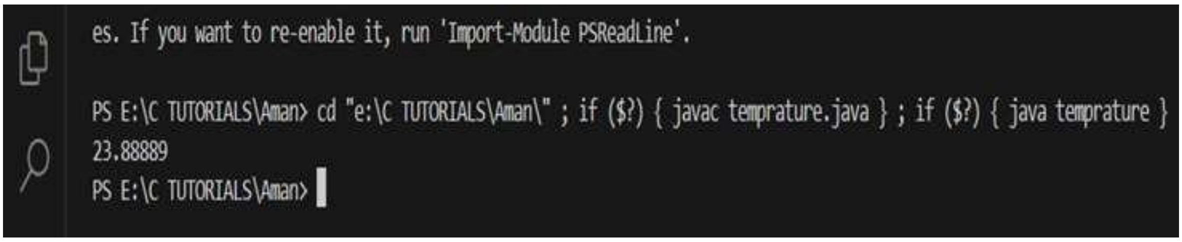
PROCEDURE

1. Start the program.
2. Take Fahrenheit temperature as input from user.
3. Apply formula celsius = (fahrenheit - 32) * 5 / 9.
4. Print the Celsius value.
5. Stop.

CODE

```
public class temprature{
   public static void main(String args[]){
      float f = 75;
      float c;
      c = (f-32)* 5/9;
      System.out.println(c);
   }
}
```

Output:-



```
es. If you want to re-enable it, run 'Import-Module PSReadLine'.

PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac temprature.java } ; if ($?) { java temprature }
23.88889
PS E:\C TUTORIALS\Aman> 
```

Practice 2: Program to implement the Sorting Operation using Control Statements

AIM

Write a Java program to accept 10 integer values from the user, store them in an array, and:
A. Arrange the array in ascending and descending order using Bubble Sort
B. Find the Maximum, Minimum, and Average
C. Print only either Odd or Even numbers

PROCEDURE

1. Start program.
2. Declare an array of size 10.
3. Accept 10 integers from user.
4. Apply Bubble Sort for ascending order:

o Repeat passes for array length.

o Swap adjacent elements if out of order.

5. Print Ascending order.
6. Reverse the array to get Descending order.
7. Find Maximum (last element), Minimum (first element), and Average.
8. Ask user choice (odd/even) and print accordingly.
9. Stop.

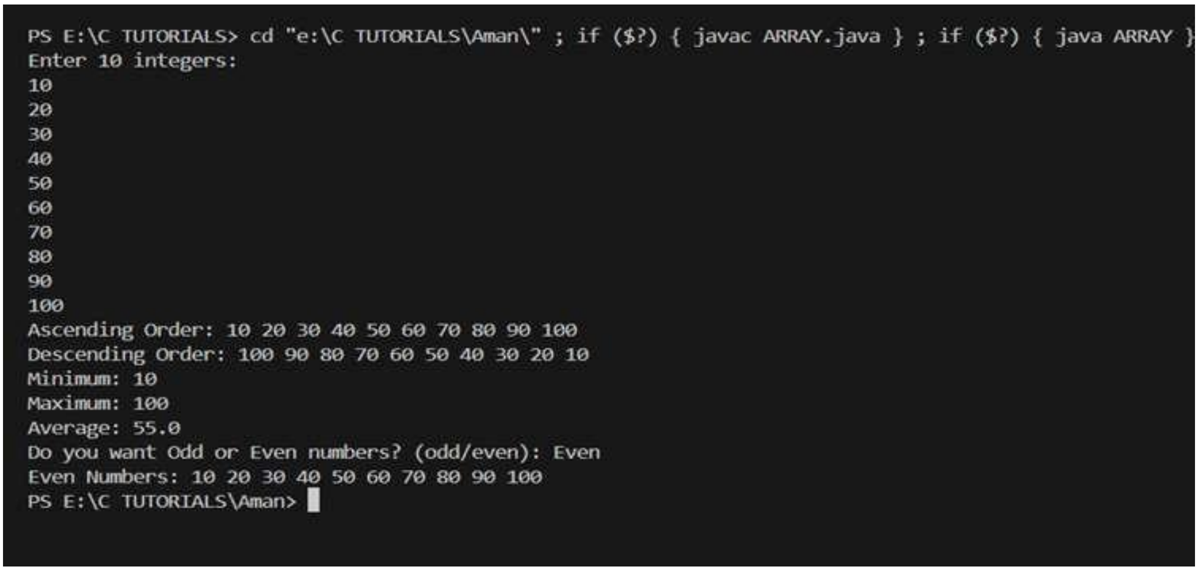
CODE

```
import java.util.Scanner;
 class ARRAY {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int arr[] = new int[10];

      System.out.println("Enter 10 integers: ");
      for (int i = 0; i < 10; i++) {
         arr[i] = sc.nextInt();
      }

      // Bubble Sort for Ascending Order
      for (int i = 0; i < arr.length - 1; i++) {
```

```
    for (int j = 0; j < arr.length - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            // swap
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}
// Print Ascending
System.out.print("Ascending Order: ");
for (int num : arr) {
    System.out.print(num + " ");
}
System.out.println();
// Print Descending
System.out.print("Descending Order: ");
for (int i = arr.length - 1; i >= 0; i--) {
    System.out.print(arr[i] + " ");
}
System.out.println();
// Max, Min, Average
int min = arr[0];
int max = arr[arr.length - 1];
double sum = 0;
for (int num : arr) sum += num;
double avg = sum / arr.length;
System.out.println("Minimum: " + min);
System.out.println("Maximum: " + max);
System.out.println("Average: " + avg);
// Print Odd/Even
```

```
        System.out.print("Do you want Odd or Even numbers? (odd/even): ");
        String choice = sc.next();

        if (choice.equalsIgnoreCase("odd")) {
            System.out.print("Odd Numbers: ");
            for (int num : arr) {
                if (num % 2 != 0) System.out.print(num + " ");
            }
        } else {
            System.out.print("Even Numbers: ");
            for (int num : arr) {
                if (num % 2 == 0) System.out.print(num + " ");
            }
        }
    }
}
```

Output:-



```
PS E:\C TUTORIALS> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac ARRAY.java } ; if ($?) { java ARRAY }
Enter 10 integers:
10
20
30
40
50
60
70
80
90
100
Ascending Order: 10 20 30 40 50 60 70 80 90 100
Descending Order: 100 90 80 70 60 50 40 30 20 10
Minimum: 10
Maximum: 100
Average: 55.0
Do you want Odd or Even numbers? (odd/even): Even
Even Numbers: 10 20 30 40 50 60 70 80 90 100
PS E:\C TUTORIALS\Aman> █
```

Practice 3: Program to implement the Stack operations using Array

AIM

To implement the fundamental Stack operations (**Push, Pop, Peek, and Display**) using an array in Java.

PROCEDURE

1. Start the program.

2. Create a class StackArray with methods:

o **push()** → insert element at top

o **pop()** → remove element from top

o **peek()** → show top element

o **display()** → print all stack elements

3. Maintain top variable to track last inserted element.

4. Use menu-driven program for operations.

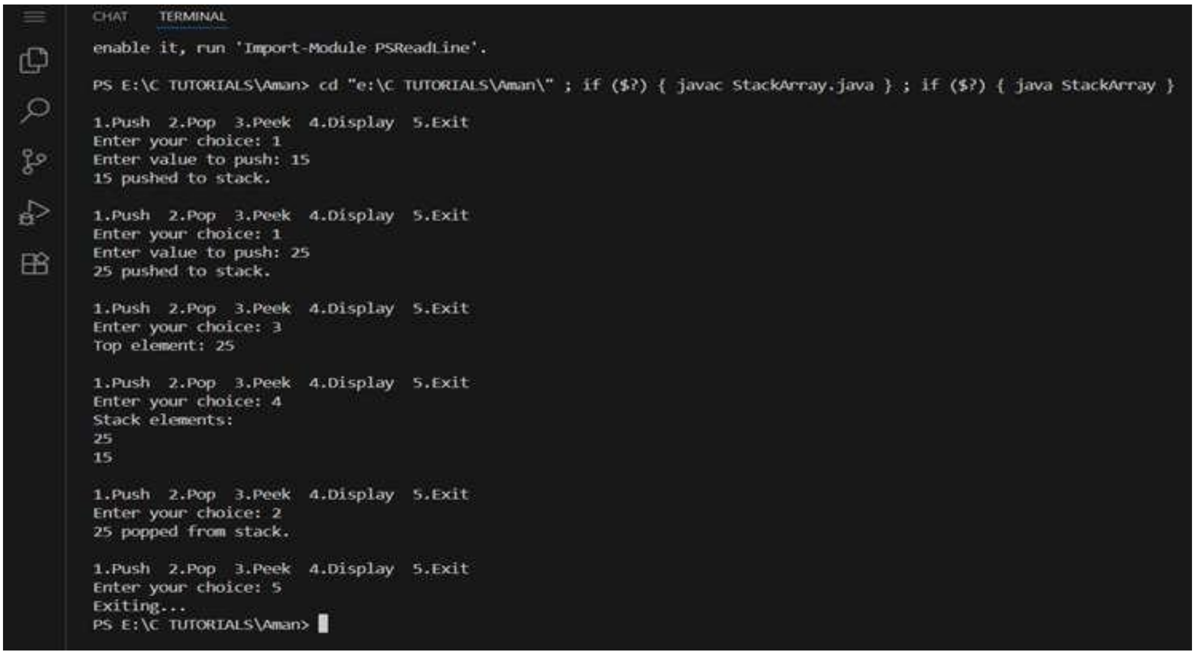
5. Stop.

CODE

```
import java.util.Scanner;

public class StackArray {
    int top;
    int maxSize;
    int[] stack;

    // Constructor
    StackArray(int size) {
        maxSize = size;
        stack = new int[maxSize];
        top = -1; // Stack is empty
    }

    // Push operation
```

```
void push(int value) {
    if (top == maxSize - 1) {
        System.out.println("Stack Overflow!");
    } else {
        stack[++top] = value;
        System.out.println(value + " pushed to stack.");
    }
}

// Pop operation
void pop() {
    if (top == -1) {
        System.out.println("Stack Underflow!");
    } else {
        System.out.println(stack[top--] + " popped from stack.");
    }
}

// Peek operation
void peek() {
    if (top == -1) {
        System.out.println("Stack is empty!");
    } else {
        System.out.println("Top element: " + stack[top]);
    }
}

// Display operation
void display() {
    if (top == -1) {
        System.out.println("Stack is empty!");
    } else {
        System.out.println("Stack elements:");
        for (int i = top; i >= 0; i--) {
            System.out.println(stack[i]);
        }
    }
}
// Main method
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    StackArray stack = new StackArray(5); // stack of size 5

    int choice;
    do {
        System.out.println("\n1.Push  2.Pop  3.Peek  4.Display  5.Exit");
        System.out.print("Enter your choice: ");
        choice = sc.nextInt();

        switch (choice) {
            case 1:
                System.out.print("Enter value to push: ");
                int val = sc.nextInt();
                stack.push(val);
                break;
            case 2:
                stack.pop();
                break;
            case 3:
                stack.peek();
                break;
            case 4:
                stack.display();
                break;
            case 5:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice!");
        }
    } while (choice != 5);

    sc.close();
  }
}
```

OUTPUT:



```
enable it, run 'Import-Module PSReadLine'.

PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac StackArray.java } ; if ($?) { java StackArray }

1.Push  2.Pop  3.Peek  4.Display  5.Exit
Enter your choice: 1
Enter value to push: 15
15 pushed to stack.

1.Push  2.Pop  3.Peek  4.Display  5.Exit
Enter your choice: 1
Enter value to push: 25
25 pushed to stack.

1.Push  2.Pop  3.Peek  4.Display  5.Exit
Enter your choice: 3
Top element: 25

1.Push  2.Pop  3.Peek  4.Display  5.Exit
Enter your choice: 4
Stack elements:
25
15

1.Push  2.Pop  3.Peek  4.Display  5.Exit
Enter your choice: 2
25 popped from stack.

1.Push  2.Pop  3.Peek  4.Display  5.Exit
Enter your choice: 5
Exiting...
PS E:\C TUTORIALS\Aman> 
```

Practice 4: Program to implement the Queue operations using Classes and Objects

AIM

To implement the basic Queue operations (**Enqueue, Dequeue, Peek, and Display**) using classes and objects in Java.

PROCEDURE

1. Start program.
2. Create a class QueueArray with methods:

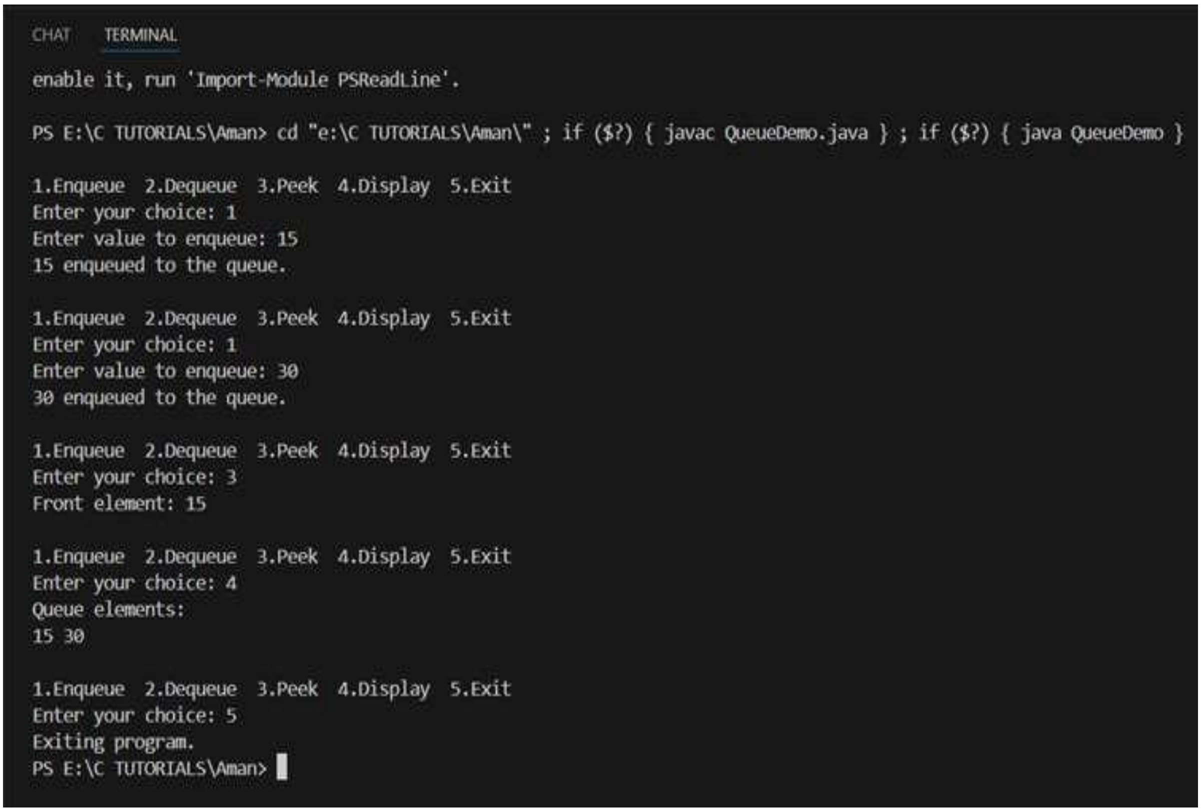
- **enqueue()** → add element at rear
- **dequeue()** → remove element from front
- **peek()** → show front element
- **display()** → print all elements

3. Maintain front and rear pointers.
4. Use menu-driven approach.
5. Stop.

CODE

```
class Queue {
  int front, rear, size;
  int[] queue;

  // Constructor
  Queue(int capacity) {
    size = capacity;
    queue = new int[size];
    front = 0;
    rear = -1;
  }

  // Enqueue operation
  void enqueue(int value) {
    if (rear == size - 1) {
        System.out.println("Queue is full (Overflow)!");
    } else {
        rear++;
        queue[rear] = value;
        System.out.println(value + " enqueued to the queue.");
    }
}

// Deque operation
void dequeue() {
    if (front > rear) {
        System.out.println("Queue is empty (Underflow)!");
    } else {
        System.out.println(queue[front] + " dequeued from the queue.");
        front++;
    }
}

// Peek operation
void peek() {
    if (front > rear) {
        System.out.println("Queue is empty!");
    } else {
        System.out.println("Front element: " + queue[front]);
    }
}

// Display operation
void display() {
    if (front > rear) {
            System.out.println("Queue is empty!");
        } else {
            System.out.println("Queue elements:");
            for (int i = front; i <= rear; i++) {
                System.out.print(queue[i] + " ");
            }
            System.out.println();
        }
    }
}

public class QueueDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Queue q = new Queue(5); // Creating queue of size 5

        int choice;
        do {
            System.out.println("\n1.Enqueue  2.Dequeue  3.Peek  4.Display  5.Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter value to enqueue: ");
                    int val = sc.nextInt();
                    q.enqueue(val);
                    break;
                case 2:
                    q.dequeue();
                    break;
                case 3:
                    q.peek();
                    break;
                case 4:
                    q.display();
                    break;
                case 5:
                    System.out.println("Exiting program.");
                    break;
                default:
                    System.out.println("Invalid choice!");
            }
        } while (choice != 5);

        sc.close();
    }
}
```

OUTPUT:-



```
enable it, run 'Import-Module PSReadLine'.

PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac QueueDemo.java } ; if ($?) { java QueueDemo }

1.Enqueue  2.Dequeue  3.Peek  4.Display  5.Exit
Enter your choice: 1
Enter value to enqueue: 15
15 enqueued to the queue.

1.Enqueue  2.Dequeue  3.Peek  4.Display  5.Exit
Enter your choice: 1
Enter value to enqueue: 30
30 enqueued to the queue.

1.Enqueue  2.Dequeue  3.Peek  4.Display  5.Exit
Enter your choice: 3
Front element: 15

1.Enqueue  2.Dequeue  3.Peek  4.Display  5.Exit
Enter your choice: 4
Queue elements:
15 30

1.Enqueue  2.Dequeue  3.Peek  4.Display  5.Exit
Enter your choice: 5
Exiting program.
PS E:\C TUTORIALS\Aman> 
```

AIM

Write a Java program to create a Calculator. Use classes and methods to perform addition (+), subtraction (−), multiplication (×), division (÷), and modulus (%).

PROCEDURE

1. Start the program.

2. Create a class Calculator with methods:

o add(a, b) → returns sum

o subtract(a, b) → returns difference

o multiply(a, b) → returns product

o divide(a, b) → returns quotient

o modulus(a, b) → returns remainder

3. In main(), accept two numbers from user.

4. Ask user to choose operation.

5. Call respective method and print result.

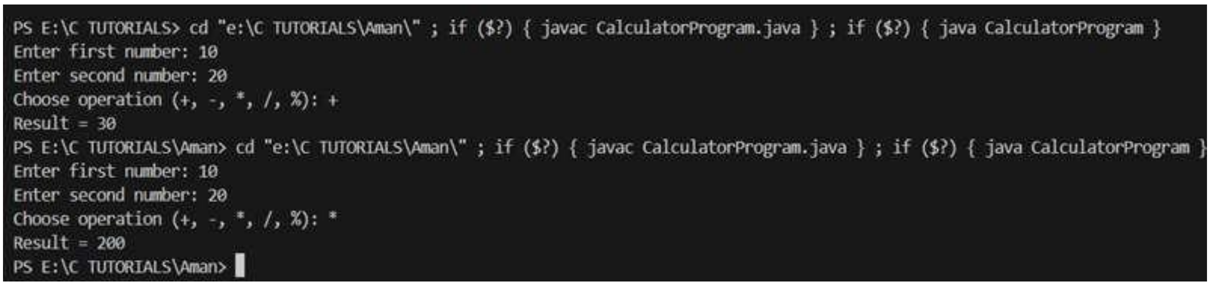
6. Stop.

CODE

```
import java.util.Scanner;

// Calculator class
class Calculator {
   // method for addition
   public int add(int a, int b) {
      return a + b;
   }

   // method for subtraction
   public int subtract(int a, int b) {
      return a - b;
   }
```

```
    // method for multiplication
    public int multiply(int a, int b) {
        return a * b;
    }

    // method for division
    public double divide(int a, int b) {
        if (b == 0) {
            System.out.println("Error: Division by zero!");
            return 0;
        }
        return (double) a / b;
    }

    // method for modulus
    public int modulus(int a, int b) {
        if (b == 0) {
            System.out.println("Error: Modulus by zero!");
            return 0;
        }
        return a % b;
    }
}

// Main class
public class CalculatorProgram {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Calculator calc = new Calculator();

        System.out.print("Enter first number: ");
        int num1 = sc.nextInt();

        System.out.print("Enter second number: ");
        int num2 = sc.nextInt();

        System.out.print("Choose operation (+, -, *, /, %): ");
        char operation = sc.next().charAt(0);

        switch (operation) {
            case '+':
                System.out.println("Result = " + calc.add(num1, num2));
                break;
            case '-':
                System.out.println("Result = " + calc.subtract(num1, num2));
                break;
            case '*':
                System.out.println("Result = " + calc.multiply(num1, num2));
                break;
            case '/':
                System.out.println("Result = " + calc.divide(num1, num2));
                break;
            case '%':
                System.out.println("Result = " + calc.modulus(num1, num2));
                break;
            default:
                System.out.println("Invalid operation!");
        }

    }
}
```

OUTPUT



```
PS E:\C TUTORIALS> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac CalculatorProgram.java } ; if ($?) { java CalculatorProgram }
Enter first number: 10
Enter second number: 20
Choose operation (+, -, *, /, %): +
Result = 30
PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac CalculatorProgram.java } ; if ($?) { java CalculatorProgram }
Enter first number: 10
Enter second number: 20
Choose operation (+, -, *, /, %): *
Result = 200
PS E:\C TUTORIALS\Aman> █
```

AIM(i)

Write a Java program to check whether the input number is part of the Fibonacci series or not, and print the Fibonacci series till that point.

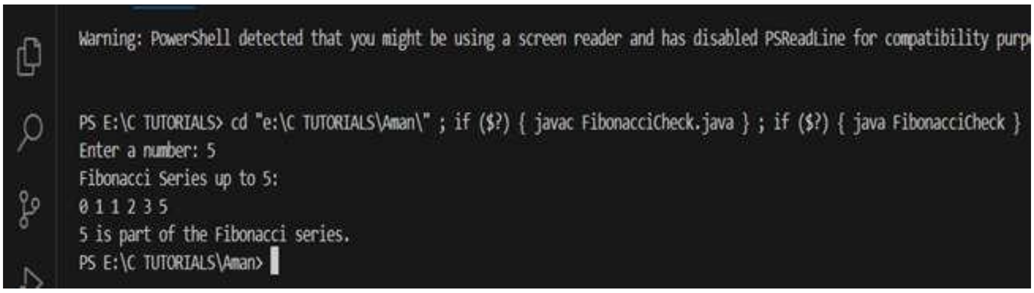
PROCEDURE

1. Start the program.
2. Take a number n from user.
3. Generate Fibonacci series (0, 1, 1, 2, 3, 5 …) until value ≥ n.
4. Print the series.
5. If any term equals n, then n is part of Fibonacci series, else not.
6. Stop.

CODE

```
import java.util.Scanner;

class FibonacciCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        int a = 0, b = 1;
        boolean isFibonacci = false;

        System.out.println("Fibonacci Series up to " + num + ":");

        // Print Fibonacci series till the number
        while (a <= num) {
            System.out.print(a + " ");
            if (a == num) {
                isFibonacci = true;
            }
            int c = a + b;

            a = b;

            b = c;

        }


        System.out.println();


        // Check result

        if (isFibonacci) {

            System.out.println(num + " is part of the Fibonacci series.");

        } else {

            System.out.println(num + " is NOT part of the Fibonacci series.");

        }

    }

}
```

OUTPUT



```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purp


PS E:\C TUTORIALS> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac FibonacciCheck.java } ; if ($?) { java FibonacciCheck }
Enter a number: 5
Fibonacci Series up to 5:
0 1 1 2 3 5
5 is part of the Fibonacci series.
PS E:\C TUTORIALS\Aman> 
```

Practice 5 : Implement Tower of Hanoi program using Recursion

AIM : Write a java program to implement Tower of Hanoi program using Recursion.

Tower of Hanoi using Recursion in Java

three simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
3. No disk may be placed on top of a smaller disk.

Move n-1 disks from the source to the auxiliary .

Move the largest disk (n-th disk) from the source to the destination .

Move the n-1 disks from the auxiliary to the destination.

```
 public class TowerOfHanoi {

 public void solve(int n, char source, char auxiliary, char destination) {

if (n == 1) {        //If there is only one disk, move it from source to destination.

        System.out.println("Move disk 1 from " + source + " to " + destination);

        return;

    }

    solve(n - 1, source, destination, auxiliary);  // Recursive step 1: Move n-1 disks from source to auxiliary

    // Recursive step 2: Move the n-th disk from source to destination.

    System.out.println("Move disk " + n + " from " + source + " to " + destination);


    // Recursive step 3: Move the n-1 disks from auxiliary to destination.

    solve(n - 1, auxiliary, source, destination);

  }

  public static void main(String[] args) {

    int numberOfDisks = 3;

    TowerOfHanoi tower = new TowerOfHanoi();

    System.out.println("Solving Tower of Hanoi with " + numberOfDisks + " disks:");

    tower.solve(numberOfDisks, 'A', 'B', 'C');

  }}
```

Practice 5(A): Implement Overloading Methods, Constructors program.

AIM

- **Inheritance** (one class derives from another).
- **Method Overriding** (child class provides its own version of a parent method).
- **Abstract class & Abstract methods** (base class defines methods without implementation, forcing subclasses to implement them).

Procedure

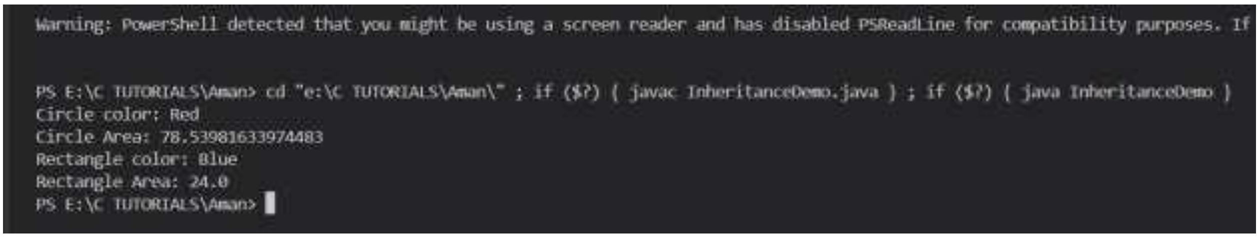
1. Start the program.
2. Define an **abstract class Shape** that contains:
    - A constructor to initialize color.
    - An **abstract method area()**.
    - A normal method displayColor().
3. Create two child classes:
    - Circle (inherits Shape) and implements area() and overrides displayColor().
    - Rectangle (inherits Shape) and implements area() and overrides displayColor().
4. In the main() method:
    - Create objects of Circle and Rectangle using **polymorphism** (Shape reference).
    - Call displayColor() and area() methods.
5. Observe how **method overriding** works when child classes redefine parent methods.
6. Stop the program.

CODE:

```
// Abstract class (cannot be instantiated)

abstract class Shape {

    String color;


    // Constructor

    Shape(String color) {

        this.color = color;

    }
    // Abstract method (must be implemented by subclasses)
    abstract double area();

    // Concrete method (can be overridden)
    void displayColor() {
        System.out.println("Shape color: " + color);
    }
}

// Inherited class (Circle is a Shape)
class Circle extends Shape {
    double radius;

    // Constructor
    Circle(String color, double radius) {
        super(color); // calling parent constructor
        this.radius = radius;
    }

    // Method overriding: providing implementation of abstract method
    @Override
    double area() {
        return Math.PI * radius * radius;
    }

    // Overriding concrete method
    @Override
    void displayColor() {
        System.out.println("Circle color: " + color);
    }
}
```

```
// Another child class (Rectangle is a Shape)
class Rectangle extends Shape {
    double length, width;

    Rectangle(String color, double length, double width) {
        super(color);
        this.length = length;
        this.width = width;
    }

    @Override
    double area() {
        return length * width;
    }

    @Override
    void displayColor() {
        System.out.println("Rectangle color: " + color);
    }
}

// Main class
public class InheritanceDemo {
    public static void main(String[] args) {
        Shape circle = new Circle("Red", 5);
        Shape rectangle = new Rectangle("Blue", 4, 6);

        circle.displayColor();
        System.out.println("Circle Area: " + circle.area());
        rectangle.displayColor();

        System.out.println("Rectangle Area: " + rectangle.area());

    }

}
```

OUTPUT



```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If


PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if ($?) { javac InheritanceDemo.java } ; if ($?) { java InheritanceDemo }
Circle color: Red
Circle Area: 78.53981633974483
Rectangle color: Blue
Rectangle Area: 24.0
PS E:\C TUTORIALS\Aman>
```

Practice 6(i): Program to implement Linked List Concept

Aim:

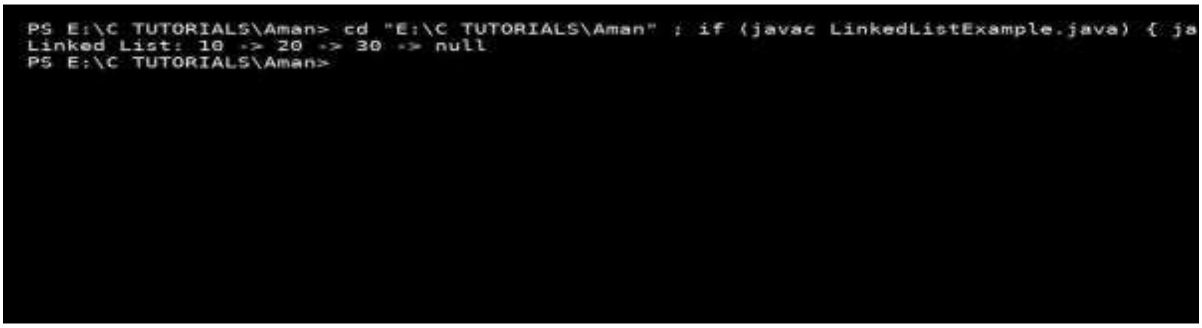
To implement the concept of a **singly linked list** in Java with basic operations like:

- Insertion
- Display

Java Code (Singly Linked List)

```
// Node class
 class Node {
    int data;
    Node next;

    Node(int d) {
        data = d;
        next = null;
    }
}

// LinkedList class
public class LinkedListExample {
    Node head;

    // Insert at the end
    public void insert(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
            return;
        }

        Node current = head;
        while (current.next != null) {
            current = current.next;
        }

        current.next = newNode;
    }

    // Display the list
    public void display() {
        Node current = head;
        System.out.print("Linked List: ");
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }

    // Main method
    public static void main(String[] args) {
        LinkedListExample list = new LinkedListExample();
        list.insert(10);
        list.insert(20);
        list.insert(30);

        list.display();
    }
}
```

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "E:\C TUTORIALS\Aman" ; if (javac LinkedListExample.java) { ja
Linked List: 10 -> 20 -> 30 -> null
PS E:\C TUTORIALS\Aman>
```

Practice 6(ii).Program to implement String Class

Aim:

To demonstrate the use of **Java String class** and its methods like:

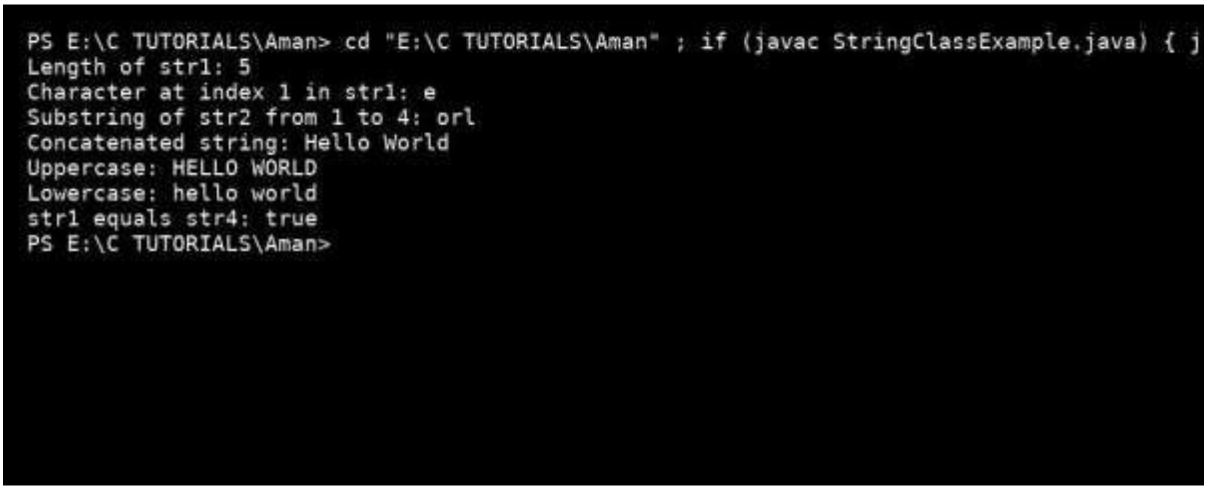
- length()
- charAt()
- substring()
- equals()
- toUpperCase() / toLowerCase()
- concat()

Code:

```
public class StringClassExample {
  public static void main(String[] args) {
    String str1 = "Hello";
    String str2 = "World";

    // Length of string
    System.out.println("Length of str1: " + str1.length());

    // Character at position
    System.out.println("Character at index 1 in str1: " + str1.charAt(1));

    // Substring
    System.out.println("Substring of str2 from 1 to 4: " + str2.substring(1, 4));

    // Concatenation
    String str3 = str1.concat(" ").concat(str2);
    System.out.println("Concatenated string: " + str3);

    // Uppercase and Lowercase
    System.out.println("Uppercase: " + str3.toUpperCase());
    System.out.println("Lowercase: " + str3.toLowerCase());
```

```
        // Equals

        String str4 = "Hello";

        System.out.println("str1 equals str4: " + str1.equals(str4));

    }

}
```

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "E:\C TUTORIALS\Aman" ; if (javac StringClassExample.java) { j
Length of str1: 5
Character at index 1 in str1: e
Substring of str2 from 1 to 4: orl
Concatenated string: Hello World
Uppercase: HELLO WORLD
Lowercase: hello world
str1 equals str4: true
PS E:\C TUTORIALS\Aman>
```

Practice 7: Program to Implement Inheritance, Method Overriding, Abstract classes and methods,

Aim: To write a Java program that demonstrates the use of Packages and Interfaces.

Procedure

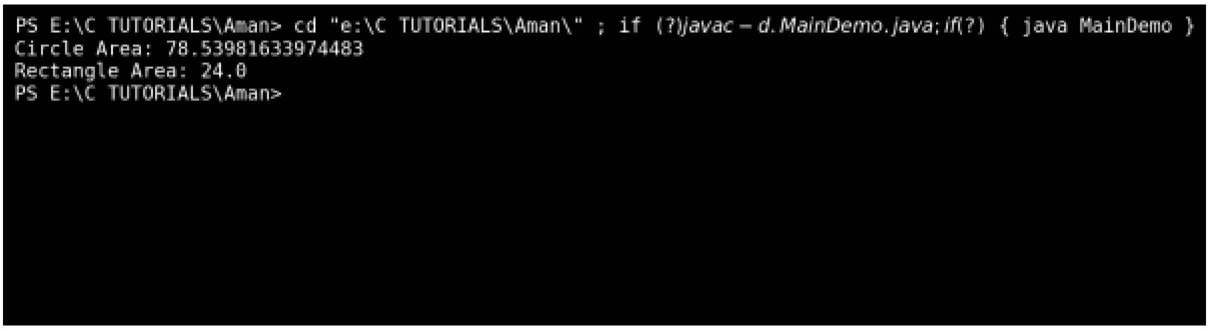
1. Create a package named shapes.
2. Inside the package, create an interface Shape with an abstract method area().
3. Create two classes (Circle, Rectangle) inside the package that implement the interface.
4. In the main class (outside the package), import the package and use the classes.
5. Call the implemented methods and display the results.

CODE:

```
// File: shapes/Shape.java
package shapes;
// Interface declaration
public interface Shape {
    double area();   // abstract method
}
// File: shapes/Circle.java
package shapes;
public class Circle implements Shape {
    double radius;

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

    // Implementing area() method
    public double area() {
        return Math.PI * radius * radius;
    }
}

// File: shapes/Rectangle.java
```

```
package shapes;

public class Rectangle implements Shape {
    double length, width;

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

    // Implementing area() method
    public double area() {
        return length * width;
    }
}

// File: MainDemo.java
import shapes.*;  // Importing user-defined package

public class MainDemo {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(4, 6);

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

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if (?)javac − d.MainDemo.java;if(?) { java MainDemo }
Circle Area: 78.53981633974483
Rectangle Area: 24.0
PS E:\C TUTORIALS\Aman>
```

Practice 8: Program to implement the concept Packages, Interfaces

Aim

To write a Java program that demonstrates the concept of **Packages** and **Interfaces**.

Procedure

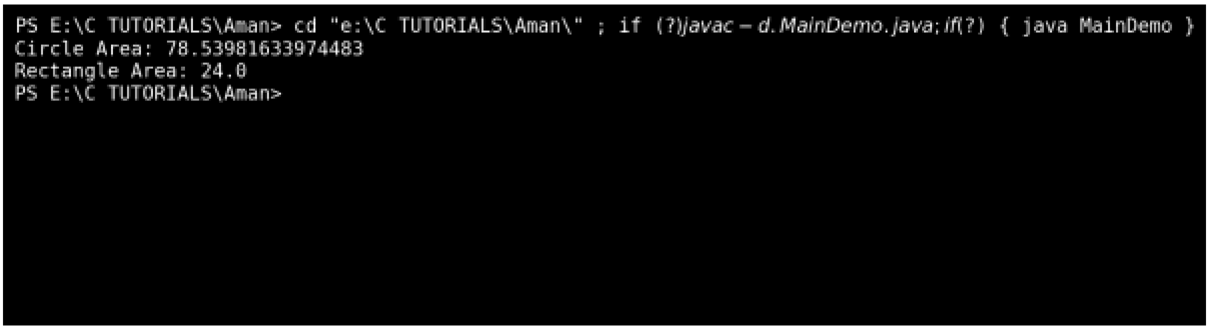
1. Create a **package** named shapes.
2. Inside the package, define an **interface Shape** that declares an abstract method area().
3. Create two classes Circle and Rectangle inside the package that **implement the interface** and provide their own definition for the area() method.
4. Write a main class MainDemo outside the package.
5. In the MainDemo class, **import the package** and create objects of Circle and Rectangle using interface references.
6. Call the area() method for both objects and display the results.
7. Compile the program using javac -d . MainDemo.java and run with java MainDemo.

Code

```
// File: shapes/Shape.java

package shapes;


// Interface declaration

public interface Shape {

   double area();   // abstract method

}


// File: shapes/Circle.java

package shapes;


public class Circle implements Shape {

   double radius;


   public Circle(double radius) {

      this.radius = radius;

   }
```

```
    // Implementing area() method
    public double area() {
        return Math.PI * radius * radius;
    }
}
// File: shapes/Rectangle.java
package shapes;

public class Rectangle implements Shape {
    double length, width;

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

    // Implementing area() method
    public double area() {
        return length * width;
    }
}
// File: MainDemo.java
import shapes.*;  // Importing user-defined package

public class MainDemo {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(4, 6);

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

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if (?)javac − d.MainDemo.java;if(?) { java MainDemo }
Circle Area: 78.53981633974483
Rectangle Area: 24.0
PS E:\C TUTORIALS\Aman>
```

Practice 9: Implement Exception Handling program.

Aim

To write a Java program that demonstrates the concept of **Exception Handling** using try, catch, and finally.

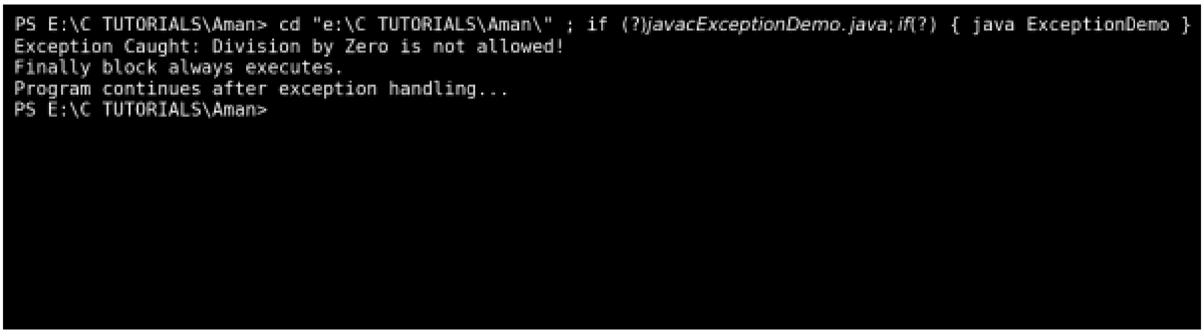
Procedure

1. Start the program.
2. Declare a try block that may throw an exception (e.g., divide by zero).
3. Use catch block(s) to handle specific exceptions like ArithmeticException and general exceptions.
4. Use finally block to execute code that always runs, whether an exception occurs or not.
5. Compile and run the program.
6. Observe the handled exception message and execution of the finally block.

Code

```
public class ExceptionDemo {
    public static void main(String[] args) {
        try {
            int a = 10;
            int b = 0;  // this will cause ArithmeticException
            int result = a / b;  // risky code
            System.out.println("Result: " + result);
        }
        catch (ArithmeticException e) {
            System.out.println("Exception Caught: Division by Zero is not allowed!");
        }
        catch (Exception e) {
            System.out.println("General Exception: " + e.getMessage());
        }
        finally {
            System.out.println("Finally block always executes.");
        }
        System.out.println("Program continues after exception handling...");
    }
}
```

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if (?)javacExceptionDemo.java;if(?) { java ExceptionDemo }
Exception Caught: Division by Zero is not allowed!
Finally block always executes.
Program continues after exception handling...
PS E:\C TUTORIALS\Aman>
```

Practice 10: Implement Multithreading Program

Program to Implement Binary Search Tree

Aim :

- To implement a **Binary Search Tree (BST)** with operations:
  - insert()
  - search()
  - inorderTraversal()
- To use **multithreading** so that insertion, searching, and traversal can run concurrently.

Procedure

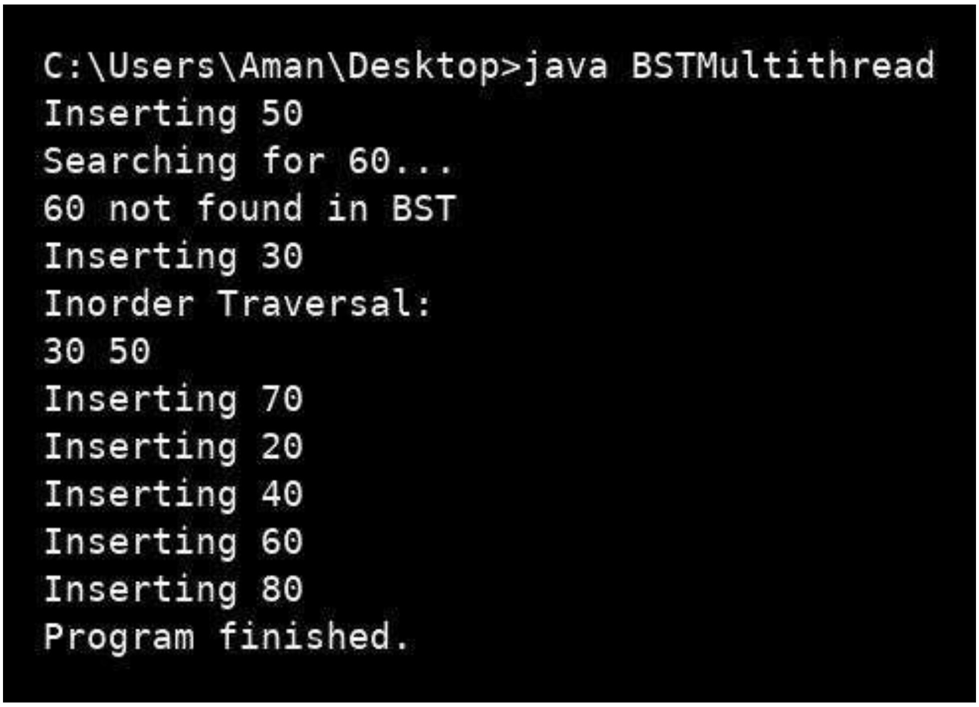
1. Create a Node class with key, left, and right.
2. Create a BST class with synchronized methods:
   - insert(int key) → to insert a node.
   - search(int key) → to search for a node.
   - inorderTraversal() → to print the BST in sorted order.
     Synchronization is used to ensure **thread-safety**.
3. Implement Runnable classes:
   - InsertTask → inserts multiple values.
   - SearchTask → searches for a specific value.
   - TraversalTask → performs inorder traversal.
4. In the main() method, create threads for each task and run them simultaneously.

CODE

```
class Node {
   int key;
   Node left, right;

   public Node(int key) {
      this.key = key;
      left = right = null;
   }
}
```

```java
class BST {
    private Node root;

    // Insert method
    public synchronized void insert(int key) {
        root = insertRec(root, key);
    }

    private Node insertRec(Node root, int key) {
        if (root == null) {
            root = new Node(key);
            return root;
        }
        if (key < root.key) {
            root.left = insertRec(root.left, key);
        } else if (key > root.key) {
            root.right = insertRec(root.right, key);
        }
        return root;
    }

    // Search method
    public synchronized boolean search(int key) {
        return searchRec(root, key);
    }

    private boolean searchRec(Node root, int key) {
        if (root == null) return false;
        if (root.key == key) return true;
        if (key < root.key) return searchRec(root.left, key);
        return searchRec(root.right, key);
    }

    // Inorder traversal
    public synchronized void inorder() {
        inorderRec(root);
        System.out.println();
    }

    private void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.key + " ");
            inorderRec(root.right);
        }
    }
}

// Runnable for insertion
class InsertTask implements Runnable {
    private BST bst;
    private int[] values;

    public InsertTask(BST bst, int[] values) {
        this.bst = bst;
        this.values = values;
    }

    public void run() {
        for (int val : values) {
            System.out.println("Inserting " + val);
            bst.insert(val);
            try { Thread.sleep(500); } catch (InterruptedException e) {}
        }
    }
}

// Runnable for searching
class SearchTask implements Runnable {
    private BST bst;
    private int value;

    public SearchTask(BST bst, int value) {
        this.bst = bst;
        this.value = value;
    }

    public void run() {
        System.out.println("Searching for " + value + "...");
        boolean found = bst.search(value);
        if (found)
            System.out.println(value + " found in BST");
        else
            System.out.println(value + " not found in BST");
    }
}

// Runnable for traversal
class TraversalTask implements Runnable {
    private BST bst;

    public TraversalTask(BST bst) {
        this.bst = bst;
    }

    public void run() {
        try { Thread.sleep(1000); } catch (InterruptedException e) {}
        System.out.println("Inorder Traversal:");
        bst.inorder();
    }
}

public class BSTMultithread {
    public static void main(String[] args) {
        BST bst = new BST();

        int[] values = {50, 30, 70, 20, 40, 60, 80};

        Thread t1 = new Thread(new InsertTask(bst, values));
        Thread t2 = new Thread(new SearchTask(bst, 60));
        Thread t3 = new Thread(new TraversalTask(bst));

        t1.start();
        t2.start();
        t3.start();

        try {
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Program finished.");
    }
}
```

OUTPUT :



```
C:\Users\Aman\Desktop>java BSTMultithread
Inserting 50
Searching for 60...
60 not found in BST
Inserting 30
Inorder Traversal:
30 50
Inserting 70
Inserting 20
Inserting 40
Inserting 60
Inserting 80
Program finished.
```

Practice 11: Program to implement the concept Legacy Classes and Binary Tree Traversal.

Aim:

To understand and implement Legacy Classes in Java (such as Vector, Stack, etc.) and perform Binary Tree Traversals (Inorder, Preorder, Postorder) using these classes.

Procedure:

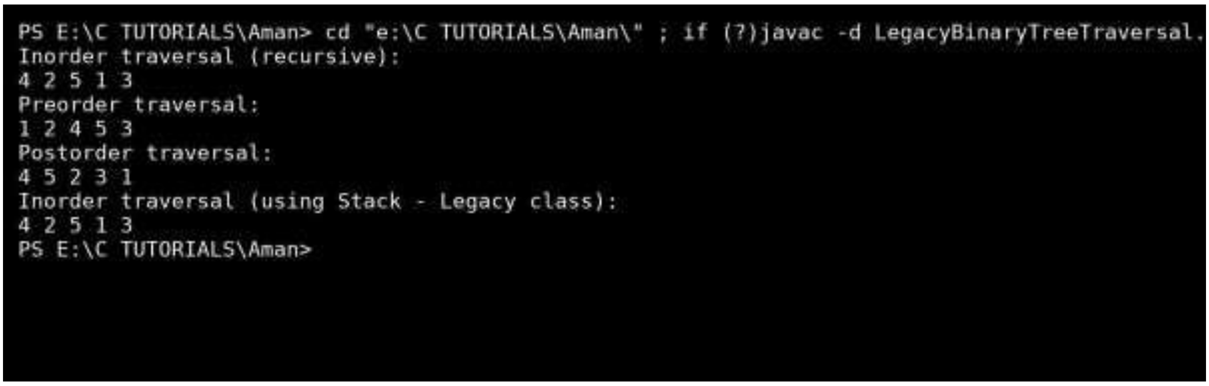
1. **Understand Legacy Classes:**
    - Legacy classes are part of Java’s original 1.0 version and were later retrofitted to implement the Collection Framework.
    - Common legacy classes: Vector, Hashtable, Stack, etc.
2. **Binary Tree Concept:**
    - A binary tree is a data structure where each node has at most two children: left and right.
    - Traversals:
        - Inorder (LNR)
        - Preorder (NLR)
        - Postorder (LRN)
3. **Implementation Steps:**
    - Define a Node class for the binary tree.
    - Create a BinaryTree class with methods for insertion and traversal.
    - Use a legacy class (Stack from java.util) to help in any non-recursive traversal (optional).
    - Write the traversal functions using recursion (or using Stack for iterative version).
    - Create the tree in the main method and call traversal methods to demonstrate functionality.

Code:

```
import java.util.Stack; // Legacy class


// Node class
class Node {
   int data;
```

```
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

// BinaryTree class
class BinaryTree {
    Node root;

    // Inorder traversal (LNR)
    void inorder(Node node) {
        if (node == null)
            return;
        inorder(node.left);
        System.out.print(node.data + " ");
        inorder(node.right);
    }

    // Preorder traversal (NLR)
    void preorder(Node node) {
        if (node == null)
            return;
        System.out.print(node.data + " ");
        preorder(node.left);
        preorder(node.right);
    }

    // Postorder traversal (LRN)
    void postorder(Node node) {
        if (node == null)
            return;
        postorder(node.left);
        postorder(node.right);
        System.out.print(node.data + " ");
    }

    // Optional: Inorder using Legacy Stack (non-recursive)
    void inorderIterative(Node node) {
        Stack<Node> stack = new Stack<>();
        Node current = node;

        while (current != null || !stack.isEmpty()) {
            while (current != null) {
                stack.push(current);
                current = current.left;
            }
            current = stack.pop();
            System.out.print(current.data + " ");
            current = current.right;
        }
    }
}

// Main class
public class LegacyBinaryTreeTraversal {
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        // Create the binary tree
        tree.root = new Node(1);

        tree.root.left = new Node(2);

        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);

        tree.root.left.right = new Node(5);


        System.out.println("Inorder traversal (recursive):");

        tree.inorder(tree.root);


        System.out.println("\nPreorder traversal:");

        tree.preorder(tree.root);


        System.out.println("\nPostorder traversal:");

        tree.postorder(tree.root);


        System.out.println("\nInorder traversal (using Stack - Legacy class):");

        tree.inorderIterative(tree.root);

    }

}
```

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if ($?)javac -d LegacyBinaryTreeTraversal.
Inorder traversal (recursive):
4 2 5 1 3
Preorder traversal:
1 2 4 5 3
Postorder traversal:
4 5 2 3 1
Inorder traversal (using Stack - Legacy class):
4 2 5 1 3
PS E:\C TUTORIALS\Aman>
```

Practice 12: Using Utility Classes implement the program

AIM: Using Utility Classes implement the program Arrays

- Collections
- Scanner
- Date
- Math

Procedure:

1. **Understand Utility Classes**:
   - **java.util.Arrays** – For array manipulation (sorting, searching, etc.)
   - **java.util.Collections** – For collection manipulation (min, max, reverse, sort, etc.)
   - **java.util.Scanner** – For input
   - **java.util.Date / LocalDate / LocalTime** – For working with date/time
   - **java.lang.Math** – For mathematical operations
2. **Design a program** that:
   - Takes user input using Scanner
   - Stores it in a collection or array
   - Sorts and searches using Arrays or Collections
   - Uses Math for some calculations (e.g., square root, power)
   - Displays current date and time using Date or LocalDate

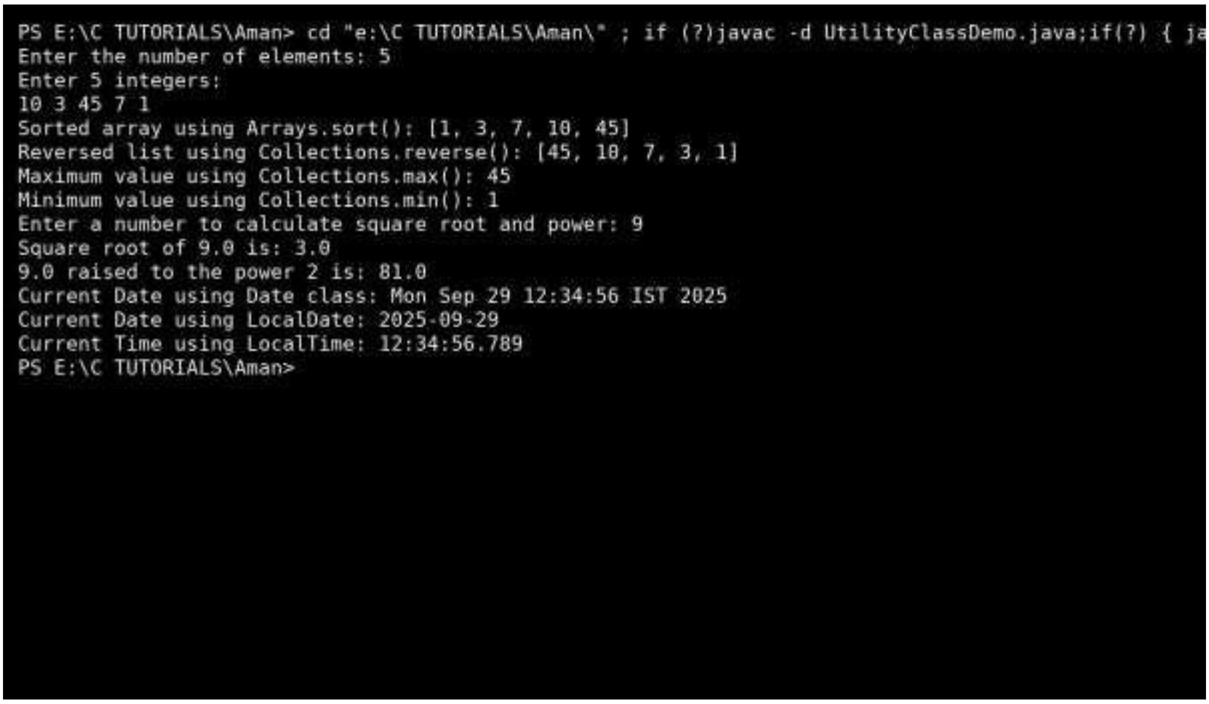
Code:

```
import java.util.*;  // For Scanner, Arrays, Collections, Date
import java.time.*;  // For LocalDate and LocalTime

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

        // 1. Use Scanner to take input
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();
        Integer[] numbers = new Integer[n];
```

```
System.out.println("Enter " + n + " integers:");
for (int i = 0; i < n; i++) {
    numbers[i] = scanner.nextInt();
}

// 2. Use Arrays utility class to sort
Arrays.sort(numbers);
System.out.println("Sorted array using Arrays.sort(): " + Arrays.toString(numbers));

// 3. Use Collections utility class
List<Integer> numberList = Arrays.asList(numbers);
Collections.reverse(numberList);
System.out.println("Reversed list using Collections.reverse(): " + numberList);
System.out.println("Maximum value using Collections.max(): " + Collections.max(numberList));
System.out.println("Minimum value using Collections.min(): " + Collections.min(numberList));

// 4. Use Math utility class
System.out.print("Enter a number to calculate square root and power: ");
double x = scanner.nextDouble();
System.out.println("Square root of " + x + " is: " + Math.sqrt(x));
System.out.println(x + " raised to the power 2 is: " + Math.pow(x, 2));

// 5. Use Date and Time utility classes
Date currentDate = new Date();  // java.util.Date
System.out.println("Current Date using Date class: " + currentDate);

LocalDate localDate = LocalDate.now();
LocalTime localTime = LocalTime.now();
System.out.println("Current Date using LocalDate: " + localDate);
System.out.println("Current Time using LocalTime: " + localTime);
```

```
        scanner.close();
    }
}
```

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "e:\C TUTORIALS\Aman\" ; if (?)javac -d UtilityClassDemo.java;if(?) { ja
Enter the number of elements: 5
Enter 5 integers:
10 3 45 7 1
Sorted array using Arrays.sort(): [1, 3, 7, 10, 45]
Reversed list using Collections.reverse(): [45, 10, 7, 3, 1]
Maximum value using Collections.max(): 45
Minimum value using Collections.min(): 1
Enter a number to calculate square root and power: 9
Square root of 9.0 is: 3.0
9.0 raised to the power 2 is: 81.0
Current Date using Date class: Mon Sep 29 12:34:56 IST 2025
Current Date using LocalDate: 2025-09-29
Current Time using LocalTime: 12:34:56.789
PS E:\C TUTORIALS\Aman>
```

Practice 13: Program to implement Event Handling concepts

Program to implement Graph

Aim:

To build a GUI-based Java application that demonstrates **event handling** through user interaction (e.g., button clicks), and **implements a graph** structure where:

- The user can **add edges** via GUI,
- The graph is stored internally and displayed via output.

Features of the Program:

- Java **Swing GUI**
- Button to **add an edge** between two vertices (entered by user)
- **Event handling** using ActionListener
- **Graph** implemented using **adjacency list**
- Output area to show the current graph structure

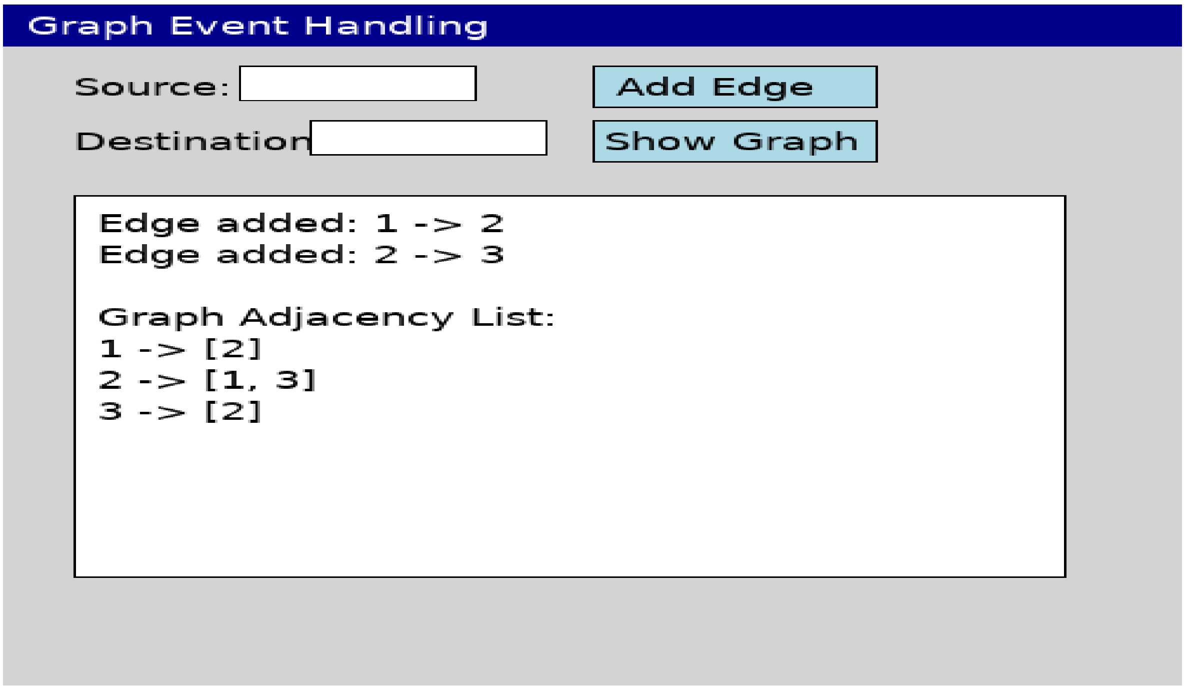
Code:

```
import javax.swing.*;

import java.awt.event.*;

import java.util.*;


public class GraphWithEventHandling extends JFrame implements ActionListener {


  // Components

  private JTextField sourceField, destField;

  private JButton addButton, showButton;

  private JTextArea outputArea;


  // Graph as adjacency list

  private Map<Integer, List<Integer>> graph;


  public GraphWithEventHandling() {

    // Initialize graph

    graph = new HashMap<>();
```

```
// Setup GUI
setTitle("Graph Event Handling");
setSize(400, 400);
setLayout(null);
setDefaultCloseOperation(EXIT_ON_CLOSE);

JLabel sourceLabel = new JLabel("Source:");
sourceLabel.setBounds(30, 30, 60, 25);
add(sourceLabel);

sourceField = new JTextField();
sourceField.setBounds(100, 30, 100, 25);
add(sourceField);

JLabel destLabel = new JLabel("Destination:");
destLabel.setBounds(30, 70, 80, 25);
add(destLabel);

destField = new JTextField();
destField.setBounds(100, 70, 100, 25);
add(destField);

addButton = new JButton("Add Edge");
addButton.setBounds(220, 30, 120, 30);
addButton.addActionListener(this);
add(addButton);

showButton = new JButton("Show Graph");
showButton.setBounds(220, 70, 120, 30);
showButton.addActionListener(this);
```

```
        add(showButton);

        outputArea = new JTextArea();
        outputArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(outputArea);
        scrollPane.setBounds(30, 120, 310, 200);
        add(scrollPane);

        setVisible(true);
    }

    // Event handler
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == addButton) {
            try {
                int src = Integer.parseInt(sourceField.getText());
                int dest = Integer.parseInt(destField.getText());
                addEdge(src, dest);
                outputArea.append("Edge added: " + src + " -> " + dest + "\n");
                sourceField.setText("");
                destField.setText("");
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(this, "Please enter valid integers.");
            }
        }

        if (e.getSource() == showButton) {
            outputArea.append("\nGraph Adjacency List:\n");
            for (int v : graph.keySet()) {
                outputArea.append(v + " -> " + graph.get(v) + "\n");
            }
        }
    }

    // Add edge to the graph (undirected)
    private void addEdge(int src, int dest) {
        graph.putIfAbsent(src, new ArrayList<>());
        graph.putIfAbsent(dest, new ArrayList<>());

        graph.get(src).add(dest);
        graph.get(dest).add(src);
    }

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

OUTPUT



```
Edge added: 1 -> 2
Edge added: 2 -> 3

Graph Adjacency List:
1 -> [2]
2 -> [1, 3]
3 -> [2]
```

Practice 14: AWT Controls, Layout Managers:

To understand and implement basic **AWT Controls** (like Button, Label, TextField, etc.) and organize them using **Layout Managers** (FlowLayout, BorderLayout, GridLayout, etc.)

Procedure:

1. Use **AWT package** to create a GUI window.
2. Add controls like:
   - Label
   - TextField
   - Button
   - Checkbox
   - Choice
3. Use a **Layout Manager** to control the layout of components.
4. Handle events using **Event Listeners** (e.g., ActionListener).

AWT Controls Used:

Control	**Description**
Label	Displays static text
TextField	Accepts single-line user input
Button	Triggers an action
Checkbox	Toggle selection (on/off)
Choice	Dropdown menu
TextArea	Multi-line input/output area

Layout Managers:

Layout Manager	**Description**
FlowLayout	Arranges components in a left-to-right flow
BorderLayout	Divides window into NORTH, SOUTH, EAST, WEST, CENTER
GridLayout	Arranges components in a grid (rows × columns)

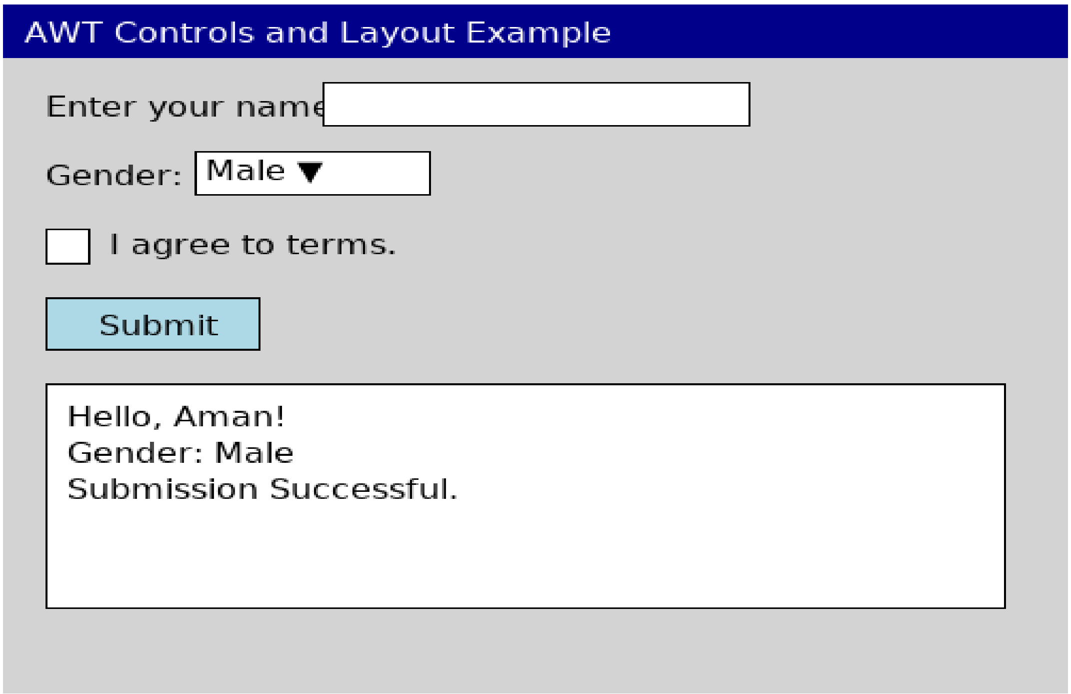
Java Code Example: AWT Controls + Layout Manager

```
import java.awt.*;

import java.awt.event.*;


public class AWTControlExample extends Frame implements ActionListener {
```

```
// AWT Controls
Label nameLabel;
TextField nameField;
Button submitButton;
Checkbox agreeCheckbox;
Choice genderChoice;
TextArea outputArea;

public AWTControlExample() {
  setTitle("AWT Controls and Layout Example");
  setSize(400, 350);
  setLayout(new FlowLayout()); // You can try GridLayout or BorderLayout as well

  // Label
  nameLabel = new Label("Enter your name:");
  add(nameLabel);

  // TextField
  nameField = new TextField(20);
  add(nameField);

  // Choice (Dropdown)
  genderChoice = new Choice();
  genderChoice.add("Male");
  genderChoice.add("Female");
  genderChoice.add("Other");
  add(new Label("Gender:"));
  add(genderChoice);

  // Checkbox
```

```
        agreeCheckbox = new Checkbox("I agree to terms.");
        add(agreeCheckbox);

        // Button
        submitButton = new Button("Submit");
        submitButton.addActionListener(this);
        add(submitButton);

        // TextArea
        outputArea = new TextArea(5, 40);
        add(outputArea);

        setVisible(true);

        // Window close logic
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                dispose();
            }
        });
    }

    // ActionListener method
    public void actionPerformed(ActionEvent e) {
        String name = nameField.getText();
        String gender = genderChoice.getSelectedItem();
        boolean agreed = agreeCheckbox.getState();

        if (agreed) {
            outputArea.setText("Hello, " + name + "!\nGender: " + gender + "\nSubmission Successful.");
        } else {
            outputArea.setText("You must agree to the terms.");
        }
    }

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

OUTPUT



```
Hello, Aman!
Gender: Male
Submission Successful.
```

Practice 15: Implement the program using JDBC Connection

Aim:

To implement a Java program using JDBC to connect to a MySQL database, perform:

- Data insertion
- Data retrieval (SELECT)
- Basic exception handling

Procedure:

- MySQL Database installed and running
- JDBC Driver (MySQL Connector/J) added to your project (JAR file)
- A sample table, e.g., students(id INT, name VARCHAR(50))

Database Setup Example:

```
-- Run in MySQL Workbench or phpMyAdmin
CREATE DATABASE sampledb;
USE sampledb;
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(50)
);
```

Code:

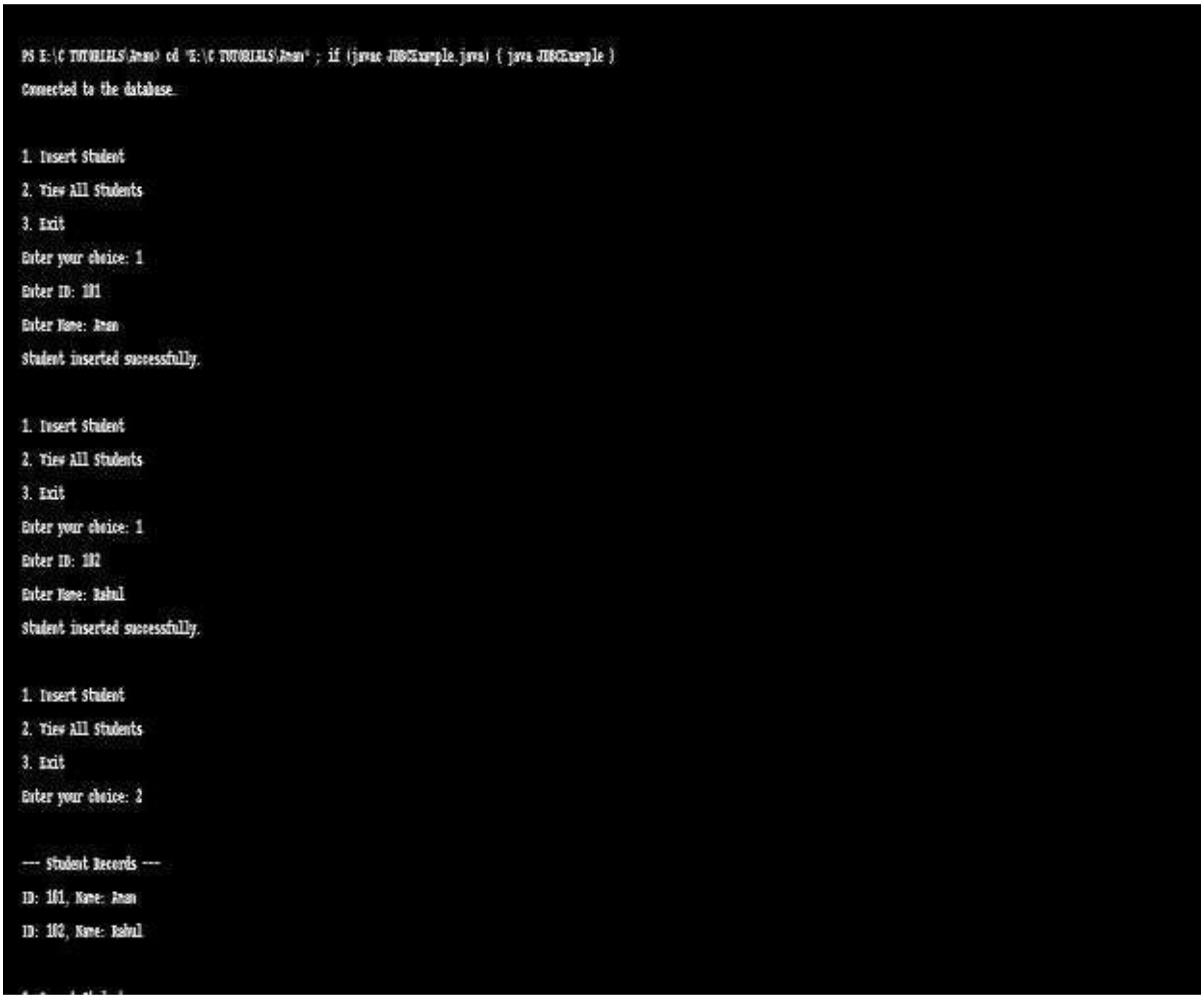
```
import java.sql.*;
import java.util.Scanner;

public class JDBCExample {
  public static void main(String[] args) {
    // Database URL, Username, Password
    String url = "jdbc:mysql://localhost:3306/sampledb"; // Change DB name
    String user = "root"; // Change if needed
    String password = "your_password"; // Change your DB password

    Scanner scanner = new Scanner(System.in);
```

```
try {
  // Load the JDBC driver
  Class.forName("com.mysql.cj.jdbc.Driver");

  // Establish connection
  Connection conn = DriverManager.getConnection(url, user, password);
  System.out.println("Connected to the database.");

  // Menu
  while (true) {
    System.out.println("\n1. Insert Student");
    System.out.println("2. View All Students");
    System.out.println("3. Exit");
    System.out.print("Enter your choice: ");
    int choice = scanner.nextInt();

    if (choice == 1) {
      // Insert
      System.out.print("Enter ID: ");
      int id = scanner.nextInt();
      scanner.nextLine(); // Consume newline
      System.out.print("Enter Name: ");
      String name = scanner.nextLine();

      String insertSQL = "INSERT INTO students (id, name) VALUES (?, ?)";
      PreparedStatement pstmt = conn.prepareStatement(insertSQL);
      pstmt.setInt(1, id);
      pstmt.setString(2, name);
      pstmt.executeUpdate();

      System.out.println("Student inserted successfully.");
```

```
        } else if (choice == 2) {
            // Retrieve
            String selectSQL = "SELECT * FROM students";
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(selectSQL);

            System.out.println("\n--- Student Records ---");
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                System.out.println("ID: " + id + ", Name: " + name);
            }

        } else if (choice == 3) {
            System.out.println("Exiting program.");
            break;
        } else {
            System.out.println("Invalid choice.");
        }
    }

    // Close connection
    conn.close();
    scanner.close();

} catch (ClassNotFoundException e) {
    System.out.println("JDBC Driver not found. Include the MySQL Connector JAR.");
    e.printStackTrace();
} catch (SQLException e) {
    System.out.println("Database error.");
```

```
            e.printStackTrace();
        }
    }
}
```

OUTPUT



```
PS E:\C TUTORIALS\Aman> cd "E:\C TUTORIALS\Aman" ; if (javac JDBCExample.java) { java JDBCExample }
Connected to the database.

1. Insert Student
2. View All Students
3. Exit
Enter your choice: 1
Enter ID: 101
Enter Name: Aman
Student inserted successfully.

1. Insert Student
2. View All Students
3. Exit
Enter your choice: 1
Enter ID: 102
Enter Name: Rahul
Student inserted successfully.

1. Insert Student
2. View All Students
3. Exit
Enter your choice: 2

--- Student Records ---
ID: 101, Name: Aman
ID: 102, Name: Rahul
```