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

```
package com.mycompany.celsiustofahrenheit;
import java.util.Scanner;
public class CelsiusToFahrenheit {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter temperature in Celsius: ");
        double celsius = input.nextDouble();
        double fahrenheit = (celsius * 9/5) + 32;
        System.out.println("Temperature in Fahrenheit: " + fahrenheit);
    }
}
```

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

```
package com.mycompany.sortingusingcontrolstatements;
import java.util.Scanner;
public class SortingUsingControlStatements {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        // Get number of elements
        System.out.print("Enter the number of elements: ");
        int n = input.nextInt();
        int[] arr = new int[n];

        // Get elements from user
        System.out.println("Enter " + n + " numbers:");
        for (int i = 0; i < n; i++) {
            arr[i] = input.nextInt();
```

```
        }

        // Selection Sort using loops (control statements)
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            // Swap
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
        // Output sorted array
        System.out.println("Sorted array in ascending order:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Practice 3: Program to implement the Stack operations using Array

```
package com.mycompany.stackarray;
import java.util.Scanner;
public class StackArray {
    int top;
    int maxSize = 5;
    int[] stack = new int[maxSize];
```

```
// Constructor
StackArray() {
    top = -1;
}
// Push operation
void push(int value) {
    if (top == maxSize - 1) {
        System.out.println("Stack Overflow (Stack is full)");
    } else {
        top++;
        stack[top] = value;
        System.out.println("Pushed: " + value);
    }
}

// Pop operation
void pop() {
    if (top == -1) {
        System.out.println("Stack Underflow (Stack is empty)");
    } else {
        System.out.println("Popped: " + stack[top]);
        top--;
    }
}

// Peek operation see what's on top, but don't remove it.
void peek() {
    if (top == -1) {
        System.out.println("Stack is empty");
```

```
    } else {
        System.out.println("Top element: " + stack[top]);
    }
}

// Display stack
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 to test
public static void main(String[] args) {
    StackArray s = new StackArray();
    Scanner input = new Scanner(System.in);
    int choice, value;
    do {
        System.out.println("\nStack Operations Menu:");
        System.out.println("1. Push\n2. Pop\n3. Peek\n4. Display\n5. Exit");
        System.out.print("Enter your choice: ");
        choice = input.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter value to push: ");
```

```
                value = input.nextInt();
                s.push(value);
                break;
            case 2:
                s.pop();
                break;
            case 3:
                s.peek();
                break;
            case 4:
                s.display();
                break;
            case 5:
                System.out.println("Exiting Stack Program...");
                break;
            default:
                System.out.println("Invalid choice. Try again.");
        }
    } while (choice != 5);
  }
}
```

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

```
package com.mycompany.QueueArray;
import java.util.Scanner;
public class QueueArray {
    int front, rear;
    int maxSize = 5;
    int[] queue = new int[maxSize];
```

```
// Constructor
QueueArray() {
    front = -1;
    rear = -1;
}

// Enqueue (Insert)
void enqueue(int value) {
    if (rear == maxSize - 1) {   //array starts with index 0. so, we check max_size-1
        System.out.println("Queue Overflow (Queue is full)");
    } else {
        if (front == -1)
        front = 0; // First insertion
        rear++;
        queue[rear] = value;
        System.out.println("Enqueued: " + value);
    }
}

// Dequeue (Delete)
void dequeue() {
    if (front == -1 || front > rear) {
        System.out.println("Queue Underflow (Queue is empty)");
    } else {
        System.out.println("Dequeued: " + queue[front]);
        front++;
    }
}

// Peek (Front Element)
```

```
void peek() {
    if (front == -1 || front > rear) {
        System.out.println("Queue is empty");
    } else {
        System.out.println("Front element: " + queue[front]);
    }
}

// Display Queue
void display() {
    if (front == -1 || 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();
    }
}

// Main method
public static void main(String[] args) {
    QueueArray q = new QueueArray();
    Scanner input = new Scanner(System.in);
    int choice, value;

    do {
        System.out.println("\nQueue Operations Menu:");
        System.out.println("1. Enqueue\n2. Dequeue\n3. Peek\n4. Display\n5. Exit");
```

```
            System.out.print("Enter your choice: ");
            choice = input.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter value to enqueue: ");
                    value = input.nextInt();
                    q.enqueue(value);
                    break;
                case 2:
                    q.dequeue();
                    break;
                case 3:
                    q.peek();
                    break;
                case 4:
                    q.display();
                    break;
                case 5:
                    System.out.println("Exiting Queue Program...");
                    break;
                default:
                    System.out.println("Invalid choice. Try again.");
            }
        } while (choice != 5);
    }
}
```

Practice 5: Implement Overloading Methods, Constructors program. Implement Tower of Hanoi program using Recursion

```
package com.mycompany.towerofhanoi;
import java.util.Scanner;
public class TowerOfHanoi {
    // Recursive method to solve Tower of Hanoi
    static void solveHanoi(int n, char from, char to, char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + from + " to " + to);
            return;
        }
        // Move n-1 disks from source to auxiliary
        solveHanoi(n - 1, from, aux, to);

        // Move nth disk from source to target
        System.out.println("Move disk " + n + " from " + from + " to " + to);

        // Move n-1 disks from auxiliary to target
        solveHanoi(n - 1, aux, to, from);
    }
    // Main method
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter number of disks: ");
        int numDisks = input.nextInt();
        System.out.println("\nSteps to solve Tower of Hanoi:");
        solveHanoi(numDisks, 'A', 'C', 'B'); // A = source, C = target, B = helper
    }
}
```

Practice 6: Program to implement Linked List Concept Program to implement String Class

```
package com.mycompany.simpledemo;
class Node {
    int data;
    Node next;
    Node(int d) {
        data = d;
        next = null;
    }
}
class MyLinkedList {
    Node head;
    void insert(int value) {
        Node newNode = new Node(value);
        if (head == null)
            head = newNode;
        else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            temp.next = newNode;
        }
    }
    void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
```

```
    }
}
public class SimpleDemo {
    public static void main(String[] args) {
        // Linked List Demo
        MyLinkedList list = new MyLinkedList();
        list.insert(1);
        list.insert(2);
        list.insert(3);
        list.display();  // Output: 1 -> 2 -> 3 -> null
        // String Demo
        String s1 = "Java";
        String s2 = "String";
        String combined = s1 + " " + s2;
        System.out.println("Combined: " + combined);
        System.out.println("Length: " + combined.length());
        System.out.println("Substring: " + combined.substring(0, 4));
        System.out.println("Equal to 'Java'? " + s1.equals("Java"));
    }
}
```

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

```
package com.mycompany.inheritance;
abstract class Shape {
    protected String name;
    public Shape(String name) {
        this.name = name;
    }
    // Abstract methods
    abstract double area();
```

```
    abstract double perimeter();
    // Concrete method
    public void display() {
        System.out.println("Shape: " + name);
    }
}
class Circle extends Shape {
    private double radius;
    public Circle(double radius) {
        super("Circle");
        this.radius = radius;
    }
    @Override
    public double area() {
        return Math.PI * radius * radius;
    }
    @Override
    public double perimeter() {
        return 2 * Math.PI * radius;
    }
}

class Rectangle extends Shape {
    private double width, height;
    public Rectangle(double width, double height) {
        super("Rectangle");
        this.width = width;
        this.height = height;
    }
    @Override
    public double area() {
        return width * height;
```

```
    }

    @Override
    public double perimeter() {
        return 2 * (width + height);
    }
}
public class Inheritance {
    public static void main(String[] args) {
        // Create a Circle object
        Shape circle = new Circle(5.0);
        // Create a Rectangle object
        Shape rectangle = new Rectangle(4.0, 6.0);
        // Display circle details
        circle.display();
        System.out.printf("Area: %.2f%n", circle.area());
        System.out.printf("Perimeter: %.2f%n", circle.perimeter());
        System.out.println("------------------------");
        // Display rectangle details
        rectangle.display();
        System.out.printf("Area: %.2f%n", rectangle.area());
        System.out.printf("Perimeter: %.2f%n", rectangle.perimeter());
    }
}
```

Practice 8: Program to implement the concept Packages, Interfaces

```
package com.mycompany.packageinterfaces;
// Interface declaration
interface Payment {
    void pay(double amount);
}
// CreditCard class implements Payment
```

```
class CreditCard implements Payment {
    private String cardNumber;
    public CreditCard(String cardNumber) {
        this.cardNumber = cardNumber;
    }
    @Override
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Credit Card ending with " +
cardNumber.substring(cardNumber.length() - 4));
    }
}
// UPI class implements Payment
class UPIPayment implements Payment {
    private String upiId;
    public UPIPayment(String upiId) {
        this.upiId = upiId;
    }
    @Override
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using UPI ID: " + upiId);
    }
}
// Main class
public class PackageInterfaces{
    public static void main(String[] args) {
        Payment payment1 = new CreditCard("1234567890123456");
        Payment payment2 = new UPIPayment("john@upi");

        payment1.pay(1500.0);
        payment2.pay(850.75);
    }
}
```

Practice 9: Implement Exception Handling program

```
package com.mycompany.exceptionhandling;

import java.util.InputMismatchException;

import java.util.Scanner;

public class ExceptionHandling {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        try {

            System.out.print("Enter numerator: ");

            int numerator = scanner.nextInt();

            System.out.print("Enter denominator: ");

            int denominator = scanner.nextInt();

            int result = numerator / denominator;

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

        } catch (ArithmeticException e) {

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

        } catch (InputMismatchException e) {

            System.out.println("Invalid input! Please enter integers only.");

        } catch (Exception e) {

            System.out.println("An unexpected error occurred: " + e.getMessage());

        } finally {

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

            scanner.close();

        }

    }
```

Practice 10: Implement Multithreading and Program to Implement Binary Search Tree

```
package com.mycompany.multithreadbst;

class Node {

    int data;
```

```
    Node left, right;
    Node(int val) {
        data = val;
    }
}
class BST {
    Node root;
    // Simple insert method (no balancing, no sync for simplicity)
    void insert(int val) {
        root = insertRec(root, val);
    }
    Node insertRec(Node root, int val) {
        if (root == null) return new Node(val);
        if (val < root.data) root.left = insertRec(root.left, val);
        else if (val > root.data) root.right = insertRec(root.right, val);
        return root;
    }
    void inorder() {
        inorderRec(root);
    }
    void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.data + " ");
            inorderRec(root.right);
        }
    }
}
// Thread class to insert values into BST
class InsertThread extends Thread {
```

```
    BST tree;
    InsertThread(BST tree) {
        this.tree = tree;
    }
    public void run() {
        tree.insert(10);
        tree.insert(5);
        tree.insert(15);
    }
}
// Main class
public class MultiThreadBST {
    public static void main(String[] args) {
        BST tree = new BST();
        InsertThread t = new InsertThread(tree);
        t.start();
        try {
            t.join(); // Wait for thread to finish
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted.");
        }
        System.out.print("Inorder Traversal: ");
        tree.inorder(); // Should print: 5 10 15
    }
}
}
```

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

```
package com.mycompany.legacybtt;

import java.util.*;
```

```
class Node {
    int data; Node left, right;
    Node(int d) { data = d; }
}
public class LegacyBTT {
    static void inorder(Node n)  { if (n != null) { inorder(n.left);  System.out.print(n.data + " "); inorder(n.right);  }}
    static void preorder(Node n) { if (n != null) { System.out.print(n.data + " "); preorder(n.left); preorder(n.right); }}
    static void postorder(Node n){ if (n != null) { postorder(n.left); postorder(n.right); System.out.print(n.data + " "); }}

    public static void main(String[] args) {
        // Legacy classes
        Vector<Integer> vec = new Vector<>(List.of(1, 2, 3));
        Stack<Integer> stk = new Stack<>(); stk.push(10);
        Hashtable<Integer, String> ht = new Hashtable<>(); ht.put(1, "One");

        System.out.println("Vector: " + vec + ", Stack: " + stk.peek() + ", Hashtable: " + ht.get(1));

        // Binary tree creation
        Node root = new Node(50);
        root.left = new Node(30); root.right = new Node(70);
        root.left.left = new Node(20); root.right.right = new Node(80);

        // Traversals
        System.out.print("Inorder: "); inorder(root);
        System.out.print("\nPreorder: "); preorder(root);
        System.out.print("\nPostorder: "); postorder(root);
    }
}
```