ADVANCED JAVA PROGRAMMING LAB FILE

ITE-471

Submitted in partial fulfilment of the

requirements for the award of the degree

of

B.TECH

in

INFORMATION TECHNOLOGY

SUBMITTED TO:

Prof. Sartaj Singh Sodhi

SUBMITTED BY:

NAME - NISHANT SRINET

ENROLLMENT NO-04816401521

BATCH - 2021-25,7th sem

UNIVERSITY SCHOOL OF INFORMATION COMMUNICATION AND TECHNOLOGY

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

INDEX

S.NO.	NAME OF PRACTICAL	DATE	SIGN
1	Write a program (WAP) to demonstrate dynamic method dispatch.		
2	WAP to find volume of a cube and cuboid using abstract class in your code.		
3	WAP to demonstrate generics programming.		
4	WAP to demonstrate the concept of inner class and access all variables of both classes.		
5	WAP to sort 10 numbers using merge sort algorithm.		
6	WAP to sort 10 numbers using quicksort algorithm.		
7	WAP to create three threads each using Thread class and Runnable interface.		
8	WAP to demonstrate deadlock condition using multithreading.		

9	WAP where client and server exchange messages by using socket programming.		
10	WAP to demonstrate remote method invocation (RMI).		

PRACTICAL 1

Aim: Write a program (WAP) to demonstrate dynamic method dispatch.

CODE:

```
class Shape {
    double area() {
        System.out.println("Calculating area of a shape");
        return 0;
    }
}

class Circle extends Shape {
    double radius;
    Circle(double radius) {
        this.radius = radius;
    }
    @Override
    double area() {
        System.out.println("\nCalculating area of a circle");
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    double length, width;
    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    @Override
    double area() {
        System.out.println("\nCalculating area of a rectangle");
        return length * width;
    }
}

public class DynamicDispatchExample {
    public static void main(String[] args) {
        Shape myShape = new Shape();
        myShape.area();
        myShape = new Circle(5);
        System.out.println("Area: " + myShape.area());
        myShape = new Rectangle(4, 6);
        System.out.println("Area: " + myShape.area());
    }
}
```

OUTPUT:

```
Calculating area of a shape

Calculating area of a circle
Area: 78.53981633974483
```

```
Calculating area of a rectangle
Area: 24.0
```

PRACTICAL 2

Aim: WAP to find volume of a cube and cuboid using abstract class in your code.

CODE:

```
abstract class Solid {
    abstract double volume();
}

class Cube extends Solid {
    double side;
    Cube(double side) {
        this.side = side;
    }
    @Override
    double volume() {
        return side * side * side;
    }
}

class Cuboid extends Solid {
    double length, width, height;
    Cuboid(double length, double width, double height) {
        this.length = length;
        this.width = width;
        this.height = height;
    }
    @Override
    double volume() {
        return length * width * height;
    }
}

public class VolumeCalculator {
    public static void main(String[] args) {
        Solid cube = new Cube(3);
        System.out.println("Volume of Cube: " + cube.volume());

        Solid cuboid = new Cuboid(4, 5, 6);
        System.out.println("Volume of Cuboid: " + cuboid.volume());
    }
}
```

OUTPUT:

```
Volume of Cube: 27.0
Volume of Cuboid: 120.0
```

PRACTICAL 3

Aim: WAP to demonstrate generics programming.

CODE:

```
class Pair<K, V> {
    private K key;
    private V value;
    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey() {
        return key;
    }
    public V getValue() {
        return value;
    }
}

public class PairDemo {
    public static void main(String[] args) {
        Pair<Integer, String> pair1 = new Pair<>(1, "One");
        System.out.println("Key: " + pair1.getKey() + ", Value: " +
pair1.getValue());

        Pair<String, Double> pair2 = new Pair<>("Pi", 3.14);
        System.out.println("Key: " + pair2.getKey() + ", Value: " +
pair2.getValue());
    }
}
```

OUTPUT:

```
Key: 1, Value: One
Key: Pi, Value: 3.14
```

PRACTICAL 4

Aim: WAP to demonstrate the concept of inner class and access all variables of both classes.

CODE:

```
class OuterClass {
    private String outer = "I am an Outer Class Variable";
    class InnerClass {
        private String inner = "I am an Inner Class Variable";
        public void displayVariables() {
            System.out.println("Accessing from Inner Class:");
            System.out.println("Accessing Outer Variable: " + outer);
            System.out.println("Accessing Inner Variable: " + inner);
        }
    }
    public void createInnerInstance() {
        InnerClass inner = new InnerClass();
        inner.displayVariables();
    }
}

public class InnerClassDemo {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.createInnerInstance();
    }
}
```

OUTPUT:

```
Accessing from Inner Class:
Accessing Outer Variable: I am an Outer Class Variable
Accessing Inner Variable: I am an Inner Class Variable
```

PRACTICAL 5

Aim: WAP to sort 10 numbers using merge sort algorithm.

CODE:

```
import java.util.Arrays;

public class MergeSort {
    public static void mergeSort(int[] array, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;
            mergeSort(array, left, mid);
            mergeSort(array, mid + 1, right);
            merge(array, left, mid, right);
        }
    }
    public static void merge(int[] array, int left, int mid, int
right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;
        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];
        for (int i = 0; i < n1; i++) {
            leftArray[i] = array[left + i];
        }
        for (int j = 0; j < n2; j++) {rightArray[j] = array[mid + 1 +
j];
        }
        int i = 0, j = 0;
        int k = left;
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                array[k] = leftArray[i];
                i++;
            } else {
                array[k] = rightArray[j];
                j++;
            }
            k++;
        }
        while (i < n1) {
            array[k] = leftArray[i];
            i++;
            k++;
        }
        while (j < n2) {
            array[k] = rightArray[j];
            j++;
            k++;
        }
    }

    public static void main(String[] args) {
        int[] numbers = {45, 2, 89, 32, 67, 14, 76, 23, 11, 54};
        System.out.println("Original Array: " +
```

```
Arrays.toString(numbers));
        mergeSort(numbers, 0, numbers.length - 1);
        System.out.println("Sorted Array: " +
Arrays.toString(numbers));
    }
}
```

OUTPUT:

```
Original Array: [45, 2, 89, 32, 67, 14, 76, 23, 11, 54]
Sorted Array: [2, 11, 14, 23, 32, 45, 54, 67, 76, 89]
```

PRACTICAL 6

Aim: WAP to sort 10 numbers using quicksort algorithm.

CODE:

```
import java.util.Arrays;

public class QuickSort {
    public static void quickSort(int[] array, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(array, low, high);
            quickSort(array, low, pivotIndex - 1);
            quickSort(array, pivotIndex + 1, high);
        }
    }
    public static int partition(int[] array, int low, int high) {
        int pivot = array[high];
        int i = low - 1;for (int j = low; j < high; j++) {
            if (array[j] <= pivot) {
                i++;
                swap(array, i, j);
            }
        }
        swap(array, i + 1, high);
        return i + 1;
    }
    public static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String[] args) {
        int[] numbers = {45, 2, 89, 32, 67, 14, 76, 23, 11, 54};
        System.out.println("Original Array: " +
Arrays.toString(numbers));
        quickSort(numbers, 0, numbers.length - 1);
        System.out.println("Sorted Array: " +
Arrays.toString(numbers));
    }
}
```

OUTPUT:

```
Original Array: [45, 2, 89, 32, 67, 14, 76, 23, 11, 54]
Sorted Array: [2, 11, 14, 23, 32, 45, 54, 67, 76, 89]
```

PRACTICAL 7

Aim: WAP to create three threads each using Thread class and Runnable interface.

CODE:

```
class MyThread extends Thread {
    private String name;
    public MyThread(String name) {
        this.name = name;
    }
    @Override
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println(name + " (Thread): Step " + i);
            try {
                Thread.sleep(500); // Pause for 500ms
            } catch (InterruptedException e) {
                System.out.println(name + " interrupted.");
            }
        }
        System.out.println(name + " (Thread) completed.");
    }
}


class MyRunnable implements Runnable {
    private String name;

    public MyRunnable(String name) {
        this.name = name;
    }
    @Override
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println(name + " (Runnable): Step " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(name + " interrupted.");
            }
        }
        System.out.println(name + " (Runnable) completed.");
    }
}

public class ThreadExample {
    public static void main(String[] args) {
        // Creating threads by extending the Thread class
        MyThread thread1 = new MyThread("Thread 1");
        Thread runnableThread2 = new Thread(new MyRunnable("Runnable
1"));
        Thread runnableThread3 = new Thread(new MyRunnable("Runnable
2"));

        // Starting the threads
```

```
        thread1.start();
        runnableThread2.start();
        runnableThread3.start();
    }
}
```

OUTPUT:

```
Runnable 1 (Runnable): Step 1
Runnable 2 (Runnable): Step 1
Thread 1 (Thread): Step 1
Runnable 2 (Runnable): Step 2
Thread 1 (Thread): Step 2
Runnable 1 (Runnable): Step 2
Thread 1 (Thread): Step 3
Runnable 1 (Runnable): Step 3
Runnable 2 (Runnable): Step 3
Runnable 1 (Runnable) completed.
Thread 1 (Thread) completed.
Runnable 2 (Runnable) completed.
```

PRACTICAL 8

Aim: WAP to demonstrate deadlock condition using multithreading.

CODE:

```
public class DeadlockDemo {
    private static final Object resource1 = new Object();
    private static final Object resource2 = new Object();

    public static void main(String[] args) {
        // Thread 1 tries to lock resource1 then resource2
        Thread thread1 = new Thread(() -> {
            synchronized (resource1) {
                System.out.println("Thread 1: Locked resource 1");

                try {
                    Thread.sleep(100);
                }
       catch (InterruptedException e) {
                    System.out.println("Thread 1 interrupted.");
                }
                System.out.println("Thread 1: Waiting to lock resource
2");
                synchronized (resource2) {
                    System.out.println("Thread 1: Locked resource 2");
                }
            }
        });

        // Thread 2 tries to lock resource2 then resource1
        Thread thread2 = new Thread(() -> {
            synchronized (resource2) {
                System.out.println("Thread 2: Locked resource 2");

                try {
                    Thread.sleep(100);
                }
       catch (InterruptedException e) {
                    System.out.println("Thread 2 interrupted.");
                }
                System.out.println("Thread 2: Waiting to lock resource
1");
                synchronized (resource1) {
                    System.out.println("Thread 2: Locked resource 1");
                }
            }
        });

        thread1.start();
        thread2.start();
    }
}
```

OUTPUT:

```
Thread 1: Locked resource 1
```

```
Thread 2: Locked resource 2
Thread 1: Waiting to lock resource 2
Thread 2: Waiting to lock resource 1
```

PRACTICAL 9

Aim: WAP where client and server exchange messages by using socket programming.

CODE:

1. Server.java

```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(5000)) {
            System.out.println("Server is listening on port 5000...");

            // Wait for a client to connect
            Socket socket = serverSocket.accept();
            System.out.println("Client connected");

            // Set up input and output streams
            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new
PrintWriter(socket.getOutputStream(), true);

            // Receive message from client
            String clientMessage = input.readLine();
            System.out.println("Client: " + clientMessage);

            // Send response to client
            String serverResponse = "Hello from the server!";
            output.println(serverResponse);
            System.out.println("Server: " + serverResponse);

            // Close connection
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2. Client.java

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 5000)) {
            System.out.println("Connected to the server");
```

```
            // Set up input and output streams
            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new
PrintWriter(socket.getOutputStream(), true);

            // Send message to server
            String clientMessage = "Hello from the client!";
            output.println(clientMessage);
            System.out.println("Client: " + clientMessage);

            // Receive response from server
            String serverResponse = input.readLine();
            System.out.println("Server: " + serverResponse);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:

Server Console:

```
Server is listening on port 5000...
Client connected
Client: Hello from the client!
Server: Hello from the server!
```

Client Console:

```
Connected to the server
Client: Hello from the client!
Server: Hello from the server!
```

PRACTICAL 10

Aim: WAP to demonstrate remote method invocation (RMI).

CODE:

1. RemoteInterface.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RemoteInterface extends Remote {
    String sayHello(String name) throws RemoteException;
}
```

2. RemoteImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RemoteImpl extends UnicastRemoteObject implements
RemoteInterface {
    protected RemoteImpl() throws RemoteException {
        super();
    }
    @Override
    public String sayHello(String name) throws RemoteException {
        return "Hello, " + name + "! Greetings from the server.";
    }
}
```

3. Server.java

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class RMIServer {
    public static void main(String[] args) {
        try {
            // Create and start the RMI registry on port 1099
            LocateRegistry.createRegistry(1099);

            // Create an instance of the remote object
            RemoteImpl remoteObj = new RemoteImpl();

            // Bind the remote object in the registry with a name
            Naming.rebind("RemoteHello", remoteObj);

            System.out.println("Server is ready.");
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
}
```

4. Client.java

```
import java.rmi.Naming;

public class RMIClient {
    public static void main(String[] args) {
        try {
            // Lookup the remote object in the registry
            RemoteInterface remoteObj = (RemoteInterface)
Naming.lookup("rmi://localhost:1099/RemoteHello");

            // Invoke the remote method
            String response = remoteObj.sayHello("Alice");
            System.out.println("Response from server: " + response);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:

Server Console:

```
Server is ready.
```

Client Console:

```
Response from server: Hello, Alice! Greetings from the server.
```