Practice list for the Java laboratory class

1. Write a program in Java to demonstrate single inheritance, multilevel inheritance, and, hierarchical inheritance.

```
// Single Inheritance
class Parent {
    void displayParent() {
        System.out.println("This is the parent class.");
    }
}

class Child extends Parent {
    void displayChild() {
        System.out.println("This is the child class.");
    }
}

// Multilevel Inheritance
class Grandparent {
    void displayGrandparent() {
        System.out.println("This is the grandparent class.");
    }
}

class Parent extends Grandparent {
    void displayParent() {
        System.out.println("This is the parent class.");
    }
}

class Child extends Parent {
    void displayChild() {
        System.out.println("This is the child class.");
    }
}

// Hierarchical Inheritance
class Parent {
    void displayParent() {
        System.out.println("This is the parent class.");
    }
}

class Child1 extends Parent {
    void displayChild1() {
        System.out.println("This is the first child class.");
    }
}

class Child2 extends Parent {
    void displayChild2() {
        System.out.println("This is the second child class.");
    }
}
```

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Child child = new Child();
        child.displayGrandparent();
        child.displayParent();
        child.displayChild();
    }
}
```

2. Write a program in Java to develop an overloaded constructor. Also, develop the copy constructor to create a new object with the state of the existing object.

```
class Person {
    String name;
    int age;

    // Overloaded Constructor
    Person(String n, int a) {
        name = n;
        age = a;
    }

    // Copy Constructor
    Person(Person p) {
        name = p.name;
        age = p.age;
    }

    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}

public class ConstructorDemo {
    public static void main(String[] args) {
        Person person1 = new Person("Alice", 25);
        Person person2 = new Person(person1); // Using the copy constructor
        person1.display();
        person2.display();
    }
}
```

3. Write a program in Java to demonstrate the use of this keyword. Check whether this can access the private members of the class r not.

```
class MyClass {
    private int num;

    MyClass(int num) {
        this.num = num;
    }

    void display() {
        System.out.println("Value of num: " + num);
    }
}

public class ThisKeywordDemo {
    public static void main(String[] args) {
        MyClass obj = new MyClass(42);
        obj.display();
    }
}
```

4. Write an application that illustrates how to access a hidden variable. Class A declares a static variable x. The class B extends A and declares an instance variable x, display() method in class B displays both of these variables.

```
class A {
    static int x = 10;
}

class B extends A {
    int x = 20;

    void display() {
        System.out.println("Hidden x in B: " + this.x); // Accessing instance variable x in class
B
        System.out.println("x in A: " + A.x); // Accessing static variable x in class A
    }
}

public class HiddenVariableDemo {
    public static void main(String[] args) {
        B obj = new B();
        obj.display();
    }
}
```

5. a) Write a program in Java to develop a user-defined exception for the Divide by Zero' error.

```
class DivideByZeroException extends Exception {
    DivideByZeroException(String message) {
        super(message);
    }
}

public class CustomExceptionDemo {
    public static void main(String[] args) {
        int numerator = 10;
        int denominator = 0;

        try {
            if (denominator == 0) {
                throw new DivideByZeroException("Division by zero is not allowed.");
            }
            int result = numerator / denominator;
            System.out.println("Result: " + result);
        } catch (DivideByZeroException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

b) Write a program in Java to demonstrate multiple try blocks and multiple catch exceptions.

```
public class MultipleTryCatchDemo {
    public static void main(String[] args) {
        try {
            int[] arr = new int[3];
            arr[3] = 10; // ArrayIndexOutOfBoundsException

            int num = Integer.parseInt("abc"); // NumberFormatException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index Out Of Bounds");
        } catch (NumberFormatException e) {
            System.out.println("Number Format Exception");
        }
    }
}
```

c) Write a Java program using nested try-catch blocks.

```
public class NestedTryCatchDemo {
    public static void main(String[] args) {
        try {
            try {
```

```
            int num = Integer.parseInt("abc"); // NumberFormatException
        } catch (NumberFormatException e) {
            System.out.println("Inner Catch: Number Format Exception");
        }

        int[] arr = new int[3];
        arr[3] = 10; // ArrayIndexOutOfBoundsException
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Outer Catch: Array Index Out Of Bounds");
    }
  }
}
```

d) If the user enters only one command line argument, the inner try block should throw an Exception.

e) If the user enters two command-line arguments, divide the first argument by the second argument. If the second argument is zero, then a proper exception should be handled.

```
public class CommandLineArgsDemo {
    public static void main(String[] args) {
        try {
            if (args.length == 1) {
                throw new Exception("Only one argument provided.");
            } else if (args.length == 2) {
                int dividend = Integer.parseInt(args[0]);
                int divisor = Integer.parseInt(args[1]);

                if (divisor == 0) {
                    throw new ArithmeticException("Division by zero.");
                }

                int result = dividend / divisor;
                System.out.println("Result: " + result);
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format.");
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

f) Create a simple Java program to create a banking application in which the user deposits BDT 1000 and begins withdrawing BDT 500 and BDT 400, and then encounters an exception with the message Not Sufficient Fund" when the user withdraws BDT 300.

```
class InsufficientFundsException extends Exception {
    InsufficientFundsException(String message) {
        super(message);
    }
}

class BankAccount {
    private double balance;

    BankAccount(double initialBalance) {
        balance = initialBalance;
    }

    void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
            throw new InsufficientFundsException("Not Sufficient Funds");
        }
        balance -= amount;
        System.out.println("Withdrawal successful. Remaining balance: " + balance);
    }
}

public class BankingAppDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);

        try {
            account.withdraw(500);
            account.withdraw(400);
            account.withdraw(300); // This will throw InsufficientFundsException
        } catch (InsufficientFundsException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

g) Write a program to handle interrupted exceptions and illegal argument exceptions using try-catch-finally and throw.

```
public class ExceptionHandlingDemo {
    public static void main(String[] args) {
        try {
            Thread.sleep(1000); // InterruptedException

            int num = Integer.parseInt("abc"); // NumberFormatException
        } catch (InterruptedException e) {
            System.out.println("Interrupted Exception");
        } catch (NumberFormatException e) {
            System.out.println("Number Format Exception");
        } finally {
            System.out.println("Finally block executed.");
        }
    }
}
```

5. All the answers in one code:

```
class DivideByZeroException extends Exception {
    DivideByZeroException(String message) {
        super(message);
    }
}

public class ExceptionHandlingDemo {
    public static void main(String[] args) {
        // a) User-defined exception for Divide by Zero error
        try {
            int numerator = 10;
            int denominator = 0;
            if (denominator == 0) {
                throw new DivideByZeroException("Division by zero is not allowed.");
            }
            int result = numerator / denominator;
            System.out.println("Result: " + result);
        } catch (DivideByZeroException e) {
            System.out.println("Error: " + e.getMessage());
        }

        // b) Multiple try blocks and multiple catch exceptions
        try {
            int[] numbers = { 1, 2, 3 };
            System.out.println(numbers[5]); // ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index out of bounds: " + e.getMessage());
        } finally {
            System.out.println("This block always executes.");
        }
```

```
// c) Nested try-catch blocks
try {
    try {
        int[] nums = { 1, 2, 3 };
        System.out.println(nums[10]); // ArrayIndexOutOfBoundsException
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Inner catch: " + e.getMessage());
    }
    int a = 10 / 0; // ArithmeticException
} catch (ArithmeticException e) {
    System.out.println("Outer catch: " + e.getMessage());
}

// d) One command line argument
try {
    int value = Integer.parseInt(args[0]);
    try {
        if (value == 1) {
            throw new Exception("Exception thrown in inner try block.");
        }
    } catch (Exception e) {
        System.out.println("Inner catch: " + e.getMessage());
    }
} catch (NumberFormatException e) {
    System.out.println("Invalid input.");
}

// e) Two command line arguments
try {
    int num1 = Integer.parseInt(args[0]);
    int num2 = Integer.parseInt(args[1]);
    try {
        int result = num1 / num2;
        System.out.println("Result: " + result);
    } catch (ArithmeticException e) {
        System.out.println("Division by zero: " + e.getMessage());
    }
} catch (NumberFormatException e) {
    System.out.println("Invalid input.");
}

// f) Banking application
class InsufficientFundsException extends Exception {
    InsufficientFundsException(String message) {
        super(message);
    }
}

class BankAccount {
    private double balance = 1000.0;

    void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
```

```
                throw new InsufficientFundsException("Not Sufficient Funds");
            }
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        }
    }

    BankAccount account = new BankAccount();
    try {
        account.withdraw(500.0);
        account.withdraw(400.0);
        account.withdraw(300.0); // Throws InsufficientFundsException
    } catch (InsufficientFundsException e) {
        System.out.println("Error: " + e.getMessage());
    }

    // g) Handling Interrupted and Illegal Argument exceptions
    try {
        Thread.sleep(1000); // InterruptedException
    } catch (InterruptedException e) {
        System.out.println("Interrupted: " + e.getMessage());
    }

    try {
        int number = Integer.parseInt("abc"); // NumberFormatException
    } catch (NumberFormatException e) {
        System.out.println("Invalid number format: " + e.getMessage());
    }
  }
}
```

6. Describe abstract class called Shape which has three subclasses say Triangle, Rectangle, and circle. Define one method area() in the abstract class and override this area() in these three subclass to calculate for specific object i.e. area() of triangle subclass should calculate area of Triangle etc. Same for Rectangle and Circle.

```
abstract class Shape {
    abstract double area();
}

class Triangle extends Shape {
    double base, height;

    Triangle(double b, double h) {
        base = b;
        height = h;
    }

    @Override
    double area() {
        return 0.5 * base * height;
```

```
    }
}

class Rectangle extends Shape {
    double length, width;

    Rectangle(double l, double w) {
        length = l;
        width = w;
    }

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

class Circle extends Shape {
    double radius;

    Circle(double r) {
        radius = r;
    }

    @Override
    double area() {
        return Math.PI * radius * radius;
    }
}
```

7. Write a program that executes two threads. One thread displays "Thread 1" every 2,000 milliseconds, and the other displays "Thread 2" every 4,000 milliseconds. Create the threads by extending the Thread class.

```
class Thread1 extends Thread {
    public void run() {
        while (true) {
            System.out.println("Thread 1");
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Thread2 extends Thread {
    public void run() {
        while (true) {
            System.out.println("Thread 2");
```

```
            try {
                Thread.sleep(4000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class ThreadDemo {
    public static void main(String[] args) {
        Thread1 t1 = new Thread1();
        Thread2 t2 = new Thread2();

        t1.start();
        t2.start();
    }
}
```

8. Write a program in Java to demonstrate use of synchronization of threads when multiple threads are trying to update common variable.

```
class Counter {
    private int count = 0;

    synchronized void increment() {
        count++;
        System.out.println("Count: " + count);
    }
}

class Worker extends Thread {
    Counter counter;

    Worker(Counter c) {
        counter = c;
    }

    public void run() {
        for (int i = 0; i < 5; i++) {
            counter.increment();
        }
    }
}

public class SynchronizationDemo {
    public static void main(String[] args) {
        Counter counter = new Counter();
        Worker w1 = new Worker(counter);
        Worker w2 = new Worker(counter);
```

```
        w1.start();
        w2.start();
    }
}
```

9.Write a program that creates an array list, adds a Loan object, a Date object, a string, and a Circle object to the list. Then, using a loop and the to String() method of each object, show each element in the list.

```
import java.util.ArrayList;
import java.util.Date;

class Loan {
    // Implement Loan class if needed
}

class Circle {
    // Implement Circle class if needed
}

public class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<Object> list = new ArrayList<>();
        list.add(new Loan());
        list.add(new Date());
        list.add("Hello");
        list.add(new Circle());

        for (Object obj : list) {
            System.out.println(obj.toString());
        }
    }
}
```

10. Write an AWT program to create a Frame.

```
import java.awt.Frame;

public class AWTFrameDemo {
    public static void main(String[] args) {
        Frame frame = new Frame("My AWT Frame");
        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

11. Develop an AWT program that draws a circle. The dimension of the applet should be 500*300 pixels. The circle should be centered in the applet and have radius of 100 pixels. Display your name centered in a circle.

```
import java.awt.*;
import java.applet.*;

public class AWTDrawCircleDemo extends Applet {
    public void paint(Graphics g) {
        int centerX = getWidth() / 2;
        int centerY = getHeight() / 2;
        int radius = 100;

        g.drawOval(centerX - radius, centerY - radius, 2 * radius, 2 * radius);
        g.drawString("Your Name", centerX, centerY);
    }
}
```

12. Develop an program that contains three check boxes and 30 * 30 pixel canvas. The three checkboxes should be labelled "Red", "Green", "Blue". The selection of the check boxes determine the color of the canvas. For example, if the user selects both "Red" and "Blue", the canvas should be purple.

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

public class AWTCheckboxCanvasDemo extends Frame implements ItemListener {
    Checkbox red, green, blue;
    Canvas canvas;

    AWTCheckboxCanvasDemo() {
        red = new Checkbox("Red");
        green = new Checkbox("Green");
        blue = new Checkbox("Blue");

        canvas = new Canvas();
        canvas.setSize(30, 30);

        red.setBounds(50, 100, 100, 30);
        green.setBounds(150, 100, 100, 30);
        blue.setBounds(250, 100, 100, 30);
        canvas.setBounds(150, 150, 30, 30);

        red.addItemListener(this);
        green.addItemListener(this);
        blue.addItemListener(this);

        add(red);
        add(green);
        add(blue);
        add(canvas);

        setLayout(null);
        setSize(400, 300);
        setVisible(true);
    }
```

```
    public void itemStateChanged(ItemEvent e) {
        int r = red.getState() ? 255 : 0;
        int g = green.getState() ? 255 : 0;
        int b = blue.getState() ? 255 : 0;
        Color color = new Color(r, g, b);
        canvas.setBackground(color);
    }

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

13. Write a TCP Client-Server program to get Date & Time detsils from Server on the Client request.

```
// TCP Server
import java.io.*;
import java.net.*;

public class TCPServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(12345);

        while (true) {
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
            out.println(new java.util.Date().toString());

            out.close();
            clientSocket.close();
        }
    }
}
// TCP Client
import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String[] args) throws IOException {
        String serverIP = "127.0.0.1"; // Server IP address
        int serverPort = 12345;

        Socket socket = new Socket(serverIP, serverPort);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String serverResponse = in.readLine();
        System.out.println("Server Date & Time: " + serverResponse);

        in.close();
        socket.close();
    }
}
```

14. Write a UDO Client-Server program in which the Client sends any string and Server responds with Reverse string.

```
// UDP Server
import java.net.*;
public class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];

        while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);
            String clientMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
            System.out.println("Received from client: " + clientMessage);

            String reversedMessage = new StringBuilder(clientMessage).reverse().toString();
            InetAddress clientIP = receivePacket.getAddress();
            int clientPort = receivePacket.getPort();
            byte[] sendData = reversedMessage.getBytes();

            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
clientIP, clientPort);
            serverSocket.send(sendPacket);
        }
    }
}

// UDP Client
import java.net.*;
public class UDPClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress serverIP = InetAddress.getByName("localhost");
        int serverPort = 9876;

        String message = "Hello UDP Server!";
        byte[] sendData = message.getBytes();

        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
serverIP, serverPort);
        clientSocket.send(sendPacket);

        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
        clientSocket.receive(receivePacket);

        String reversedMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
        System.out.println("Received from server: " + reversedMessage);
```

```
        clientSocket.close();
    }
}
```