Q.1Write a Java program that reads an integer between 0 and 1000 and adds all the digits in the integer

```
import java.util.Scanner;

class Main {

  public static void main(String[] args) {


    int n , sum = 0;

    Scanner sc = new Scanner (System.in);

    System.out.println("Enter the no.:");

    n = sc.nextInt();


    if (n>0 && n < 1000){

      while(n>0){

        sum += n%10;

        n = n/ 10;

      }

    System.out.println("The sum is ;"+sum);


    }

    else {

      System.out.println("Error input");

    }

  }

}
```

Q2Write a Java program that checks whether an input string is a palindrome.

```
import java.util.Scanner;

class Main {

  public static void main(String[] args) {


    Scanner sc = new Scanner (System.in);

    System.out.println("Enter the Word;");

    String str = sc.nextLine();


    str = str.replace("\\s+","").toLowerCase();


    boolean isPalindrome = true;

    int lenght = str.length();


    for( int i = 0 ; i < lenght/2; i++){
       if(str.charAt(i) != str.charAt(lenght - 1- i)){
          isPalindrome = false;
          break;
       }
    }
    if (isPalindrome){
       System.out.println("Yes");
    }
    else{
       System.out.println("No");
    }
  }
}
```

Q2.2Write a Java program that checks whether two input strings are anagrams of each other.

```
import java.util.Scanner;
import java.util.Arrays;

class Main {
   public static boolean areAnagrams(String str1, String str2){

      str1 = str1.replaceAll("\\s+","").toLowerCase();
      str2 = str2.replaceAll("\\s+","").toLowerCase();

      char[] charArray1 = str1.toCharArray();
      char[] charArray2 = str2.toCharArray();

      Arrays.sort(charArray1);
      Arrays.sort(charArray2);

      return Arrays.equals(charArray1, charArray2);

   }
   public static void main(String[]args) {
      Scanner sc =  new Scanner(System.in);

      System.out.println("Enter teh first word:");
      String str1 = sc.nextLine();
```

```
        System.out.println("Enter teh 2nd word:");
        String str2= sc.nextLine();


        if (areAnagrams(str1,str2)){
            System.out.println("yes");
        }
        else{
            System.out.println("no");
        }
    }
}
```

Q3Write a java program to create a class Student with data name, city and age along with method addData and printData to input and display the data. Create the two objects s1, s2 to declare and access the values.

```
import java.util.Scanner;
class Student{
    String name;
    int age;
    String city;


    public void addData(){
        Scanner sc = new Scanner(System.in);


        System.out.println("enter the name:");
        name = sc.nextLine();
        System.out.println("enter the city:");
        city = sc.nextLine();
        System.out.println("enter the age:");
        age = sc.nextInt();
    }


    public void printData(){
        System.out.println("Name:"+name);
        System.out.println("Age:"+age);
        System.out.println("city"+city);
    }
}
public class Main {
    public static void main(String[]args){

        Student s1 = new Student();

        Student s2 = new Student();

        System.out.println("Enter the deatils of S1");

        s1.addData();

        System.out.println("\nDetails of S1");

        s1.printData();

        System.out.println("\nEnter the deatils of S2");

        s2.addData();

        System.out.println("\nDetails of S2");

        s2.printData();

    }

}
```

Q4Write a Java program that creates a class hierarchy for employees of a company. The base class should be Employee, with subclasses Manager, Developer, and Programmer.

```
// Base Class Employee

class Employee {

    String name;

    String address;

    double salary;

    String jobTitle;


    // Method to add data

    public void addData(String name, String address, double salary, String jobTitle) {

        this.name = name;

        this.address = address;

        this.salary = salary;

        this.jobTitle = jobTitle;

    }


    // Method to print data

    public void printData() {

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

        System.out.println("Address: " + address);

        System.out.println("Salary: " + salary);

        System.out.println("Job Title: " + jobTitle);

    }
    // Method to calculate bonus (generic for all employees)
    public double calculateBonus() {
        return salary * 0.1;  // Default bonus calculation
    }
}

// Subclass Manager
class Manager extends Employee {
    int teamSize;

    public void addManagerData(String name, String address, double salary, int teamSize) {
        addData(name, address, salary, "Manager");
        this.teamSize = teamSize;
    }

    public void printManagerData() {
        printData();
        System.out.println("Team Size: " + teamSize);
    }

    @Override
    public double calculateBonus() {
        return salary * 0.2 + (teamSize * 500);  // Bonus based on team size
    }
}

// Subclass Developer
class Developer extends Employee {
    String programmingLanguage;

    public void addDeveloperData(String name, String address, double salary, String programmingLanguage) {
        addData(name, address, salary, "Developer");
        this.programmingLanguage = programmingLanguage;
    }

    public void printDeveloperData() {
        printData();
        System.out.println("Programming Language: " + programmingLanguage);
    }

    @Override
    public double calculateBonus() {
        return salary * 0.15;  // Bonus based on salary
    }
}

// Subclass Programmer
class Programmer extends Employee {
    int linesOfCodeWritten;

    public void addProgrammerData(String name, String address, double salary, int linesOfCodeWritten) {
        addData(name, address, salary, "Programmer");
        this.linesOfCodeWritten = linesOfCodeWritten;
    }

    public void printProgrammerData() {
        printData();
        System.out.println("Lines of Code Written: " + linesOfCodeWritten);
    }

    @Override
    public double calculateBonus() {
        return salary * 0.1 + (linesOfCodeWritten * 0.01);  // Bonus based on lines of code
    }
}

public class Main {
    public static void main(String[] args) {
        // Create objects for each type of employee
        Manager manager = new Manager();
        Developer developer = new Developer();
        Programmer programmer = new Programmer();

        // Adding data for each employee
        manager.addManagerData("Amit", "Pune", 150000, 10);
        developer.addDeveloperData("Raj", "Mumbai", 120000, "Java");
        programmer.addProgrammerData("Sneha", "Bangalore", 90000, 20000);

        // Printing employee details and bonus calculation
        System.out.println("--- Manager Details ---");
        manager.printManagerData();
        System.out.println("Bonus: ₹" + manager.calculateBonus());

        System.out.println("\n--- Developer Details ---");
        developer.printDeveloperData();
        System.out.println("Bonus: ₹" + developer.calculateBonus());

        System.out.println("\n--- Programmer Details ---");
        programmer.printProgrammerData();
        System.out.println("Bonus: ₹" + programmer.calculateBonus());
    }
}
```

Step 1: Define the Base Class Employee MIT ADT University, Pune The Employee class will contain common properties and methods that are shared among all types of employees. Step 2: Define the Subclass Manager The Manager class will extend Employee and implement the abstract methods. Step 3: Define the Subclass Developer The Developer class will also extend Employee and implement the abstract methods. Step 4: Define the Subclass Programmer The Programmer class will extend Employee and implement the abstract methods. Step 5: Testing the Hierarchy Create a main method to test the hierarchy and ensure everything works as expected.

Q5Write a java program to create an abstract class named Shape that contains two integers and an empty method named print Area (). Provide three classes named Rectangle, Triangle and Circle

```
import java.util.Scanner;

// Abstract class Shape
abstract class Shape {
    int dimension1, dimension2;

    // Abstract method to calculate and print area
    abstract void printArea();
}

// Rectangle class extending Shape
class Rectangle extends Shape {
    @Override
    void printArea() {
        int area = dimension1 * dimension2;
        System.out.println("Area of Rectangle: " + area);
    }
}

// Triangle class extending Shape
class Triangle extends Shape {
    @Override
    void printArea() {
        double area = 0.5 * dimension1 * dimension2;
        System.out.println("Area of Triangle: " + area);
    }
}

// Circle class extending Shape
class Circle extends Shape {
    @Override
    void printArea() {
        double area = Math.PI * dimension1 * dimension1; // Circle uses only one dimension (radius)
        System.out.println("Area of Circle: " + area);
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Choose a shape to calculate the area:");
        System.out.println("1. Rectangle\n2. Triangle\n3. Circle");
        int choice = sc.nextInt()
        Shape shape = null; // Reference to the abstract class
        switch (choice) {
            case 1:
                shape = new Rectangle();
                System.out.println("Enter length and breadth of the rectangle:");
                shape.dimension1 = sc.nextInt();
                shape.dimension2 = sc.nextInt();
                break;
            case 2:
```

```
            shape = new Triangle();

            System.out.println("Enter base and height of the triangle:");

            shape.dimension1 = sc.nextInt();

            shape.dimension2 = sc.nextInt();

            break;

         case 3:

            shape = new Circle();

            System.out.println("Enter the radius of the circle:");

            shape.dimension1 = sc.nextInt();

            break;

         default:

            System.out.println("Invalid choice.");

            return; // Exit program for invalid input

      }

      // Print the area

      shape.printArea();

   }

}
```

1. Create an abstract class Shape. 2. Declare two members of type integer. MIT ADT University, Pune 3. Declare method name as printArea() 4. Declare three classes Rectangle, Triangle and Circle who extends Superclass Shape. 5. In SubClasses override the method printArea() which calculates the area of a given shape. 6. Ask the user to enter the shape to calculate the area, and use the switch case for choice. 7. Print area for given shape.

Q6.1Write a program to print the names of students by creating a Student class. If no name is passed while creating an object of Student class, then the name should be "Unknown", otherwise the name should be equal to the String value passed while creating object of Student class.

```
// Student class

class Student {

   private String name;


   // Default constructor

   public Student() {

      this.name = "Unknown";

   }


   // Parameterized constructor

   public Student(String name) {

      this.name = name;

   }


   // Method to display the student's name
```

```java
    public void displayName() {
        System.out.println("Student Name: " + name);
    }
}

// Main class for testing
public class Main {
    public static void main(String[] args) {
        // Create a student object using the default constructor
        Student student1 = new Student();

        // Create a student object using the parameterized constructor
        Student student2 = new Student("Alice");

        // Display the names of the students
        System.out.println("Using Default Constructor:");
        student1.displayName();
        System.out.println("Using Parameterized Constructor:");
        student2.displayName();
    }
}
```

1. Define a Student Class: o Create a String attribute to store the name o Define a default constructor that sets the name to "Unknown". o Define a parameterized constructor that takes a String argument and sets the name. 2. Create Objects of the Student Class: o Create a student object using the default constructor. o Create a student object using the parameterized constructor. 3. Print the Names of the Students: o Use a method to display the name of the student.

Q6.2Design a class Time With data members for hr and min sec. Provide default and Parameterized constructors. Write a program to print date and time jjava.time.format.DateTimeFormatter.

```java
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

class Time {
    private int hr;
    private int min;
    private int sec;

    // Default constructor
    public Time() {
        this.hr = 0;
        this.min = 0;
        this.sec = 0;
    }

    // Parameterized constructor
    public Time(int hr, int min, int sec) {
        this.hr = hr;
        this.min = min;
        this.sec = sec;
    }

    // Method to display time in hh:mm:ss format
    public void displayTime() {
        System.out.printf("Time: %02d:%02d:%02d\n", hr, min, sec);
    }
}

public class Main {
    public static void main(String[] args) {
        // Create an object using the default constructor
        Time defaultTime = new Time();
        System.out.println("Default Time:");
        defaultTime.displayTime();

        // Create an object using the parameterized constructor
        Time customTime = new Time(14, 45, 30);
        System.out.println("Custom Time:");
        customTime.displayTime();

        // Display the current date and time
        LocalDateTime currentDateTime = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        String formattedDateTime = currentDateTime.format(formatter);
        System.out.println("Current Date and Time: " + formattedDateTime);
    }
}
```
Define the Time Class: • Create data members hr, min, and sec to represent hours, minutes, and seconds. • Define a default constructor to initialize the time to 00:00:00. • Define a parameterized constructor to initialize the time with user-provided values. 2. Create Methods to Display Time: • Create a method that displays the time in hh:mm:ss format. 3. Use Java's Date and Time API: • Use LocalDateTime.now() to get the current date and time. • Use DateTimeFormatter to format and print the current date and time.

4. Implement the Main Method: • Create objects of the Time class using both default and parameterized constructors. • Call the display method to show the set time. • Display the current date and time using LocalDateTime and DateTimeFormatter.

Q7Student Management System using Java Inheritance. Design and implement a Student Management System using Java Inheritance. The system should manage student records by categorizing them based on their academic status (e.g.undergraduate, graduate).

```
import java.util.ArrayList;

import java.util.Scanner;


// Base Class

class Student {

   protected int studentID;

   protected String name;

   protected int age;

   protected String major;


   // Constructor

   public Student(int studentID, String name, int age, String major) {

      this.studentID = studentID;

      this.name = name;

      this.age = age;

      this.major = major;

   }


   // Method to display student information

   public void displayInfo() {

      System.out.println("Student ID: " + studentID);

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

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

      System.out.println("Major: " + major);

   }

}


// Derived Class for Undergraduate Students

class UndergraduateStudent extends Student {

   private int yearOfStudy;


   public UndergraduateStudent(int studentID, String name, int age, String major, int yearOfStudy) {

      super(studentID, name, age, major);
```

```
        this.yearOfStudy = yearOfStudy;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Year of Study: " + yearOfStudy);
    }
}

// Derived Class for Graduate Students
class GraduateStudent extends Student {
    private String researchTopic;

    public GraduateStudent(int studentID, String name, int age, String major, String researchTopic) {
        super(studentID, name, age, major);
        this.researchTopic = researchTopic;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Research Topic: " + researchTopic);
    }
}

// Class to Manage Student Records
class StudentManagementSystem {
    private ArrayList<Student> studentList;

    public StudentManagementSystem() {
        studentList = new ArrayList<>();
    }

    // Method to add a student
    public void addStudent(Student student) {
        studentList.add(student);
    }

    // Method to display all students
    public void displayAllStudents() {
        for (Student student : studentList) {
            student.displayInfo();
            System.out.println("--------------------");
        }
    }
}

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

        // Student Management System instance
        StudentManagementSystem sms = new StudentManagementSystem();

        // Adding Student Keshav and Rohit
        sms.addStudent(new UndergraduateStudent(101, "Keshav", 20, "Computer Science", 2));
        sms.addStudent(new GraduateStudent(102, "Rohit", 24, "Mechanical Engineering", "AI in Robotics"));

        // Displaying all students
        System.out.println("Student Records:");
        sms.displayAllStudents();
    }
}
```

Algorithm/Steps: 1. Define the Base Class (Student): • Create data members such as studentID, name, age, and major. • Define methods to set and display student information. 2. Define Derived Classes (UndergraduateStudent and GraduateStudent • Extend the Student class. • Add specific attributes for each subclass (e.g., yearOfStudy for undergraduates, researchTopic for graduates). • Override the methods if necessary to provide specific implementations for each category. 3.Create a Class to Manage Student Records: • Implement methods to add, remove, and display students. • Use polymorphism to handle different types of students seamlessly. 4.Implement the Main Method: • Create instances of UndergraduateStudent and GraduateStudent. • Demonstrate adding, displaying, and categorizing student records

Q8 Write a Java program to create a base class Animal with methods move() and makeSound(). Create two subclasses Bird and Panthera. Override the move() method in each subclass to describe how each animal moves. Also, override the makeSound() method in each subclass to make a specific sound for each animal.

```
// Base Class
class Animal {
    // Method to describe how an animal moves
    public void move() {

        System.out.println("The animal moves in a general way.");

    }


    // Method to describe the sound the animal makes

    public void makeSound() {

        System.out.println("The animal makes a sound.");

    }

}


// Subclass Bird

class Bird extends Animal {

    // Overriding the move method to describe how a bird moves (flying)

    @Override

    public void move() {

        System.out.println("The bird flies in the sky.");

    }


    // Overriding the makeSound method to produce a bird-specific sound (chirping)

    @Override

    public void makeSound() {

        System.out.println("The bird chirps.");

    }

}


// Subclass Panthera

class Panthera extends Animal {

    // Overriding the move method to describe how a panthera species moves (running/stalking)

    @Override

    public void move() {

        System.out.println("The panthera moves silently and swiftly, stalking its prey.");

    }


    // Overriding the makeSound method to produce a panthera-specific sound (roaring)

    @Override

    public void makeSound() {

        System.out.println("The panthera roars loudly.");
    }
}

// Main Class to test the program
public class Main {
    public static void main(String[] args) {
        // Create objects of Bird and Panthera
        Animal bird = new Bird();
        Animal panthera = new Panthera();

        // Demonstrating polymorphism
        System.out.println("Bird:");
        bird.move();    // Calls Bird's move()
        bird.makeSound();  // Calls Bird's makeSound()

        System.out.println("\nPanthera:");
        panthera.move();  // Calls Panthera's move()
        panthera.makeSound();  // Calls Panthera's makeSound()
    }
}
```

1. Define the Base Class (Animal): • Create two methods: move() and makeSound(). • Define these methods with general descriptions. 2. Define the Subclass Bird: • Extend the Animal class. • Override the move() method to describe how a bird moves (e.g., flying). • Override the makeSound() method to produce a bird-specific sound. 3. Define the Subclass Panthera: • Extend the Animal class. • Override the move() method to describe how a panthera species moves (e.g., running or stalking). • Override the makeSound() method to produce a specific sound (e.g., roaring). 4. Create a Main Class to Test the Program: • Create objects of Bird and Panthera. • Use the move() and makeSound() methods to demonstrate polymorphic behavior.

Q9 The aim of the Java practical exercise is to create a program that reads a list of numbers from a file and throws an exception if any of the numbers are positive.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

// Step 1: Create the custom exception class
class PositiveNumberException extends Exception {
    public PositiveNumberException(String message) {
        super(message);  // Passing the message to the superclass Exception
    }
}

// Main class to execute the program
```

```
public class Main {

   // Step 2: Method to read numbers from a file

   public static void readNumbersFromFile(String filename) {

      try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

         String line;

         while ((line = br.readLine()) != null) {

            try {

               int number = Integer.parseInt(line);  // Convert line to integer


               // Step 3: Check for positive numbers and throw exception

               if (number > 0) {

                  throw new PositiveNumberException("Positive number encountered: " + number);

               } else {

                  System.out.println("Number: " + number);  // Print non-positive numbers

               }

            } catch (PositiveNumberException e) {

               // Step 4: Handle the PositiveNumberException

               System.out.println(e.getMessage());

               return;  // Stop further processing if positive number is encountered

            } catch (NumberFormatException e) {

               System.out.println("Invalid number format: " + line);

            }

         }

      } catch (IOException e) {

         System.out.println("Error reading file: " + e.getMessage());

      }

   }


   // Step 5: Main method to execute the program

   public static void main(String[] args) {

      // Call the method to read numbers from a file (pass the filename here)

      String filename = "numbers.txt";  // Assuming the file 'numbers.txt' exists

      readNumbersFromFile(filename);

   }

}
```

Algorithm: 1. Create a Custom Exception Class: • Define a class PositiveNumberException that extends the Exception class. • Add a constructor that takes a message as an argument and passes it to the Exception superclass. 2. Create a Method to Read Numbers from a File: • Use BufferedReader to read the contents of the file line by line. • Convert each line to an integer. 3. Check for Positive Numbers: • If a positive number is encountered, throw a PositiveNumberException. • If all numbers are non-positive, print them. MIT

ADT University, Pune 4. Implement Exception Handling: • Use try-catch blocks to catch and handle IOException for file reading issues. • Catch and handle PositiveNumberException to display an appropriate message. 5. Create a Main Class to Execute the Program: • Call the method that reads the file and handles exceptions appropriately.

Q10a Write a Java program to remove the third element from an array list.

```
import java.util.ArrayList;


public class RemoveThirdElement {

  public static void main(String[] args) {

    // Step 2: Initialize the ArrayList with five string elements

    ArrayList<String> elements = new ArrayList<>();

    elements.add("Apple");

    elements.add("Banana");

    elements.add("Cherry");

    elements.add("Date");

    elements.add("Elderberry");


    // Step 3: Display the Original ArrayList

    System.out.println("Original ArrayList: " + elements);


    // Step 4: Check if ArrayList has at least 3 elements

    if (elements.size() >= 3) {

      // Step 5: Remove the third element (index 2)

      elements.remove(2);

    } else {

      System.out.println("The ArrayList doesn't have enough elements to remove the third one.");

    }


    // Step 6: Display the Modified ArrayList

    System.out.println("Modified ArrayList after removing the third element: " + elements);

  }

}
```

1. Import the ArrayList class: The program starts by importing the ArrayList class from the java.util package. 2. Initialize the ArrayList: An ArrayList named elements is initialized and populated with five string elements. 3. Display the Original ArrayList: The original contents of the ArrayList are printed to the console. 4. Check ArrayList Size: The program checks if the ArrayList has at least three elements to ensure there is a third element to remove. 5. Remove the Third Element: The remove method is used to remove the element at index 2 (the third element, as the index is zero-based). 6. Display the Modified ArrayList: The contents of the ArrayList after removing the third element are printed to the console.

Q10b Write a Java program to insert the specified element at the specified position in the linked list.

```
import java.util.LinkedList;
```

```
public class InsertElementInLinkedList {
    public static void main(String[] args) {
        // Step 2: Create a LinkedList Object
        LinkedList<String> list = new LinkedList<>();

        // Step 3: Add Elements to the List
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");
        list.add("Date");

        // Step 4: Print the list before insertion
        System.out.println("Original LinkedList: " + list);

        // Step 5: Insert the specified element at a specified position
        int index = 2;  // Specify the position (e.g., index 2)
        String element = "Mango";  // Element to insert

        try {
            // Insert the element at the specified index
            list.add(index, element);

            // Step 6: Print the list after insertion
            System.out.println("LinkedList after insertion: " + list);
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Invalid index: " + index);
        }
    }
}
```

Algorithm: 1. Import the LinkedList Class: • Import java.util.LinkedList and other required classes. 2. Create a LinkedList Object: • Initialize a LinkedList of elements (e.g., integers or strings). 3. Add Elements to the List: • Use the add() method to populate the list with some initial values. 4. Insert the Specified Element at a Specific Position: • Use the add(int index, E element) method to insert an element at the desired index. 5. Print the List: • Display the list before and after the insertion to show the changes. 6. Handle Index Exceptions: • Use exception handling to manage scenarios where an invalid index is provided.

Q10c Write a Java program to get the number of elements in a hash set.

```
import java.util.HashSet;


public class HashSetSize {
    public static void main(String[] args) {
        // Step 2: Create a HashSet Object

        HashSet<String> set = new HashSet<>();


        // Step 3: Add Elements to the Set

        set.add("Apple");

        set.add("Banana");

        set.add("Cherry");

        set.add("Date");


        // Adding duplicate element

        set.add("Apple");


        // Step 4: Retrieve the Number of Elements

        int size = set.size();


        // Step 5: Print the Result

        System.out.println("The number of elements in the HashSet is: " + size);

    }

}
```

Algorithm: 1. Import Required Classes: • Import java.util.HashSet. 2. Create a HashSet Object: • Initialize a HashSet with a suitable data type (e.g., String, Integer). 3. Add Elements to the Set: • Use the add() method to insert elements into the HashSet. 4. Retrieve the Number of Elements: • Use the size() method to get the number of elements present in the HashSet. 5. Print the Result: • Display the total number of elements in the set. 6. Handle Duplicates: • Understand that adding duplicate elements will not increase the size of the HashSet, as duplicates are automatically discarded.