**EX:NO:3**

**A)**

**DATE:**

**INHERITANCE AND INTERFACE**

**1.Write a Java program to create a class called Employee with methods called work() and getSalary(). Create a subclass called HRManager that overrides the work() method and adds a new method called addEmployee().**

**AIM:**

To write a java program to perform the condition using inheritance and interface.

**PROGRAM:**

class Employee { public void work() {

System.out.println("Employee is working");

}

public double getSalary() { return 50000;

}

} class HRManager extends Employee {

@Override public void work() {

System.out.println("HR Manager is overseeing human resources.");

}

public void addEmployee(String name) {

System.out.println(name + " has been added as an employee.");

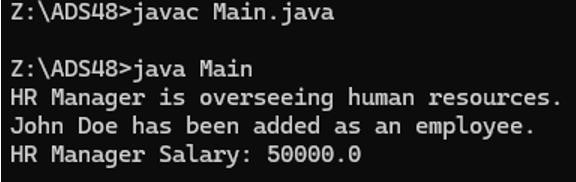
}

} public class Main { public static void main(String[] args) { HRManager hrManager = new HRManager(); hrManager.work(); hrManager.addEmployee("John Doe"); System.out.println("HR Manager Salary: " + hrManager.getSalary());

}

}

**OUTPUT:**



**2.Write a Java program to create a class known as Person with methods called getFirstName() and getLastName(). Create a subclass called Employee that adds a new method named getEmployeeId() and overrides the getLastName() method to include the employee's job title.**

**PROGRAM:**

class Person { private String firstName; private String lastName; public Person(String firstName, String lastName) { this.firstName = firstName; this.lastName = lastName;

}

public String getFirstName() { return firstName;

}

public String getLastName() { return lastName;

}

} class Employee extends Person { private String employeeId; public Employee(String firstName, String lastName, String employeeId) { super(firstName, lastName); this.employeeId = employeeId;

}

@Override public String getLastName() { return super.getLastName() + " - Job Title: Employee";

}

public String getEmployeeId() { return employeeId;

}

} public class Main2 { public static void main(String[] args) {

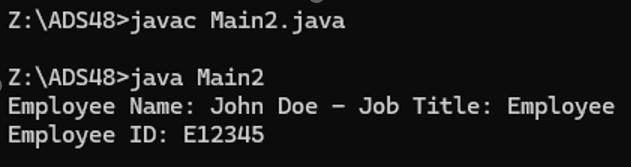
Employee emp = new Employee("John", "Doe", "E12345");

System.out.println("Employee Name: " + emp.getFirstName() + " " + emp.getLastName()); System.out.println("Employee ID: " + emp.getEmployeeId());

}

}

**OUTPUT:**



**3.Write a Java program to create a vehicle class hierarchy. The base class should be Vehicle, with subclasses Truck, Car and Motorcycle. Each subclass should have properties such as make, model, year, and fuel type**. **Implement methods for calculating fuel efficiency, distance traveled, and maximum speed.**

**PROGRAM:**

class Vehicle { protected String make; protected String model; protected int year; protected String fuelType; public Vehicle(String make, String model, int year, String fuelType) { this.make = make; this.model = model; this.year = year; this.fuelType = fuelType;

}

public double calculateFuelEfficiency() { return 15.0;

}

public double calculateDistanceTraveled(double fuelAmount) { return calculateFuelEfficiency() \* fuelAmount;

}

public int getMaxSpeed() { return 120;

}

} class Truck extends Vehicle { public Truck(String make, String model, int year, String fuelType) { super(make, model, year, fuelType)

}

@Override public int getMaxSpeed() { return 100;

}

} class Car extends Vehicle { public Car(String make, String model, int year, String fuelType) { super(make, model, year, fuelType);

}

@Override public int getMaxSpeed() { return 150;

}

} class Motorcycle extends Vehicle { public Motorcycle(String make, String model, int year, String fuelType) { super(make, model, year, fuelType);

}

@Override public int getMaxSpeed() { return 180;

}

} public class Main3 { public static void main(String[] args) {

Vehicle truck = new Truck("Ford", "F-150", 2023, "Diesel");

System.out.println("Truck Max Speed: " + truck.getMaxSpeed()); System.out.println("Truck Distance with 10 gallons: " + truck.calculateDistanceTraveled(10));

Vehicle car = new Car("Toyota", "Camry", 2022, "Gasoline");

System.out.println("Car Max Speed: " + car.getMaxSpeed());

System.out.println("Car Distance with 10 gallons: " + car.calculateDistanceTraveled(10));

Vehicle motorcycle = new Motorcycle("Harley", "Sportster", 2022, "Gasoline");

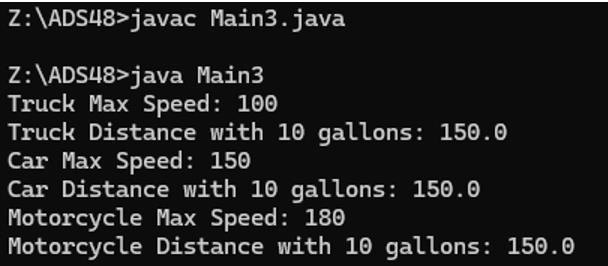
System.out.println("Motorcycle Max Speed: " + motorcycle.getMaxSpeed());

System.out.println("Motorcycle Distance with 10 gallons: " + motorcycle.calculateDistanceTraveled(10));

}

}

**OUTPUT:**



**4.Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and**

**Basketball that implement the Playable interface and override the play() method to play the respective sports.**

**PROGRAM:**

interface Playable { void play();

} class Football implements Playable {

@Override public void play() {

System.out.println("Playing football.");

}

} class Volleyball implements Playable {

@Override public void play() {

System.out.println("Playing volleyball.");

}

} class Basketball implements Playable {

@Override public void play() {

System.out.println("Playing basketball.");

}

} public class Main4 { public static void main(String[] args) {

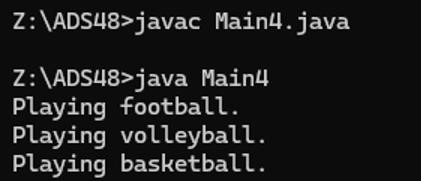
Playable football = new Football();

Playable volleyball = new Volleyball(); Playable basketball = new Basketball(); football.play(); volleyball.play(); basketball.play();

}

}

**OUTPUT:**



**5.Write a Java program to create an interface Drawable with a method draw() that takes no arguments and returns void. Create three classes Circle, Rectangle, and Triangle that implement the Drawable interface and override the draw() method to draw their respective shapes**

**PROGRAM:**

interface Drawable { void draw();

} class Circle implements Drawable {

@Override public void draw() {

System.out.println("Drawing a Circle.");

}

} class Rectangle implements Drawable {

@Override public void draw() {

System.out.println("Drawing a Rectangle.");

}

} class Triangle implements Drawable {

@Override public void draw() {

System.out.println("Drawing a Triangle.");

}

} public class solution { public static void main(String[] args) {

Drawable circle = new Circle();

Drawable rectangle = new Rectangle();

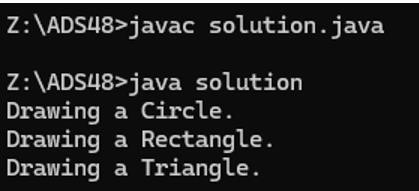
Drawable triangle = new Triangle();

circle.draw(); rectangle.draw(); triangle.draw();

}

}

**OUTPUT:**



**RESULT:**

Thus the program to perform inheritance and interface is executed successfully.

**EX.NO:3b)**

**METHOD OVERRIDING**

**DATE:**

**AIM:**

To write a program to perform the conditions using method overriding.

**1. Java Method Overriding**

When a subclass inherits from a superclass, it also inherits its methods; however, it can also override the superclass methods (as well as declare and implement new ones). Consider the following Sports class:

class Sports{

String getName(){ return "Generic Sports";

}

void getNumberOfTeamMembers(){

System.out.println( "Each team has n players in " + getName() );

}

}

Next, we create a Soccer class that inherits from the Sports class. We can override the getName method and return a different, subclass-specific string:

class Soccer extends Sports{

@Override

String getName(){ return "Soccer

Class";

}

}

**Note:** When overriding a method, you should precede it with the @Override annotation. The parameter(s) and return type of an overridden method must be exactly the same as those of the method inherited from the supertype.

**Task:**

Complete the code in your editor by writing an

overridden getNumberOfTeamMembers method that prints the same statement as the

superclass' getNumberOfTeamMembers method, except that it replaces with (the number of players on a Soccer team).

**Output Format:**

When executed, your completed code should print the following:

Generic Sports

Each team has n players in Generic Sports

Soccer Class

Each team has 11 players in Soccer Class

**PROGRAM:**

import java.util.\*; class Sports{

String getName(){ return "Generic Sports";

}

void getNumberOfTeamMembers(){

System.out.println( "Each team has n players in " + getName() );

} }

class Soccer extends Sports{

@Override

String getName(){ return "Soccer Class";

}

@Override

void getNumberOfTeamMembers() {

System.out.println("Each team has 11 players in " + getName());

}

}

public class Solution{

public static void main(String []args){

Sports c1 = new Sports();

Soccer c2 = new Soccer(); System.out.println(c1.getName()); c1.getNumberOfTeamMembers(); System.out.println(c2.getName()); c2.getNumberOfTeamMembers();

} }

**OUTPUT:**

Generic Sports

Each team has n players in Generic Sports

Soccer Class

Each team has 11 players in Soccer Class

**2. Java Method Overriding 2 (Super Keyword)**

When a method in a subclass overrides a method in superclass, it is still possible to call the overridden method using **super** keyword. If you write super.func() to call the function func(), it will call the method that was defined in the superclass.

You are given a partially completed code in the editor. Modify the code so that the code prints the following text:

Hello I am a motorcycle, I am a cycle with an engine.

My ancestor is a cycle who is a vehicle with pedals.

**PROGRAM:**

import java.util.\*; import java.io.\*; class BiCycle{

String define\_me(){ return "a vehicle with pedals."; }

}

class MotorCycle extends BiCycle{

String define\_me(){ return "a cycle with an engine.";

}

MotorCycle(){

System.out.println("Hello I am a motorcycle, I am "+ define\_me());

String temp = super.define\_me();

System.out.println("My ancestor is a cycle who is "+ temp );

} }

class Solution{ public static void main(String []args){

MotorCycle M=new MotorCycle();

}

}

**OUTPUT:**

Hello I am a motorcycle, I am a cycle with an engine.

My ancestor is a cycle who is a vehicle with pedals.

**3. Java Inheritance I**

Using inheritance, one class can acquire the properties of others. Consider the following Animal class:

class Animal{ void walk(){

System.out.println("I am walking"); }

}

This class has only one method, walk. Next, we want to create a Bird class that also has a fly method. We do this using extends keyword:

class Bird extends Animal { void fly() {

System.out.println("I am flying"); }

}

Finally, we can create a Bird object that can both fly and walk. public class Solution{

public static void main(String[] args){ Bird bird = new Bird(); bird.walk(); bird.fly();

}

}

The above code will print:

I am walking

I am flying

This means that a Bird object has all the properties that an Animal object has, as well as some additional unique properties.

The code above is provided for you in your editor. You must add a sing method to the Bird class, then modify the main method accordingly so that the code prints the following lines:

I am walking

I am flying

I am singing

**PROGRAM:**

import java.io.\*; import java.util.\*; import java.text.\*; import java.math.\*; import java.util.regex.\*; class Animal{ void walk(){

System.out.println("I am walking");

} }

class Bird extends Animal {

void fly() {

System.out.println("I am flying");

} void sing() {

System.out.println("I am singing");

} } public class Solution{

public static void main(String args[]){ Bird bird = new Bird(); bird.walk(); bird.fly();

bird.sing();

}

}

**OUTPUT:**

I am walking

I am flying

I am singing

**4. Java Inheritance II**

Write the following code in your editor below:

A class named Arithmetic with a method named add that takes integers as parameters and returns an integer denoting their sum.

A class named Adder that inherits from a superclass named Arithmetic.

Your classes should not be be .

**Input Format:**

You are not responsible for reading any input from stdin; a locked code stub will test your submission by calling the add method on an Adder object and passing it integer parameters.

**Output Format:**

You are not responsible for printing anything to stdout. Your add method must return the sum of its parameters.

**Sample Output:**

The main method in the Solution class above should print the following:

My superclass is: Arithmetic

42 13 20

**PROGRAM:**

import java.io.\*; import java.util.\*; import java.text.\*; import java.math.\*; import java.util.regex.\*; class Arithmetic { int add(int a, int b) { return a + b;

} }

class Adder extends Arithmetic {

}

public class Solution{

public static void main(String []args){ Adder a = new Adder();

System.out.println("My superclass is: " + a.getClass().getSuperclass().getName());

System.out.print(a.add(10,32) + " " + a.add(10,3) + " " + a.add(10,10) + "\n");

}

}

**OUTPUT:**

My superclass is : Arithmetic

42 13 20

**5. Remove Duplicates -Method Overloading**

Method should have 1) public as its Access specifier 2) remDup as its name 3) return type is String for one method and Integer array for another method Let arr[]={1,2,3,1,4,3} arr is passed to remDup where duplicate elements are removed then return {1,2,3,4} Similarly str="happy" str is passed to remDup where duplicate elements are removed then return hapy **Input Format:**

N denotes length of array arr is array of Integers str is a String which is in lowercase **Constraints:**

--

**Output Format:** one method returns Integer array Another return String **Sample Input:**

5 1 2 3 4 3 three

**Sample Output:**

[1, 2, 3, 4] thre

**PROGRAM:**

import java.io.\*; import java.util.\*; import java.text.\*; import java.math.\*; import java.util.regex.\*; class RmvDup{

public Integer[] remDup(int[] arr) {

LinkedHashSet<Integer> set = new LinkedHashSet<>(); for (int num : arr) { set.add(num);

}

return set.toArray(new Integer[0]);

}

public String remDup(String str) {

StringBuilder result = new StringBuilder(); HashSet<Character> seen = new HashSet<>(); for (char ch : str.toCharArray()) {

if (!seen.contains(ch)) {

result.append(ch); seen.add(ch);

} }

return result.toString();

} }

public class Solution {

public static void main(String[] args) { Scanner on=new Scanner(System.in); int N=on.nextInt(); int arr[]=new int[N]; for(int i=0;i<N;i++)

{ arr[i]=on.nextInt();

}

String str=on.next();

RmvDup obj=new RmvDup();

System.out.println(Arrays.toString(obj.remDup(arr)));

System.out.println(obj.remDup(str));

} } **OUTPUT:**

[1,2,3,4] thre

**RESULT:**

Thus the program to perform the conditions using method overriding is executed successfully.

**EX**

**.**

**NO:4**

**DATE:**

**STRINGS**

1. **Write a Java program to get the length of a given string. Sample Output: The string length of 'example.com' is: 11 AIM:**

To write a java program to perform the condition using strings.

**PROGRAM:**

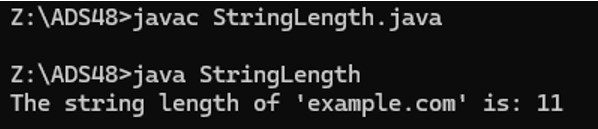
class StringLength { public static void main(String[] args) { String str = "example.com"; int length = str.length();

System.out.println("The string length of '" + str + "' is: " + length);

}

}

**OUTPUT:**



1. **Write a Java program to get the character at the given index within the string. Sample Output: Original String = Java Exercises! The character at position 0 is J The character at position 10 is i**

**PROGRAM:**

class ch { public static void main(String[] args) {

String str = "Java Exercises!";

System.out.println("Original String = " + str);

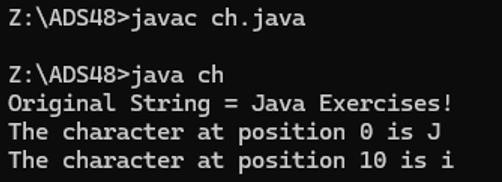
System.out.println("The character at position 0 is " + str.charAt(0));

System.out.println("The character at position 10 is " + str.charAt(10));

}

}

**OUTPUT:**



1. **Write a Java program to compare two strings, ignoring case differences. Sample Output: String 1: This is exercise 4 String 2: This is Exercise 4 "This is exercise 1" is equal to "This is Exercise 1"**

**PROGRAM:**

class Compare {

public static void main(String[] args) { String str1 = "This is exercise 4"; String str2 = "This is Exercise 4"; if (str1.equalsIgnoreCase(str2)) {

System.out.println("\"" + str1 + "\" is equal to \"" + str2 + "\"");

} else {

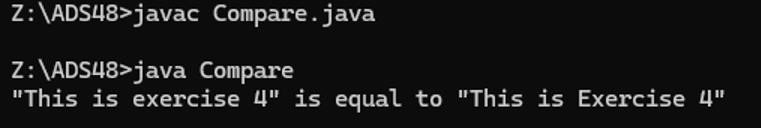
System.out.println("\"" + str1 + "\" is not equal to \"" + str2 + "\"");

}

}

}

**OUTPUT:**



**4. Write a Java program to find out whether a region in the current string matches a region in another string. Sample Output: str1[0 - 7] == str2[28 - 35]? true str1[9 - 15] == str2[9 - 15]? false**

**PROGRAM:**

class StringRegionMatch {

public static void main(String[] args) { String str1 = "This is a test string"; String str2 = "And here is a test string"; boolean match1 = str1.regionMatches(0, str2, 11, 8);

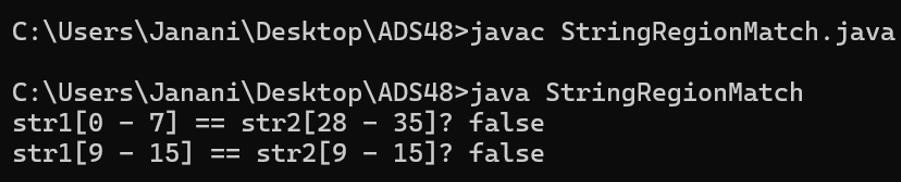
System.out.println("str1[0 - 7] == str2[28 - 35]? " + match1); boolean match2 = str1.regionMatches(9, str2, 9, 7);

System.out.println("str1[9 - 15] == str2[9 - 15]? " + match2);

}

}

**OUTPUT:**



**5. Write a Java program to replace each substring of a given string that matches the given regular expression with the given replacement. Sample string : "The quick brown fox jumps over the lazy dog." In the above string replace all the fox with cat. Sample Output: Original string: The quick brown fox jumps over the lazy dog. New String: The quick brown cat jumps over the lazy dog.**

**PROGRAM:**

class ReplaceSubstring { public static void main(String[] args) {

String originalString = "The quick brown fox jumps over the lazy dog.";

System.out.println("Original string: " + originalString);

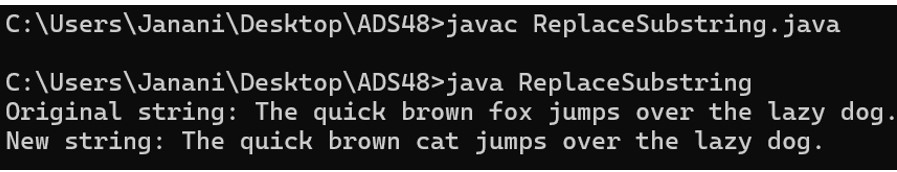
String newString = originalString.replaceAll("fox", "cat");

System.out.println("New string: " + newString);

}

}

**OUTPUT:**



**RESULT:**

Thus the program to perform string operations is executed successfully.