Title

Write a java program for employee class to display basic information .

Aim

This program demonstrates you the array of objects. Array of object is the collection of objects of the same class. The given syntax will define an array of reference therefore assign object to individual reference in the array to declare the array of object.

Objective : Demonstrate the use of array of objects.

Theory. :The syntax to define array of object is:

<class-name><array-name> [size];

For example, the following statement will define an array of object having size

6 references to the class Box.

Box b[6];

For creating an array of objects (allocating memory) you have to write down

the following statements

for (int i=0; i<6; i++) {

b[i] = new Box();

// assigning object to individual reference in the array.

}

Procedure :

public class Employee {

```
private String name ;
    private int id;
    private String department;

    // Constructor to initialize employee details
    public Employee(String name, int id, String department) {
        this.name = name;
        this.id = id;
        this.department = department;
    }

    // Method to display employee information
    public void displayInfo() {
        System.out.println("Employee Name: " + name);
        System.out.println("Employee ID: " + id);
        System.out.println("Department: " + department);
    }

    // Main method to create and display employee object
    public static void main(String[] args) {

        // Create an employee object
        Employee emp1 = new Employee("John Doe", 12345, "Engineering");

        // Display employee information
        emp1.displayInfo();
```

```
        }
    }
```

Title : Design a class in java to perform various mathematical operations on given numbers.

Aim : Java Program for Calculator - This program will read two integer numbers and calculate the arithmetic operators, in this example we used switch case and if else statement. User will enter a choice after entering two numbers and based on user choice program will return the result.

Objective : 1. Demonstrate use of arithmetic operators.

2. Demonstrate use of functions.

Theory : Arithmetic Operators

These operators involve the mathematical operators that can be used to perform various simple or advanced arithmetic operations on the primitive data types referred to as the operands. These operators consist of various unary and binary operators that can be applied on a single or two operands. Let's look at the various operators that Java has to provide under the arithmetic operators.

Operators	Result
+	Addition of two numbers
-	Subtraction of two numbers
*	Multiplication of two numbers
/	Division of two numbers
%	(Modulus Operator)Divides two numbers and returns the remainder
++	Increment Operator
--	Decrement Operator

1. **Arithmetic Operators**: Arithmetic operators are used to perform arithmetic/mathematical operations on operands.

- **Addition ('+')**: Adds two operands, like (A + B) or (8 + 3).
- **Subtraction ('-')**: Subtracts two operands, like (A – B) or (3 – 2).

- **Multiplication (‘*’)**: Multiplies two operands, like (A * B) or (5 * 2).
- **Division (‘/’)**: Divides the first operand by the second, like (A / B) or (8 / 2).
- **Modulus (‘%’)**: Returns the remainder when first operand is divided by the second.
- **Increment (‘++’)**: Increment the value of an integer. When placed before the variable name (also called pre-increment operator), its value is incremented instantly.
- **Decrement (‘–‘)**: Decrement the value of an integer. When placed before the variable name (also called pre-decrement operator), its value is decremented instantly.

Procedure :

```
import java.util.Scanner;

public class Calculator {

   public static void main(String[] args) {

      int a, b, choice;

      float result = 0;


      Scanner buf = new Scanner(System.in);

      System.out.print("Enter first number: ");

      a = buf.nextInt();

      System.out.print("Enter second number: ");

      b = buf.nextInt();


      System.out.print("\n1: Addition.\n2: Subtraction.");
```

```
System.out.print("\n3: Multiplication.\n4: Divide.");
System.out.print("\n5: Remainder.\n6: Exit.");
System.out.print("\nEnter your choice: ");
choice = buf.nextInt();

switch (choice) {
   case 1:
      result = a + b;
      break;
   case 2:
      result = a - b;
      break;
   case 3:
      result = a * b;
      break;
   case 4:
      if (b != 0) {
         result = (float) a / b;
      } else {
         System.out.println("Error: Cannot divide by zero!");
         return;
      }
      break;
```

```
            case 5:
                result = a % b;
                break;
            case 6:
                System.out.println("Exiting the program...");
                return;
            default:
                System.out.println("Invalid Choice!");
                return;
        }

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

Title : Write a java program for calculating area of circle.

Aim : Calculate area of circle.

Objective : To learn use of static methods and constructors.

Theory :

Java program to calculate or to print area of a circle in a simple method. The following Java program to print the area of a circle has been written in different methods such as static method, using constructor, Interface, inheritance with sample outputs for each program.

In mathematics, the area of a circle can be calculated using the formula:

Different Methods :

1. Static Method

2 .Using Interface

3 .Using Inheritance

4 .Using Method

5. Using Constructor

Procedure :

1. Static Method :

```
import java.util.Scanner;

class AreaOfCircle {

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

```
 Scanner s = new Scanner(System.in);
System.out.println("Enter the radius:");
            double r = s.nextDouble();
            double area = (22 * r * r) / 7;
            System.out.println("Area of Circle is: " + area);
        }
    }
```

2 .Using Interface :

```
import java.util.Scanner;
interface AreaCal {
   void circle(double r);
}
class AreaOfCircle implements AreaCal {
   double area;
   public void circle(double r) {
      area = (22 * r * r) / 7;
   }
   public static void main(String args[]) {
      AreaOfCircle x;
      Scanner s = new Scanner(System.in);
      System.out.println("Enter the radius:");
      double rad = s.nextDouble();
      x = new AreaOfCircle();
      x.circle(rad);
      System.out.println("Area of Circle is: " + x.area);
```

```
    }
}
```

3 .Using Inheritance :

```
import java.util.Scanner;
class AreaCalculation {
    double area;
    void circle(double r) {
        area = (22 * r * r) / 7;
    }
}
class AreaOfCircle extends AreaCalculation {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the radius:");
        double rad = s.nextDouble();
        AreaOfCircle a = new AreaOfCircle();
        a.circle(rad);
        System.out.println("Area of Circle is: " + a.area);
    }
}
```

4 .Using Method :

```
import java.util.Scanner;
class AreaOfCircle {
```

```
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the radius:");
        double rad = s.nextDouble();
        double a = area(rad);
        System.out.println("Area of Circle is: " + a);
    }
    static double area(double r) {
        return (22 * r * r) / 7;
    }
}
```

5. Using Constructor :

```
import java.util.Scanner;
class Area {
    double area;
    Area(double r) {
        area = (22 * r * r) / 7;
    }
}
class AreaOfCircle {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the radius:");
        double rad = s.nextDouble();
        Area a = new Area(rad);
```

```
        System.out.println("Area of Circle is: " + a.area);
    }
}
```

Title : Write a program for implementing single inheritance for student class.

Aim : To Learn about interface in java.

Objective : To implement concept of inheritance in java.

Theory :

Interface in Java :

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. Java Interface also represents the IS-A relationship.

Syntax for Java Interfaces :

```
interface {
   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

Uses of Interfaces in Java :

- *It is used to achieve total abstraction.*
- *Since java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances.*
- *Any class can extend only 1 class, but can any class implement an infinite number of interfaces.*

- *It is also used to achieve loose coupling.*
- *Interfaces are used to implement abstraction.*

Procedure :

```
// Parent class
class Person {
    String name;
    int age;
    // Constructor
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    // Method to display details
    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
// Child class inheriting from Person
class Student extends Person {
    int rollNumber;
    String course;
    // Constructor
```

```
    Student(String name, int age, int rollNumber, String course) {
        super(name, age); // Call to parent class constructor
        this.rollNumber = rollNumber;
        this.course = course;
    }
    // Method to display details including student-specific details
    void displayStudentDetails() {
        displayDetails(); // Call to parent class method
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Course: " + course);
    }
}
// Main class
public class Main {
    public static void main(String[] args) {
        // Creating an instance of Student
        Student student = new Student("Your Name",21, 27, "Computer Science");
        // Calling method to display student details
        student.displayStudentDetails();
    }
}
```

Title : Write a program for implementing multilevel inheritance for employee class.

Aim : To demonstrate the concept of multilevel inheritance.

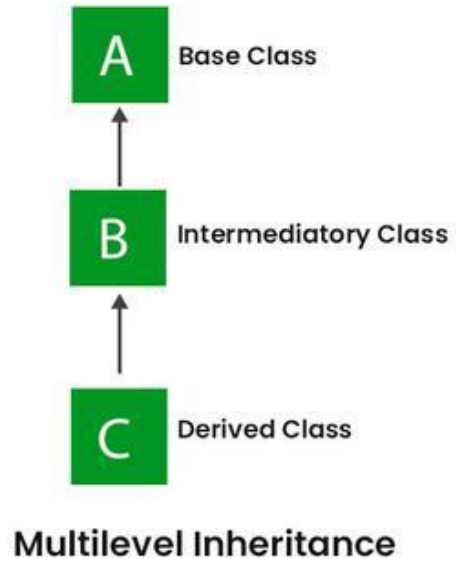
Objective : To learn the structure of multilevel inheritance.

Theory :

Multilevel inheritance in Java refers to the inheritance hierarchy where a derived class extends another derived class, forming a chain of inheritance. In other words, it involves inheriting from a class that is already inheriting from another class.

Note that the classes must be at different levels. Hence, there exists a single base class and single derived class but multiple intermediate base classes.

Example :



Multilevel Inheritance

As shown in above block diagram, class C has class B and class A as parent classes. Depending on the relation the level of inheritance can be extended to any level. As in other inheritance, based on the visibility mode used or access specifier used while

deriving, the properties of the base class are derived. Access specifier can be private, protected or public**.**

Procedure :

```
// Person.java
class Person {
 String name;
 int age;
 public Person(String name, int age) {
 this.name = name;
 this.age = age;
}
public void displayDetails() {
System.out.println("Name: " + name);
System.out.println("Age: " + age);
}
}
// Employee.java (Child of Person)
class Employee extends Person {
    String employeeId;
    double salary;
    public Employee(String name, int age, String employeeId, double salary) {
        super(name, age);
        this.employeeId = employeeId;
        this.salary = salary;
    }
    public void displayEmployeeDetails() {
        displayDetails();
```

```
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Salary: $" + salary);
    }
}
// Manager.java (Child of Employee)
class Manager extends Employee {
    String department;
    public Manager(String name, int age, String employeeId, double salary, String
department) {
        super(name, age, employeeId, salary);
        this.department = department;
    }
    public void displayManagerDetails() {
        displayEmployeeDetails();
        System.out.println("Department: " + department);
    }
}
public class Main {
    public static void main(String[] args) {
        Manager manager = new Manager("Your Name", 25, "PRN", 50000, "Engg");
        manager.displayManagerDetails();
    }
}
```

Title: Implementing Java Program to Display Content of Array

Aim: To study the use of arrays in Java programming.

Objective: To demonstrate the concept of arrays in Java and how to display their content.

Introduction: An array is a data structure that stores a fixed-size sequential collection of elements of the same type. It is used to store multiple values of the same type under a single variable name. Each item in an array is called an element, and each element is accessed by its numerical index.

In this lab, we will implement a Java program to create an array, initialize it with some values, and then display the content of the array.

Java Program:

java

```
// Importing required packages
import java.util.Arrays;

// Main class
public class DisplayArrayContent {
    // Main method
    public static void main(String[] args) {
        // Define and initialize an array
        int[] numbers = {10, 20, 30, 40, 50};

        // Displaying content of the array
        System.out.println("Content of the array:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("Element at index " + i + ": " +
numbers[i]);
        }
    }
}
```

Explanation:

1. We import the `java.util.Arrays` package to use utility methods related to arrays.
2. We define a class named `DisplayArrayContent`.
3. Inside the class, we define the `main` method, which is the entry point of our program.
4. We declare and initialize an array named `numbers` of type `int`. The array is initialized with some values `{10, 20, 30, 40, 50}`.
5. We then display the content of the array using a `for` loop. The loop iterates through each element of the array using the index `i` from `0` to `numbers.length - 1`. Inside the loop, we print the index and the value of each element.

Certainly! Below are two alternative ways to implement a Java program to display the content of an array:

Alternative 1: Using Enhanced for Loop

java

```
public class DisplayArrayContent {
    public static void main(String[] args) {
        // Define and initialize an array
        int[] numbers = {10, 20, 30, 40, 50};

        // Displaying content of the array using enhanced for loop
        System.out.println("Content of the array:");
        for (int number : numbers) {
            System.out.println("Element: " + number);
        }
    }
}
```

Explanation:

- In this alternative, we use an enhanced for loop (also known as a for-each loop) to iterate through the elements of the array.
- The loop automatically iterates over each element of the array `numbers`, assigning the value of each element to the variable `number`.
- Inside the loop, we print each element.

Alternative 2: Using Arrays.toString() Method

java

```
import java.util.Arrays;

public class DisplayArrayContent {
    public static void main(String[] args) {
        // Define and initialize an array
        int[] numbers = {10, 20, 30, 40, 50};

        // Displaying content of the array using Arrays.toString()
method
        System.out.println("Content of the array:");
        System.out.println(Arrays.toString(numbers));
    }
}
```

Explanation:

- In this alternative, we use the `Arrays.toString()` method from the `java.util.Arrays` package to directly print the content of the array.
- The `Arrays.toString()` method converts the array `numbers` into a string representation, where each element of the array is separated by commas and enclosed within square brackets.
- We simply print the result returned by the `Arrays.toString()` method.

Both alternatives achieve the same objective of displaying the content of the array, but they use different approaches to iterate through and print the elements of the array.

Title: Finding Prime Numbers from 1 to 20 in Java

Aim: To find prime numbers using Java programming.

Objective: To demonstrate the use of loops in Java programming.

Introduction: Prime numbers are natural numbers greater than 1 that have no positive divisors other than 1 and themselves. In this lab, we will write a Java program to find prime numbers from 1 to 20.

Java Program:

java

```
// Main class
public class PrimeNumbers {
    // Method to check if a number is prime
    static boolean isPrime(int num) {
        if (num <= 1)
            return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0)
                return false;
        }
        return true;
    }

    // Main method
    public static void main(String[] args) {
        System.out.println("Prime numbers from 1 to 20:");
        for (int i = 1; i <= 20; i++) {
            if (isPrime(i))
                System.out.print(i + " ");
        }
    }
}
```

Explanation:

1. We define a class named `PrimeNumbers`.

2. Inside the class, we define a method `isPrime(int num)` to check if a given number is prime.
    - We first check if the number is less than or equal to 1. If so, we return `false` as numbers less than or equal to 1 are not prime.
    - We then loop from 2 to the square root of the number (`Math.sqrt(num)`). If the number is divisible by any integer in this range, it is not prime, and we return `false`.
    - If the number is not divisible by any integer in the range, we return `true`.
3. In the `main` method, we print a message indicating the range of numbers we are checking for prime numbers (from 1 to 20).
4. We then use a `for` loop to iterate through numbers from 1 to 20.
    - Inside the loop, we call the `isPrime` method for each number.
    - If the number is prime (the `isPrime` method returns `true`), we print it.

Conclusion: In this lab, we have learned how to implement a Java program to find prime numbers from 1 to 20 using loops. We used a method to check if a number is prime and then iterated through a range of numbers to find and print the prime numbers. This demonstrates the use of loops and methods in Java programming.

Title: Write a java program to display Fibonacci series of any number. Aim: To implement a Java program to display the Fibonacci series of any number.

Objective: To understand and implement the concept of generating Fibonacci series using Java programming.

Introduction: The Fibonacci series is a sequence of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1. In this lab, we will write a Java program to display the Fibonacci series of any number provided by the user.

Java Program:

java

```
import java.util.Scanner;

// Main class
public class FibonacciSeries {
    // Method to display Fibonacci series
    static void displayFibonacci(int n) {
        int first = 0, second = 1, next;

        System.out.println("Fibonacci series of " + n + ":");
        System.out.print(first + " " + second + " ");

        for (int i = 2; i < n; i++) {
            next = first + second;
            System.out.print(next + " ");
            first = second;
            second = next;
        }
    }

    // Main method
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of terms: ");
        int n = scanner.nextInt();
        scanner.close();
```

```
        displayFibonacci(n);
    }
}
```

Explanation:

1. We import the `java.util.Scanner` class to take input from the user.
2. We define a class named `FibonacciSeries`.
3. Inside the class, we define a method `displayFibonacci(int n)` to display the Fibonacci series up to `n` terms.
    - We initialize variables `first` and `second` to 0 and 1 respectively, which are the first two terms of the Fibonacci series.
    - We print the first two terms.
    - Using a `for` loop, we calculate and print the next terms of the Fibonacci series until we reach `n` terms.
        - The next term is the sum of the previous two terms (`next = first + second`).
        - We update `first` and `second` for the next iteration.
4. In the `main` method, we create a `Scanner` object to take input from the user for the number of terms in the Fibonacci series.
5. We prompt the user to enter the number of terms and store the input in the variable `n`.
6. We then call the `displayFibonacci` method with the number of terms provided by the user.

Conclusion: In this lab, we have learned how to implement a Java program to display the Fibonacci series of any number of terms. We used a loop to generate the series and explained the concept of swapping variables to calculate the next term in the series. This demonstrates the use of loops and user input in Java programming.

Title: Performing Addition of Two Numbers in Java

Aim: To implement a Java program to perform the addition of two numbers and accept the numbers from the user.

Objective: To understand and demonstrate the concept of accepting input from the user in Java programming.

Introduction: In Java programming, we can accept input from the user using the Scanner class, which is available in the java.util package. In this lab, we will write a Java program to accept two numbers from the user and perform their addition.

Java Program:

java

```
import java.util.Scanner;

// Main class
public class AdditionOfTwoNumbers {
    // Main method
    public static void main(String[] args) {
        // Creating Scanner object to accept input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompting the user to enter the first number
        System.out.print("Enter the first number: ");
        // Accepting the first number from the user
        double num1 = scanner.nextDouble();

        // Prompting the user to enter the second number
        System.out.print("Enter the second number: ");
        // Accepting the second number from the user
        double num2 = scanner.nextDouble();

        // Closing the Scanner object
        scanner.close();

        // Calculating the sum of the two numbers
```

```
        double sum = num1 + num2;

        // Displaying the result
        System.out.println("Sum of " + num1 + " and " + num2 + "
is: " + sum);
    }
}
```

Explanation:

1. We import the `java.util.Scanner` class to accept input from the user.
2. We define a class named `AdditionOfTwoNumbers`.
3. Inside the class, we define the `main` method, which is the entry point of our program.
4. We create a Scanner object named `scanner` to accept input from the user.
5. We prompt the user to enter the first number by displaying the message "Enter the first number: ".
6. We accept the first number entered by the user using the `nextDouble()` method of the Scanner class and store it in the variable `num1`.
7. Similarly, we prompt the user to enter the second number by displaying the message "Enter the second number: ".
8. We accept the second number entered by the user using the `nextDouble()` method and store it in the variable `num2`.
9. We close the Scanner object to release the resources.
10. We calculate the sum of the two numbers and store it in the variable `sum`.
11. Finally, we display the result by printing the sum of the two numbers.

Conclusion: In this lab, we have learned how to implement a Java program to perform the addition of two numbers and accept the numbers from the user using the Scanner class. This demonstrates the concept of accepting input from the user in Java programming.