Practical 11: WAP to demonstrate concept of ``this``.

Code:

**//this is a reference variable that refers to the current object.**

class Student{

int rollno;

String name;

float fee;

Student(int rollno,String name,float fee){

**//to refer current class instance variable**

this.rollno=rollno;

this.name=name;

this.fee=fee;

**//to invoke current class method**

this.display();

}

Student(int rollno, String name) {

**//to invoke current class constructor**

this(rollno, name, 0.0f);

}

Student getInstance() {

**//this keyword can be used to return current class instance**

return this;

}

void S\_name(Student obj){

System.out.println(obj.name);

}

void showName(){

//to pass as an argument in the method

S\_name(this);

}

void createNewStudent(){

**// Using "this" to pass as an argument in the constructor call**

Student s = new Student(this.rollno + 1, "New Student", this.fee + 1000);

System.out.println("New Student created:");

s.display();

}

void display(){System.out.println(rollno+" "+name+" "+fee);}

}

class Q11{

public static void main(String args[]){

Student s1=new Student(111,"ankit",5000f);

Student s2=new Student(112,"sumit",6000f);

s2.showName();

s1.createNewStudent();

}}

Output:

111 ankit 5000.0

112 sumit 6000.0

sumit

112 New Student 6000.0

New Student created:

112 New Student 6000.0

Practical 12: WAP to demonstrate multi-level and hierarchical inheritance.

Code:

**//Multilevel Inheritance: A subclass inherits from a superclass, and another subclass inherits from this subclass, forming a chain of inheritance.**

**//Hierarchical Inheritance: Multiple subclasses inherit from a single superclass, creating a hierarchy of classes.**

// Base class

class Animal {

    void eat() {

        System.out.println("Animal is eating");

    }

}

// Derived class inheriting from Animal

class Dog extends Animal {

    void bark() {

        System.out.println("Dog is barking");

    }

}

**//Multilevel inheritance**

// Derived class inheriting from Dog

class Labrador extends Dog {

    void color() {

        System.out.println("Labrador is brown in color");

    }

}

**//hierarchical inheritance**

// Another derived class inheriting from Animal

class Cat extends Animal {

    void meow() {

        System.out.println("Cat is meowing");

    }

}

public class Q12{

    public static void main(String[] args) {

        // Creating objects

        Labrador labrador = new Labrador();

        Cat cat = new Cat();

        // Demonstrating multi-level inheritance

        labrador.eat(); // Inherited from Animal

        labrador.bark(); // Inherited from Dog

        labrador.color(); // Specific to Labrador

        System.out.println();

        // Demonstrating hierarchical inheritance

        cat.eat(); // Inherited from Animal

        cat.meow(); // Specific to Cat

    }

}

Output:

Animal is eating

Dog is barking

Labrador is brown in color

Animal is eating

Cat is meowing

Practical 13: WAP to demonstrate multi-level and hierarchical inheritance.

Code:

**//The super keyword in Java is used in subclasses to access superclass**

members (attributes, constructors and methods).

class Superclass {

    String message = "This is the superclass";

    Superclass() {

        System.out.println("Superclass constructor called");

    }

    void display() {

        System.out.println("Display method in superclass");

    }

}

class Subclass extends Superclass {

    String message = "This is the subclass";

    Subclass() {

**//Use of super() to access superclass constructor**

        super();

        System.out.println("Subclass constructor called");

    }

    void display() {

**// Access Overridden Methods of the superclass**

        super.display();

        System.out.println("Display method in subclass");

    }

    void printMessage() {

**//Access Attributes of the Superclass**

        System.out.println(super.message);

        System.out.println(this.message);

// Access attribute of subclass

    }

}

public class Q13 {

    public static void main(String[] args) {

        Subclass obj = new Subclass();

        obj.display(); // Calls Subclass's display

        obj.printMessage(); // Prints messages from both classes

    }

}

Output:

Superclass constructor called

Subclass constructor called

Display method in superclass

Display method in subclass

This is the superclass

This is the subclass

Practical 14: WAP to demonstrate run-time polymorphism.

Code:

**// Method overriding is an example of runtime polymorphism. In method overriding, a subclass overrides a method with the same signature as that of in its superclass.**

class Animal {

   public void move() {

      System.out.println("Animals can move");

   }

}

class Dog extends Animal {

**//Function overriding (run time polymorphism)**

   public void move() {

      System.out.println("Dogs can walk and run");

   }

}

public class TestDog {

   public static void main(String args[]) {

      Animal a = new Animal();

      Animal b = new Dog();

      a.move();

      b.move();

   }

}

Output:

Animals can move

Dogs can walk and run

Practical 15: WAP to implement abstract classes.

Code:

**//Java abstract class is a class that can not be initiated by itself, it needs to be subclassed by another class to use its properties. An abstract class is declared using the “abstract” keyword in its class definition.**

**// Constructors are allowed.**

**// We can have an abstract class without any abstract method.**

**// There can be a final method in abstract class but any abstract method in class(abstract class) can not be declared as final  or in simpler terms final method can not be abstract itself as it will yield an error: “Illegal combination of modifiers: abstract and final”**

**// We can define static methods in an abstract class**

**// We can use the abstract keyword for declaring top-level classes (Outer class) as well as inner classes as abstract**

**// If a class contains at least one abstract method then compulsory should declare a class as abstract**

**// If the Child class is unable to provide implementation to all abstract methods of the Parent class then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract method**

**// Abstract class**

abstract class Sunstar {

    abstract void printInfo();

}

**// Abstraction performed using extends**

class Employee extends Sunstar {

    void printInfo()

    {

        String name = "avinash";

        int age = 21;

        float salary = 222.2F;

        System.out.println(name);

        System.out.println(age);

        System.out.println(salary);

    }

}

class Q15 {

    public static void main(String args[])

    {

        Sunstar s = new Employee();

        s.printInfo();

    }

}

Output:

avinash

21

222.2

Practical 16: WAP to demonstrate the concept of interface when two interfaces have

unique methods and same data members.

Practical 21: WAP to demonstrate creation of multiple child threads.

CODE:

// Multithreading using Runnable

public class Prac21 {

    public static void main(String[] args) {

        for (int i = 0; i < 5; i++) {

            Thread thread = new Thread(new ChildThread());

            thread.start();

        }

    }

}

class ChildThread implements Runnable {

    @Override

    public void run() {

        // Code to be executed by each child thread

        System.out.println("Thread " + Thread.currentThread().threadId() + " is running");

    }

}

OUTPUT:

Thread 32 is running

Thread 33 is running

Thread 35 is running

Thread 36 is running

Thread 34 is running

Practical 22: WAP that has two threads where one thread prints table of 5 and other thread prints a string 10 times. Set and display the names and priorities of these threads.

CODE:

// create threads

public class Prac22 {

    public static void main(String[] args) {

            Thread table = new Thread(new TableOf5());

            Thread string = new Thread(new String10());

            try {

                table.start();

                table.join();

                string.start();

                string.join();

            } catch (Exception e) {

                System.out.println(e);

            }

    }

}

class String10 implements Runnable {

    public void run() {

        Thread t = Thread.currentThread();

        t.setName("stringThread");

        t.setPriority(Thread.MIN\_PRIORITY);

        for (int i = 1; i <= 10; i++) {

            System.out.println("String " + i);

        }

        System.out.println(t.getName() + " has priority " + t.getPriority());

    }

}

class TableOf5 implements Runnable {

    public void run() {

        Thread t = Thread.currentThread();

        t.setName("tableThread");

        t.setPriority(Thread.MAX\_PRIORITY);

        for (int i = 1; i <= 10; i++) {

            System.out.println(5 + " x " + i + " = " + 5 \* i);

        }

        System.out.println(t.getName() + " has priority " + t.getPriority());

    }

}

OUTPUT:

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

tableThread has priority 10

String 1

String 2

String 3

String 4

String 5

String 6

String 7

String 8

String 9

String 10

stringThread has priority 1

Practical 23: WAP to create random access file and read & write integer data in it.

CODE:

// Import RandomAccess

import java.io.RandomAccessFile;

public class Prac23 {

    public static void main(String[] args) {

        try {

            RandomAccessFile raf = new RandomAccessFile("prac23.txt", "rw");

            raf.writeInt(10);

            raf.writeInt(20);

            raf.writeInt(30);

            raf.writeInt(40);

            raf.writeInt(50);

            raf.seek(0);

            System.out.println(raf.readInt());

            System.out.println(raf.readInt());

            System.out.println(raf.readInt());

            System.out.println(raf.readInt());

            System.out.println(raf.readInt());

            raf.close();

        } catch (Exception e) {

            System.out.println(e);

        }

    }

}

OUTPUT:

10

20

30

40

50

Practical 24: WAP to create random access file and read & write integer data in it.

**Application Based Practical**

1. Create a class employee which have name, age and address of employee, include methods getdata() and showdata(), getdata() takes the input from the user, showdata() display the data in following format:

Name:

Age:

Address:

**Code:**

import java.util.Scanner;

public class Employee {

private String name;

private int age;

private String address;

public void getData() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter name: ");

name = scanner.nextLine();

System.out.print("Enter age: ");

age = scanner.nextInt();

scanner.nextLine(); // Consume newline character left by nextInt()

System.out.print("Enter address: ");

address = scanner.nextLine();

}

public void showData() {

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

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

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

}

public static void main(String[] args) {

Employee emp = new Employee();

emp.getData();

System.out.println("\nEmployee Details:");

emp.showData();

}

}

**Output:**

Enter name: Chhota Bheem

Enter age: 29

Enter address: Dholakpur

Employee Details:

Name: Chhota Bheem

Age: 29

Address: Dholakpur