**Day-4 Assignment**

1. Create Multilevel inheritance for

// Vehicle

// Four\_Wheeler

// Petrol\_Four\_Wheeler

//FiveSeater\_Petrol\_Four\_Wheeler

// Baleno\_FiveSeater\_Petrol\_Four\_Wheeler

**Program:**

**class** Vehicle

{

**void** show()

{

System.***out***.println("Vehicle");

}

}

**class** Four\_wheeler **extends** Vehicle

{

**void** show()

{

System.***out***.println("Four Wheeler");

}

}

**class** Petrol\_Four\_Wheeler **extends** Four\_wheeler

{

**void** show()

{

System.***out***.println("Petrol Four Wheeler");

}

}

**class** FiveSeater\_Petrol\_Four\_Wheeler **extends** Petrol\_Four\_Wheeler

{

**void** show()

{

System.***out***.println("Five Seater Petrol Four Wheeler");

}

}

**class** Baleno\_FiveSeater\_Petrol\_Four\_Wheeler **extends** FiveSeater\_Petrol\_Four\_Wheeler

{

**void** show()

{

System.***out***.println("Baleno Five Seater Petrol Four Wheeler");

}

}

**public** **class** Multilevel\_Inheritance\_Demo

{

**public** **static** **void** main(String[] args)

{

Baleno\_FiveSeater\_Petrol\_Four\_Wheeler b = **new** Baleno\_FiveSeater\_Petrol\_Four\_Wheeler();

b.show();

}

}

**Output:**

Baleno Five Seater Petrol Four Wheeler

1. Demonstrate the use of the super keyword

**Program:**

**class** Parent

{

String name = "Parent Class";

Parent()

{

System.***out***.println("Parent Constructor");

}

**void** display()

{

System.***out***.println("Parent Display");

}

}

**class** Child **extends** Parent

{

String name = "Child Class";

Child()

{

**super**();

System.***out***.println("Child Constructor");

}

**void** display()

{

**super**.display();

System.***out***.println("Child Display");

System.***out***.println("Parent name: " + **super**.name);

}

}

**public** **class** SuperKeywordDemo

{

**public** **static** **void** main(String[] args)

{

Child c = **new** Child();

c.display();

}

}

**Output:**

Parent Constructor

Child Constructor

Parent Display

Child Display

Parent name: Parent Class

1. Create Hospital super class and access this class inside the patient child class and access properties from Hospital class

**Program:**

**class** Hospital

{

String hospitalName = "City Hospital";

**void** hospitalInfo()

{

System.***out***.println("Hospital Name: " + hospitalName);

}

}

**class** Patient **extends** Hospital

{

String patientName = "John";

**void** patientInfo()

{

hospitalInfo();

System.***out***.println("Patient Name: " + patientName);

}

}

**public** **class** HospitalPatientDemo

{

**public** **static** **void** main(String[] args)

{

Patient p = **new** Patient();

p.patientInfo();

}

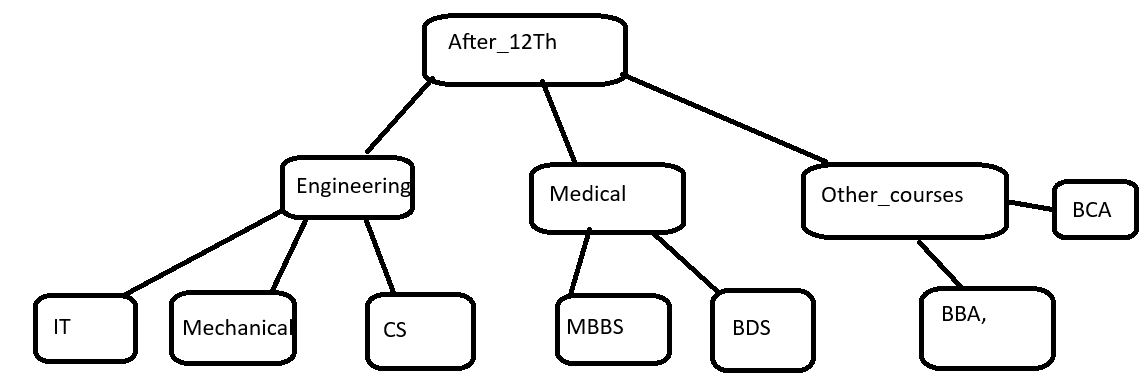
}

**Output:**

Hospital Name: City Hospital

Patient Name: John

1. Create Hierarchical inheritance



**Program:**

**class** Animal

{

**void** sound()

{

System.***out***.println("Animal Sound");

}

}

**class** Dog **extends** Animal

{

**void** sound()

{

System.***out***.println("Dog Barks");

}

}

**class** Cat **extends** Animal

{

**void** sound()

{

System.***out***.println("Cat Meows");

}

}

**public** **class** HierarchicalInheritanceDemo

{

**public** **static** **void** main(String[] args)

{

Dog d = **new** Dog();

Cat c = **new** Cat();

d.sound();

c.sound();

}

}

**Output:**

Dog Barks

Cat Meows

**Polymorphism:**

1. Create a class Calculator with the following overloaded add()

1. add(int a, int b)

2. add(int a, int b, int c)

3. add(double a, double b)

**Program:**

**class** Calculator

{

**int** add(**int** a, **int** b)

{

**return** a + b;

}

**int** add(**int** a, **int** b, **int** c)

{

**return** a + b + c;

}

**double** add(**double** a, **double** b)

{

**return** a + b;

}

}

**public** **class** CalculatorDemo

{

**public** **static** **void** main(String[] args)

{

Calculator calc = **new** Calculator();

System.***out***.println("Sum of 2 ints: " + calc.add(5, 3));

System.***out***.println("Sum of 3 ints: " + calc.add(5, 3, 2));

System.***out***.println("Sum of doubles: " + calc.add(5.5, 3.2));

}

}

**Output:**

Sum of 2 ints: 8

Sum of 3 ints: 10

Sum of doubles: 8.7

1. Create a base class Shape with a method area() that prints a message

Then create two sub-classes

Circle🡪override area() to calculator and print area of circle Rectangle🡪 override area() to calculate and print area of a rectangle

**Program:**

**class** Shape

{

**void** area()

{

System.***out***.println("Calculating area of a shape");

}

}

**class** Circle **extends** Shape

{

**double** radius;

Circle(**double** radius)

{

**this**.radius = radius;

}

@Override

**void** area()

{

**double** result = Math.***PI*** \* radius \* radius;

System.***out***.println("Area of Circle: " + result);

}

}

**class** Rectangle **extends** Shape

{

**double** length, width;

Rectangle(**double** length, **double** width)

{

**this**.length = length;

**this**.width = width;

}

@Override

**void** area()

{

**double** result = length \* width;

System.***out***.println("Area of Rectangle: " + result);

}

}

**public** **class** ShapeDemo

{

**public** **static** **void** main(String[] args)

{

Shape s1 = **new** Circle(5);

Shape s2 = **new** Rectangle(4, 6);

s1.area();

s2.area();

}

}

**Output:**

Area of Circle: 78.53981633974483

Area of Rectangle: 24.0

1. Create a Bank class with a method getInterest()

create sub-class:

SBI🡪return 6.7%

ICICI🡪return 7.0%

HDFC🡪return 7.5%

**Program:**

**class** Bank

{

**double** getInterestRate()

{

**return** 0.0;

}

}

**class** SBI **extends** Bank

{

@Override

**double** getInterestRate()

{

**return** 6.7;

}

}

**class** ICICI **extends** Bank

{

@Override

**double** getInterestRate()

{

**return** 7.0;

}

}

**class** HDFC **extends** Bank

{

@Override

**double** getInterestRate()

{

**return** 7.5;

}

}

**public** **class** BankDemo {

**public** **static** **void** main(String[] args)

{

Bank b1 = **new** SBI();

Bank b2 = **new** ICICI();

Bank b3 = **new** HDFC();

System.***out***.println("SBI Interest Rate: " + b1.getInterestRate() + "%");

System.***out***.println("ICICI Interest Rate: " + b2.getInterestRate() + "%");

System.***out***.println("HDFC Interest Rate: " + b3.getInterestRate() + "%");

}

}

**Output:**

SBI Interest Rate: 6.7%

ICICI Interest Rate: 7.0%

HDFC Interest Rate: 7.5%

1. Runtime Polymorphism with constructor Chaining

create a class vehicle with a constructor that prints “Vehicle Created”.

Create a subclass Bike that override a method and uses super() in constructor.

**Program:**

**class** Vehicle1

{

Vehicle1()

{

System.***out***.println("Vehicle Created");

}

**void** run()

{

System.***out***.println("Vehicle is running");

}

}

**class** Bike **extends** Vehicle1

{

Bike()

{

**super**();

System.***out***.println("Bike Created");

}

@Override

**void** run()

{

System.***out***.println("Bike is running");

}

}

**public** **class** VehicleDemo

{

**public** **static** **void** main(String[] args)

{

Vehicle1 v = **new** Bike();

v.run();

}

}

**Output:**

Vehicle Created

Bike Created

Bike is running

**Combined Questions:**

1. Create an abstract class SmartDevice with methods like turnOn(), turnOff() and performFunction().

Create child classes():

* SmartPhone: performs calling and browsing.
* SmartWatch: tracks fitness and time.
* SmartSpeaker: play music and responds to voice commands.

Write code to store all objects in array and use polymorphism to invoke their performFunction().

**Program:**

**abstract** **class** SmartDevice

{

**abstract** **void** turnOn();

**abstract** **void** turnOff();

**abstract** **void** performFunction();

}

**class** SmartPhone **extends** SmartDevice

{

**void** turnOn()

{

System.***out***.println("SmartPhone ON");

}

**void** turnOff()

{

System.***out***.println("SmartPhone OFF");

}

**void** performFunction()

{

System.***out***.println("Making calls and browsing internet");

}

}

**class** SmartWatch **extends** SmartDevice

{

**void** turnOn()

{

System.***out***.println("SmartWatch ON");

}

**void** turnOff()

{

System.***out***.println("SmartWatch OFF");

}

**void** performFunction()

{

System.***out***.println("Tracking fitness and showing time");

}

}

**class** SmartSpeaker **extends** SmartDevice

{

**void** turnOn()

{

System.***out***.println("SmartSpeaker ON");

}

**void** turnOff()

{

System.***out***.println("SmartSpeaker OFF");

}

**void** performFunction()

{

System.***out***.println("Playing music and responding to voice commands");

}

}

**public** **class** SmartDeviceDemo

{

**public** **static** **void** main(String[] args)

{

SmartDevice[] devices =

{

**new** SmartPhone(),

**new** SmartWatch(),

**new** SmartSpeaker()

};

**for** (SmartDevice device : devices)

{

device.turnOn();

device.performFunction();

device.turnOff();

System.***out***.println();

}

}

}

**Output:**

SmartPhone ON

Making calls and browsing internet

SmartPhone OFF

SmartWatch ON

Tracking fitness and showing time

SmartWatch OFF

SmartSpeaker ON

Playing music and responding to voice commands

SmartSpeaker OFF

1. Design an interface bank with methods deposit(), withdraw() and getBalance(). Implement thin in SavingsAccount and CurrentAcount classes.

Use inheritance to create a base Account class.

Demonstrate method overriding with customized logic for withdrawal (e.g., minimum balance in SavingsAccount).

**Program:**

**interface** Bank1

{

**void** deposit(**double** amount);

**void** withdraw(**double** amount);

**double** getBalance();

}

**class** Account

{

**protected** **double** balance;

Account(**double** balance)

{

**this**.balance = balance;

}

}

**class** SavingsAccount **extends** Account **implements** Bank1

{

**private** **static** **final** **double** ***MIN\_BALANCE*** = 500;

SavingsAccount(**double** balance)

{

**super**(balance);

}

**public** **void** deposit(**double** amount)

{

balance += amount;

System.***out***.println("Deposited in Savings: " + amount);

}

**public** **void** withdraw(**double** amount)

{

**if** (balance - amount >= ***MIN\_BALANCE***)

{

balance -= amount;

System.***out***.println("Withdrew from Savings: " + amount);

}

**else**

{

System.***out***.println("Cannot withdraw. Minimum balance required: " + ***MIN\_BALANCE***);

}

}

**public** **double** getBalance()

{

**return** balance;

}

}

**class** CurrentAccount **extends** Account **implements** Bank1

{

CurrentAccount(**double** balance)

{

**super**(balance);

}

**public** **void** deposit(**double** amount)

{

balance += amount;

System.***out***.println("Deposited in Current: " + amount);

}

**public** **void** withdraw(**double** amount)

{

balance -= amount;

System.***out***.println("Withdrew from Current: " + amount);

}

**public** **double** getBalance()

{

**return** balance;

}

}

**public** **class** BankInterfaceDemo

{

**public** **static** **void** main(String[] args)

{

Bank1 savings = **new** SavingsAccount(1000);

Bank1 current = **new** CurrentAccount(2000);

savings.deposit(500);

savings.withdraw(1200);

System.***out***.println("Savings Balance: " + savings.getBalance());

current.deposit(1000);

current.withdraw(2500);

System.***out***.println("Current Balance: " + current.getBalance());

}

}

1. Create a base class Vehicle with method start().

Drive Car, Bike and Truck from it and override the start() method.

* Create a static method that accepts Vehicle type and calls start().
* Pass different vehicle objects to test polymorphism.

**Program:**

**class** Vehicle0

{

**void** start()

{

System.***out***.println("Vehicle starting...");

}

}

**class** Car1 **extends** Vehicle0

{

**void** start()

{

System.***out***.println("Car starting with key");

}

}

**class** Bike1 **extends** Vehicle0

{

**void** start()

{

System.***out***.println("Bike starting with kick");

}

}

**class** Truck **extends** Vehicle0

{

**void** start()

{

System.***out***.println("Truck starting with heavy engine");

}

}

**public** **class** VehiclePolymorphismDemo

{

**static** **void** testStart(Vehicle0 v)

{

v.start();

}

**public** **static** **void** main(String[] args)

{

*testStart*(**new** Car1());

*testStart*(**new** Bike1());

*testStart*(**new** Truck());

}

}

1. Design an abstract class Person with fields like name, age and abstract method getRoleInfo().

Create subclasses:

* Student: has course and roll number.
* Professor: has subject and salary.
* TeachingAssistant: extends Student and implements getRoleInfo() in a hybrid way.
* Create and print info for all roles using overridden getRoleInfo().

**Program:**

**abstract** **class** Person

{

String name;

**int** age;

Person(String name, **int** age)

{

**this**.name = name;

**this**.age = age;

}

**abstract** **void** getRoleInfo();

}

**class** Student **extends** Person

{

String course;

**int** rollNumber;

Student(String name, **int** age, String course, **int** rollNumber)

{

**super**(name, age);

**this**.course = course;

**this**.rollNumber = rollNumber;

}

**void** getRoleInfo() {

System.***out***.println("Student: " + name + ", Age: " + age +

", Course: " + course + ", Roll: " + rollNumber);

}

}

**class** Professor **extends** Person

{

String subject;

**double** salary;

Professor(String name, **int** age, String subject, **double** salary)

{

**super**(name, age);

**this**.subject = subject;

**this**.salary = salary;

}

**void** getRoleInfo()

{

System.***out***.println("Professor: " + name + ", Age: " + age +

", Subject: " + subject + ", Salary: " + salary);

}

}

**class** TeachingAssistant **extends** Student

{

TeachingAssistant(String name, **int** age, String course, **int** rollNumber)

{

**super**(name, age, course, rollNumber);

}

**void** getRoleInfo()

{

System.***out***.println("Teaching Assistant: " + name + ", Age: " + age +

", Course: " + course + ", Roll: " + rollNumber + " (Assists Professors)");

}

}

**public** **class** PersonRoleDemo

{

**public** **static** **void** main(String[] args)

{

Person[] people = {

**new** Student("Alice", 20, "BSc", 101),

**new** Professor("Dr. Bob", 45, "Physics", 90000),

**new** TeachingAssistant("Charlie", 25, "MSc", 202)

};

**for** (Person p : people)

{

p.getRoleInfo();

}

}

}

1. Create:

* Interface Drawable with method draw().
* Abstract class Shape with abstract method area()

Subclasses: Circle, Rectangle and Triangle.

* Calculate area using appropriate formulas.
* Demonstrate how interface and abstract class work together.

**Program:**

**interface** Drawable

{

**void** draw();

}

**abstract** **class** Shape1

{

**abstract** **double** area();

}

**class** Circle1 **extends** Shape1 **implements** Drawable

{

**double** radius;

Circle1(**double** radius)

{

**this**.radius = radius;

}

**public** **void** draw()

{

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

}

**double** area()

{

**return** Math.***PI*** \* radius \* radius;

}

}

**class** Rectangle1 **extends** Shape1 **implements** Drawable

{

**double** length, width;

Rectangle1(**double** length, **double** width)

{

**this**.length = length;

**this**.width = width;

}

**public** **void** draw()

{

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

}

**double** area()

{

**return** length \* width;

}

}

**class** Triangle **extends** Shape1 **implements** Drawable

{

**double** base, height;

Triangle(**double** base, **double** height)

{

**this**.base = base;

**this**.height = height;

}

**public** **void** draw()

{

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

}

**double** area()

{

**return** 0.5 \* base \* height;

}

}

**public** **class** DrawableShapeDemo

{

**public** **static** **void** main(String[] args)

{

Drawable[] drawables = {

**new** Circle1(5),

**new** Rectangle1(4, 6),

**new** Triangle(3, 8)

};

**for** (Drawable d : drawables)

{

d.draw();

Shape1 s = (Shape1) d;

System.***out***.println("Area: " + s.area());

System.***out***.println();

}

}

}