WAP to show multilevel Inheritance :

```java
package com.test.multi_level;

class Employee
{
    protected String name;
    protected double salary;

    public Employee(String name, double salary)
    {
        super();
        this.name = name;
        this.salary = salary;
    }

    @Override
    public String toString()
    {
        return "Employee [name=" + name + ", salary=" + salary + "]";
    }
}

class Manager extends Employee
{
    protected String department;

    public Manager(String name, double salary, String department)
    {
        super(name, salary);
        this.department = department;
    }

    @Override
    public String toString()
    {
        return super.toString() + "Manager [department=" + department + "]";
    }
}

class TeamLead extends Manager
{
    protected int teamSize;
    protected String projectTitle;
    public TeamLead(String name, double salary, String department, int teamSize, String projectTitle)
    {
        super(name, salary, department);
        this.teamSize = teamSize;
        this.projectTitle = projectTitle;
    }

    @Override
    public String toString()
    {
        return super.toString() + "TeamLead [teamSize=" + teamSize + ", projectTitle=" + projectTitle + "]";
    }
}

public class MultiLevelDemo
{
    public static void main(String[] args)
    {
        TeamLead teamLead = new TeamLead("Alice", 75000, "IT", 8, "Online Cart");
        System.out.println(teamLead);
    }
}
```

WAP to show hierarchical Inheritance :

```java
package com.test.hierarchical_demo;

class Student
{
    protected int id;
    protected String name;
    protected double counter;

    public Student(int id, String name, double counter) {
        super();
        this.id = id;
        this.name = name;
        this.counter = counter;
    }
}

class HostelStudent extends Student
{
    protected double hostelFees;

    public HostelStudent(int id, String name, double counter, double hostelFees)
    {
        super(id, name, counter);
        this.hostelFees = hostelFees;
    }

    @Override
    public String toString()
    {
        return "HostelStudent [id=" + id + ", name=" + name + ", counter=" + counter + ", hostelFees="
                + hostelFees + "]";
    }
}

class DayScholar extends Student
{
    protected double transportFees;

    public DayScholar(int id, String name, double counter, double transportFees)
    {
        super(id, name, counter);
        this.transportFees = transportFees;
    }

    @Override
    public String toString() {
        return "DayScholar [id=" + id + ", name=" + name + ", counter=" + counter + ", 
transportFees=" + transportFees
                + "]";
    }
}

public class HierarchicalInheritance
{
    public static void main(String[] args)
    {
        HostelStudent hostelStudent = new HostelStudent(1, "Alex", 1600, 150000);
        System.out.println(hostelStudent);

        DayScholar dayScholar = new DayScholar(2, "Seth", 2500, 90000);
        System.out.println(dayScholar);
    }
}
```

Assignment :
- - - - - - - - - - - -
       Vehicle

    Car

  EV   **Petrol**

*** Why Java does not support multiple inheritance ?

Multiple Inheritance is a situation where a sub class wants to inherit the properties two or more than two super classes.

In every constructor we have super ( ) or this( ) to the first line. When compiler will add super( ) to the first line of the constructor then we have an ambiguity issue that super( ) will call which super class constructor as shown in the diagram (either A class OR B class).

It is also known as Diamond Problem in Java so the final conclusion is we can't achieve multiple inheritance using classes but same we can achieve by using interface (interface does not contain any constructor).

**** What is method overriding in Java ?

> Method Signature : Method name along with method parameter
>
> In method overriding : Method Signature must be same + return type must be compatible
>
> Method Overloading : Only possible with super & sub class with non static method
>
> Method Hiding : Only possible with super & sub class with static method

* For defining **super class non static method** with **sub class non static method in such a way that method signature (Method name along with parameter) must be same and return type must be same**.

Example :
```java
class Super
{
    public void m1()
    {
    }
}

class Sub extends Super
{
    public void m1() //Overridden method
    {
    }
}
```

* Method overriding is not possible without inheritance.

* While working with overriding, return type must be compatible so, generally we cannot change the return type of the method but from JDK 1.5V, We have a concept called **Co Variant** through which we can change the return type of the method.

Why we should override a method
- - - - - - - - - - - - - - - - - - - -
If we want to modify the super class method implementation, wants to provide our own implementation then we should override a method.

Advantage of Method Overriding :
- - - - - - - - - - - - - - - - - - - -
The advantage of Method Overriding is we can provide our own specific behaviour to a method.