Experiment No-6

Experiment Name: Demonstration of Inheritance (Multilevel & Hierarchical)

Task No-1: Grandfather Father Son (Multilevel Inheritance)

Task Name: Create a class Grandfather with a method company(). Create a class Father that extends Grandfather and adds a method car(). Create a class Son that extends Father and add a method. Demonstrate how Son can access all properties.

Objective:

- To understand and demonstrate multilevel inheritance in object-oriented programming (OOP), where a class inherits from another derived class, forming a chain (e.g., Grandfather → Father → Son). The goal is to verify that the Son class can access methods defined in both Father and Grandfather classes.

Problem analysis:

- In multilevel inheritance, a class (e.g., Father) inherits from a base class (Grandfather), and another class (Son) inherits from that derived class (Father).
- The deepest derived class (Son) should have access to all non-private members of its ancestors.
- We need to:
  - Define three classes in a hierarchy.
  - Each class adds at least one unique method.
  - Instantiate the Son class and invoke all inherited methods to verify accessibility.
- Assumptions:
  - All methods are public.
  - No method overriding is used (to keep it simple and demonstrate inheritance, not (polymorphism).

Algorithm:

1. Define class Grandfather with method company().
2. Define class Father that extends Grandfather, with method car().
3. Define class Son that extends Father, with method laptop().
4. In the main program:
   - Create an object of class Son.
   - Call company(), car(), and laptop() on this object.
5. Verify successful execution of all three methods.

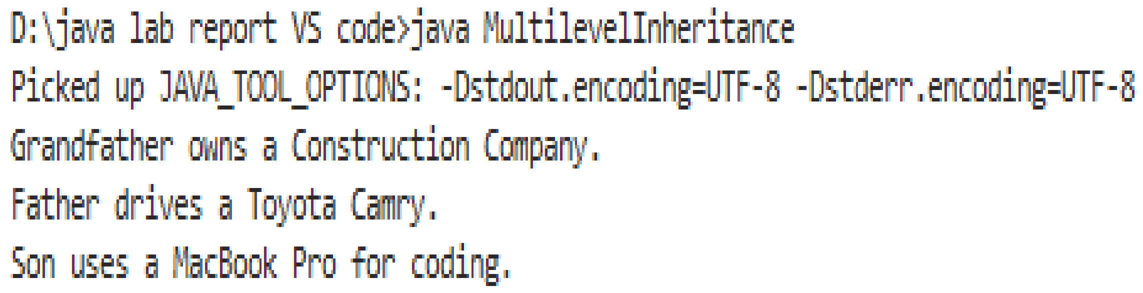
Source Code :

```
class Grandfather {
    public void company() {
        System.out.println("Grandfather owns a Construction Company.");
    }
}
class Father extends Grandfather {
    public void car() {
        System.out.println("Father drives a Toyota Camry.");
    }
}
```

```java
class Son extends Father {
    public void laptop() {
        System.out.println("Son uses a MacBook Pro for coding.");
    }
}

public class MultilevelInheritance {
    public static void main(String[] args) {
        Son son = new Son();
        son.company();
        son.car();
        son.laptop();
    }
}
```

Output:



```
D:\java lab report VS code>java MultilevelInheritance
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8
Grandfather owns a Construction Company.
Father drives a Toyota Camry.
Son uses a MacBook Pro for coding.
```

Discussion:

Multilevel inheritance allows a natural modeling of hierarchical real-world relationships (e.g., family lineage, organizational structure).The Son class implicitly inherits all public (and protected) methods from Father and Grandfather, promoting code reusability.Java (and most OOP languages) support multilevel inheritance, but not *multiple inheritance* of classes (to avoid the "diamond problem").

Caution: Deep inheritance chains can reduce code maintainability and increase coupling — use judiciously. Prefer *composition over inheritance* where appropriate.

In this task, no method overriding or data members were used for simplicity, but in real applications, subclasses often extend or specialize ancestor behavior.

Task No-2: Person Doctor/Teacher/Engineer (Hierarchical Inheritance) →

Task Name: Create a superclass Person with method displayInfo(). Create subclasses Doctor, Teacher, and Engineer extending Person. Each subclass will have its own method to describe their work.

Objective:

- To demonstrate hierarchical inheritance, where multiple subclasses (Doctor, Teacher, Engineer) inherit from a single superclass (Person). This illustrates code reuse, polymorphism, and how a common base class can support diverse specialized behaviors.

Problem analysis:

- Hierarchical inheritance: One base class → many derived classes (tree-like structure).
- Person holds shared attributes/behavior (displayInfo()).
- Each subclass adds domain-specific functionality (treat(), teach(), design()).
- Goal: Verify all subclasses independently inherit and extend Person, without interfering with each other.

Algorithm:

1.Define class Person with method displayInfo().

2.Define three subclasses:

- Doctor → adds treat()
- Teacher → adds teach()
- Engineer → adds design()

3.In main():

- Create one object of each subclass.
- Call displayInfo() (inherited) + their unique method.

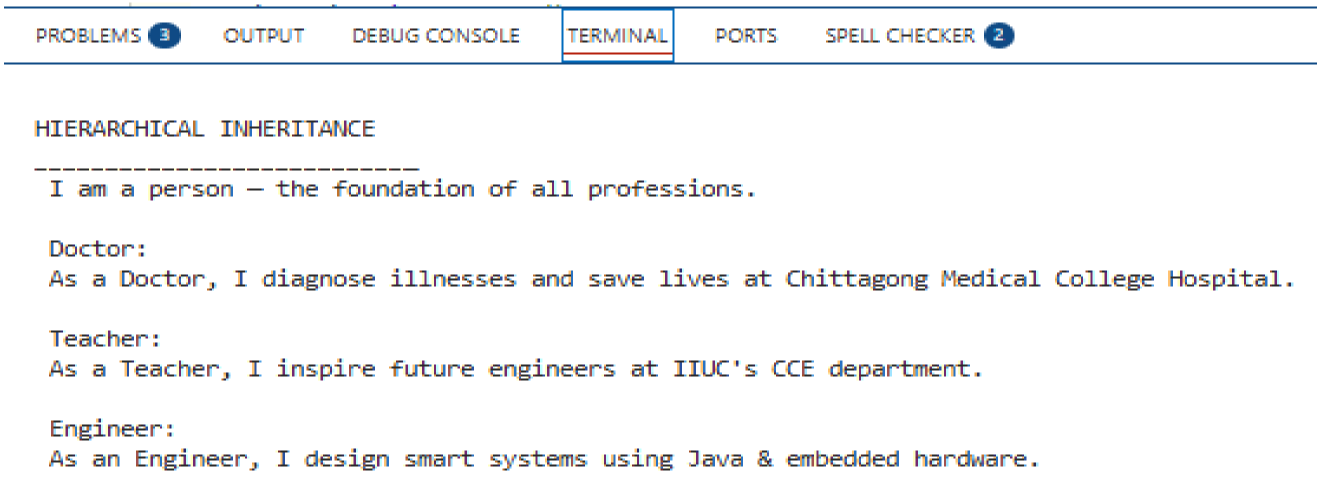
4.Observe independent, correct behavior for each.

Source code:

```
class Person {
    public void displayInfo() {
    System.out.println(" I am a person — the foundation of all professions.");
    }
}
class Doctor extends Person {
    public void treat() {
      System.out.println("As a Doctor, I diagnose illnesses and save lives at Chittagong
      Medical College Hospital.");
    }
}
class Teacher extends Person {
    public void teach() {
      System.out.println("As  a  Teacher,  I  inspire  future  engineers  at  IIUC's  CCE
      department.");
    }
}
class Engineer extends Person {
    public void design()
```

```java
{
      System.out.println("As an Engineer, I design smart systems using Java & embedded
      hardware.");
    }
}
public class Hierarchicalinheritance {
    public static void main(String[] args) {
        System.out.println("");
        System.out.println("HIERARCHICAL INHERITANCE");
        System.out.println("___________________________");
        Person general = new Person();
        general.displayInfo();
        System.out.println("\n Doctor:");
        Doctor dr = new Doctor();
        dr.treat();
        System.out.println("\n Teacher:");
        Teacher prof = new Teacher();
        prof.teach();
        System.out.println("\n Engineer:");
        Engineer eng = new Engineer();
        eng.design();
    }
}
```

Output:



```
HIERARCHICAL INHERITANCE
___________________________
 I am a person — the foundation of all professions.

 Doctor:
 As a Doctor, I diagnose illnesses and save lives at Chittagong Medical College Hospital.

 Teacher:
 As a Teacher, I inspire future engineers at IIUC's CCE department.

 Engineer:
 As an Engineer, I design smart systems using Java & embedded hardware.
```

Discussion:

Hierarchical and multilevel inheritance both promote code reuse and logical modeling of real-world relationships. Multilevel inheritance shows *transitive extension* (Son → Father → Grandfather), while hierarchical inheritance demonstrates *specialization from a common base* (Doctor/Teacher/Engineer ← Person). However, deep or wide inheritance trees can lead to tight coupling and maintenance challenges. Hence, inheritance should be used when there's a true *"is-a"* relationship — otherwise, composition is preferred for flexibility.