Problem 1: Simple Inheritance Example

Problem:
Create two classes: `Person` (superclass) and `Employee` (subclass). The `Person` class should have attributes like `name` and `age`. The `Employee` class should inherit from `Person` and add a new attribute `salary`. Write a method to display employee details.

Solution:

```
java
Copy code
class Person {
    String name;
    int age;

    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method to display person details
    public void displayPersonInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}

class Employee extends Person {
    double salary;

    // Constructor
    public Employee(String name, int age, double salary) {
        super(name, age); // Calling the superclass constructor
        this.salary = salary;
    }

    // Method to display employee details
    public void displayEmployeeInfo() {
        displayPersonInfo();  // Reusing Person's method
        System.out.println("Salary: " + salary);
    }
}

public class Main {
    public static void main(String[] args) {
        Employee emp = new Employee("John", 30, 50000.0);
        emp.displayEmployeeInfo();
    }
}
```

Output:

```
makefile
```

```
Copy code
Name: John
Age: 30
Salary: 50000.0
```

Problem 2: Overriding a Method

Problem:
Create a `Vehicle` superclass with a method `start()`. Create a subclass `Car` that overrides the `start()` method to display a specific message. Demonstrate method overriding.

Solution:

```
java
Copy code
class Vehicle {
    public void start() {
        System.out.println("The vehicle is starting.");
    }
}

class Car extends Vehicle {
    @Override
    public void start() {
        System.out.println("The car is starting with a key.");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle vehicle = new Vehicle();
        vehicle.start();  // Calls Vehicle's start method

        Car car = new Car();
        car.start();  // Calls Car's start method (overridden)
    }
}
```

Output:

```
vbnet
Copy code
The vehicle is starting.
The car is starting with a key.
```

Problem 3: Multilevel Inheritance

Problem:
Create three classes: `Animal` (superclass), `Mammal` (subclass of `Animal`), and `Dog` (subclass of `Mammal`). Each class should have a method to print information specific to that class. Demonstrate multilevel inheritance.

Solution:

```
java
Copy code
class Animal {
    public void eat() {
        System.out.println("This animal eats food.");
    }
}

class Mammal extends Animal {
    public void breathe() {
        System.out.println("This mammal breathes air.");
    }
}

class Dog extends Mammal {
    public void bark() {
        System.out.println("The dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();    // Inherited from Animal
        dog.breathe(); // Inherited from Mammal
        dog.bark();   // Defined in Dog
    }
}
```

Output:

```
Copy code
This animal eats food.
This mammal breathes air.
The dog barks.
```

Problem 4: Using `super` Keyword

Problem:
Create a superclass `Shape` with a `draw()` method. Create a subclass `Circle` that overrides the `draw()` method but calls the `Shape`'s `draw()` method within it using the `super` keyword.

Solution:

```
java
Copy code
class Shape {
    public void draw() {
        System.out.println("Drawing a shape.");
    }
}
```

```
class Circle extends Shape {
    @Override
    public void draw() {
        super.draw();  // Call superclass method
        System.out.println("Drawing a circle.");
    }
}

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle();
        circle.draw();
    }
}
```

Output:

```
css
Copy code
Drawing a shape.
Drawing a circle.
```

Problem 5: Constructor Chaining with Inheritance

Problem:
Create a superclass `Building` with a constructor that initializes the number of floors. Create a subclass `House` that has an additional attribute `rooms`. Use constructor chaining to initialize both floors and rooms.

Solution:

```
java
Copy code
class Building {
    int floors;

    // Constructor
    public Building(int floors) {
        this.floors = floors;
    }
}

class House extends Building {
    int rooms;

    // Constructor
    public House(int floors, int rooms) {
        super(floors); // Call superclass constructor
        this.rooms = rooms;
    }

    public void displayHouseInfo() {
```

```
        System.out.println("House with " + floors + " floors and " + rooms +
" rooms.");
    }
}

public class Main {
    public static void main(String[] args) {
        House house = new House(2, 5);
        house.displayHouseInfo();
    }
}
```

Output:

```
csharp
Copy code
House with 2 floors and 5 rooms.
```