# Lab Sheet: Object-Oriented Programming in Java – Inheritance & Encapsulation

**Course:** Introduction to Programming Language II (Java)
**Topic:** Types of Inheritance, Encapsulation, super(), this(), Constructor Chaining
**Level:** Beginner to Intermediate
**Duration:** 2.5 hours

**Prepared by:** Ashraful Islam Paran, Dept. of CSE, SEU

## 1. Objectives

By the end of this lab, students will be able to:

- Implement Single, Multilevel, and Hierarchical inheritance.
- Understand why Java does not support Multiple Inheritance through classes.
- Use encapsulation with private fields and public methods.
- Use `super()` and `this()` for constructor chaining.
- Override methods and call parent methods using `super`.

## 2. What is Inheritance?

Inheritance allows a class (subclass) to inherit fields and methods from another class (superclass) using the `extends` keyword. **Benefits:**

- Code Reusability
- Method Overriding
- Runtime Polymorphism
- Natural hierarchical organization

## 3. Types of Inheritance Supported in Java

- Single Inheritance `(Class B extends Class A)`
- Multilevel Inheritance `(C extends B extends A)`
- Hierarchical Inheritance `(B, C, D extend A)`
- Multiple & Hybrid → Not supported with classes (only via interfaces)

## 4. Example 1: Single Inheritance

```java
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}
public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat(); // Inherited from Animal
        d.bark(); // Defined in Dog
    }
}
```

**Output:**

```
This animal eats food.
The dog barks.
```

### Task 4.1 – Single Inheritance Practice (10 min)

a. Create a class `Bird` that extends `Animal`.

b. Add a method `fly()` in `Bird`.

c. In `main()`, create a `Bird` object and call both `eat()` and `fly()`.

d. Try creating an `Animal` reference that holds a `Bird` object (introduction to polymorphism).

## 5. Example 2: Multilevel Inheritance

```java
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}
class Puppy extends Dog {
    void weep() {
        System.out.println("The puppy weeps.");
    }
}
public class Main {
    public static void main(String[] args) {
        Puppy p = new Puppy();
        p.eat(); // From Animal (Grandparent)
        p.bark(); // From Dog (Parent)
        p.weep(); // From Puppy
    }
}
```

**Output:**

```
This animal eats food.
The dog barks.
The puppy weeps.
```

### Task 5.1 – Multilevel Inheritance (12 min)

Create the chain: `Vehicle` → `Car` → `SportsCar`

- `Vehicle` has `start()` and `stop()`
- `Car` adds `accelerate()`
- `SportsCar` adds `nitroBoost()`

Demonstrate that a `SportsCar` object can call all four methods.

## 6. Example 3: Hierarchical Inheritance

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}
class Dog extends Animal {
    void bark() { System.out.println("The dog barks."); }
}
class Cat extends Animal {
    void meow() { System.out.println("The cat meows."); }
}
class Puppy extends Dog {
    void weep() { System.out.println("The puppy weeps."); }
}
public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        Cat c = new Cat();
        Puppy p = new Puppy();

        d.eat(); d.bark();
        c.eat(); c.meow();
        p.eat(); p.bark(); p.weep();
    }
}
```

**Output:**

```
This animal eats food.
The dog barks.
This animal eats food.
The cat meows.
This animal eats food.
The dog barks.
The puppy weeps.
```

### Task 6.1 – Hierarchical Inheritance (15 min)

a. Create a base class `Shape` with method `draw()`.

b. Create three subclasses: `Circle`, `Rectangle`, `Triangle` – each overriding `draw()` with its own message.

c. In `main()`, create one object of each subclass and call `draw()`.

## 7. Example 4.1: Encapsulation – Student ID Card

```java
class Student {
    // Data is hidden (private)
    private String name;
    private int id;

    // Constructor     only way to set data
    public Student(String name, int id) {
        this.name = name;
        this.id = id;
    }

    // Public methods     only way to read data
    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }

    // Nice display method
    public void showInfo() {
        System.out.println("Name: " + name + " | ID: " + id);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student("Rahim", 2219001);
        s.showInfo();

        // These lines will NOT compile error (good!)
        // s.name = "Karim";   // private     not allowed
        // System.out.println(s.id);

        // Correct way     use getter
        System.out.println("Name from getter: " + s.getName());
    }
}
```

**Output:**

```
Name: Rahim | ID: 2219001
Name from getter: Rahim
```

## 7. Example 4.2: Encapsulation

```java
class Animal {
    private String name;

    Animal() {
        this("Unknown Animal"); // this() chaining
    }

    Animal(String name) {
        this.name = name;
    }

    void eat() {
        System.out.println(name + " is eating.");
    }

    public String getName() { return name; }
}
class Dog extends Animal {
    Dog() {
        super(); // Calls Animal()     which calls Animal(String)
    }

    Dog(String name) {
        super(name); // Directly call parameterized constructor
    }

    void bark() {
        System.out.println(getName() + " is barking.");
    }
}
public class Main {
    public static void main(String[] args) {
        Dog d1 = new Dog();
        Dog d2 = new Dog("Buddy");
        d1.eat(); d1.bark();
        d2.eat(); d2.bark();
    }
}
```

**Output:**

```
Unknown Animal is eating.
Unknown Animal is barking.
Buddy is eating.
Buddy is barking.
```

### Task 7.1 – Encapsulation & Constructor Chaining (20 min)

a. Create a class `Student` with private fields: `name`, `id`, `cgpa`.

b. Provide three constructors:

- No-arg → sets default values using `this("Unknown", 000, 0.0)`
- One String parameter (name only)

- Full three-parameter constructor

c. Use `this()` for chaining inside the class.

d. Provide public `getter` methods only (no setters for practice).

e. Test all three constructors in `main()`.

## 8. Example 5: Method Overriding with super keyword

```
class Animal {
    void sound() {
        System.out.println("The animal makes a sound");
    }
}
class Dog extends Animal {
    @Override
    void sound() {
        super.sound(); // Call parent method
        System.out.println("The dog says: Bow Wow");
    }
}
public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.sound();
    }
}
```

**Output:**

```
The animal makes a sound
The dog says: Bow Wow
```

**Task 8.1 – Method Overriding Practice (12 min)** Create:

- `Employee` with method `work()` → "Employee works"
- `Manager` extends `Employee` and overrides `work()` to first call `super.work()` then print "Manager manages team"

Also add a `Developer` class that extends `Employee` and prints "Developer writes code".

## 9. Real-World Example: Employee → Manager (with Encapsulation)

```
class Employee {
    private String name;
    private double salary;
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    void display() {
        System.out.println("Name: " + name + ", Salary: $" + salary);
    }
}
class Manager extends Employee {
    private double bonus;
    Manager(String name, double salary, double bonus) {
        super(name, salary);
        this.bonus = bonus;
    }
    @Override
    void display() {
        System.out.println("Manager: " + super.getClass().getSimpleName() +
                           " | Total Pay: $" + (salary + bonus));
    }
}
public class Main {
    public static void main(String[] args) {
        Manager m = new Manager("Karim", 60000, 20000);
        m.display();
    }
}
```

## 10. Example 10: super to Access Parent Class Variable

```
class Parent {
    int x = 10;
}

class Child extends Parent {
    int x = 20;

    void display() {
        System.out.println("Child x    = " + x);        // 20
        System.out.println("Child this.x = " + this.x); // 20
        System.out.println("Parent x   = " + super.x);  // 10     KEY
    }
}

public class Main {
    public static void main(String[] args) {
        new Child().display();
    }
}
```

**Output:**

```
Child x    = 20
Child this.x = 20
Parent x   = 10
```

## 11. Example 11: super to Call Parent Class Method

```
class Parent {
    void greet() {
        System.out.println("Hello from Parent");
    }
}

class Child extends Parent {
    @Override
    void greet() {
        super.greet();                    // Calls Parent's greet()
        System.out.println("Hello from Child");
    }
}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.greet();
    }
}
```

**Output:**

```
Hello from Parent
Hello from Child
```

## 12. Example 12: super() – Calling Parent Constructor (No-arg)

```
class Parent {
    Parent() {
        System.out.println("Parent constructor called");
    }
}

class Child extends Parent {
    Child() {
        super();  // Explicit call (optional if no-arg exists)
        System.out.println("Child constructor called");
    }
}

public class Main {
    public static void main(String[] args) {
        new Child();
    }
}
```

**Output:**

```
Parent constructor called
Child constructor called
```

## 13. Example 13: super() with Parameters – Person → Student

```
class Person {
    String name;
    Person(String name) {
        this.name = name;
        System.out.println("Person constructor: " + name);
    }
}

class Student extends Person {
    int id;
    Student(String name, int id) {
        super(name);           // Must pass name to Parent
        this.id = id;
        System.out.println("Student constructor: ID = " + id);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student("Ashraful", 2219022);
    }
}
```

**Output:**

```
Person constructor: Ashraful
Student constructor: ID = 2219022
```

## Example 14: Constructor Chaining

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

    // Constructor 1: No-arg
    Person() {
        this("Unknown", 0);  // chaining inside same class
        System.out.println("Person() called");
    }

    // Constructor 2: One-arg
    Person(String name) {
        this(name, 0);       // again chaining
        System.out.println("Person(String) called");
    }

    // Constructor 3: Full (main one)
    Person(String name, int age) {
        this.name = name;
        this.age = age;
        System.out.println("Person(String, int) called");
    }
}

class Student extends Person {
    int roll;

    Student() {
        super();  // calls Person()     which chains down
        System.out.println("Student() called");
    }

    Student(String name, int age, int roll) {
        super(name, age);           // directly call full parent
            ↪ constructor
        this.roll = roll;
        System.out.println("Student(full) called");
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println("--- Creating Student() ---");
        Student s1 = new Student();

        System.out.println("\n--- Creating Student(full) ---");
        Student s2 = new Student("Alice", 20, 101);
    }
}
```

**Expected Output:**

```
--- Creating Student() ---
Person(String, int) called
Person(String) called
```

```
Person() called
Student() called

--- Creating Student(full) ---
Person(String, int) called
Student(full) called
```

## 15. Summary Table

| Concept | Description | Example |
|---|---|---|
| Single Inheritance | One class extends one class | Dog extends Animal |
| Multilevel Inheritance | Chain: C → B → A | Puppy → Dog → Animal |
| Hierarchical | Many classes extend one class | Dog, Cat extend Animal |
| Encapsulation | private fields + public methods | private String name |
| super.variable | Access parent field | super.x |
| super.method() | Call parent method | super.greet() |
| super() | Call parent constructor | super(name) |
| this() | Call another constructor | this("Unknown") |
| Method Overriding | Redefine inherited method | @Override void sound() |

## 16. Final Task

Design a small system with the following classes using proper inheritance and encapsulation:

- `Vehicle` (base)
- `Car` and `Bike` (derived)
- `ElectricCar` extends `Car`

Each class should have private fields, proper constructors, getters/setters, and at least one overridden method.

## 17. Practice Questions

1. Explain why Java does not allow multiple inheritance with classes.
2. What is the difference between `super` and `super()`?
3. Write a program showing hierarchical inheritance with `Vehicle → Car, Bike, Truck`.
4. Create a `Person → Student → GraduateStudent` multilevel chain with proper constructors and `super()`.
5. Implement a `BankAccount` class and `SavingsAccount` subclass with interest calculation using overriding.

6. What happens if you place any statement before `super()` or `this()` in a constructor?

**Keep Coding!**