SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23CSB101- OBJECT ORIENTED PROGRAMMING

I YEAR /II SEMESTER

Unit II – INHERITANCE, PACKAGES AND INTERFACES

Topic : INHERITANCE

INHERITANCE

- **Inheritance** is the process by which **new classes** called derived classes are **created from** existing classes called **base classes**.
- The derived classes have all the features of the base class and the programmer can choose to add new features specific to the newly created derived class.

INHERITANCE

Features or Advantages of Inheritance:

1. Code reusability:

- Inheritance helps the code to be reused in many situations.
- The base class is defined and once it is compiled, it need not be reworked.
- Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

INHERITANCE

Features or Advantages of Inheritance:

2. Method Overriding:

- It is achieved only through inheritance. Also means of achieving run-time polymorphism

3. Abstraction:

- This does not provide all details only shows the functionality to the user.

INHERITANCE

Terminologies:

Super class/ Parent class: class whose features are inherited

Subclass / child class: class that inherits from another class. This class can have its own methods and variables apart in addition to base class members.

INHERITANCE

For Class Inheritance (Gen. format)

```
class ParentClass {
    // Fields (variables)
    // Methods
}

class ChildClass extends ParentClass {
    // Additional fields and methods
}
```

INHERITANCE

For Interface Inheritance (Gen. format)

```
interface InterfaceName {
    // Abstract methods (no body)
}

class ClassName implements InterfaceName {
    // Implementation of interface methods
}
```

INHERITANCE

For Multiple Interface Inheritance (Gen. format)

```
interface Interface1 {
    // Methods
}
interface Interface2 {
    // Methods
}
class ClassName implements Interface1, Interface2 {
    // Implementation of methods from both interfaces
}
```

INHERITANCE

Inheritance and Access Specifiers

Private:

- Not inherited by subclasses. Only accessible within the same class.

Default (Package-Private, no modifier):

- Accessible only within the same package.
- Not accessible from a subclass in a different package.

Cont…

INHERITANCE

Inheritance in Java and Access Specifiers

Protected :

- Accessible within the same package.
- Accessible in subclasses of different packages.
- Protected members can be accessed only via inheritance.

Public:

- Accessible everywhere (same class, same package, different package).

INHERITANCE

Access Specifiers in Java

Modifier	Same Class	Same Package	Different Package (Subclass)	Different Package (Non-Subclass)
private	✅ Yes	❌ No	❌ No	❌ No
default *(no modifier)*	✅ Yes	✅ Yes	❌ No	❌ No
protected	✅ Yes	✅ Yes	✅ Yes	❌ No
public	✅ Yes	✅ Yes	✅ Yes	✅ Yes

Type of Inheritance

Five types of inheritance:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance (through Interfaces)
5. Hybrid Inheritance (Combination of Multiple Types - through Interfaces)

1. Single Inheritance

- One class inherits from another class (one parent, one child).

```
class Parent {
    void display() {
      System.out.println("This is the Parent class.");
    }
}

class Child extends Parent {
    void show() {
        System.out.println("This is the Child class.");
    }
}
```

```
public class Main {

    public static void main(String[] args) {

        Child obj = new Child();

        obj.display(); // Inherited method

        obj.show();    // Child's own method

    }

}
```

2. Multilevel Inheritance

- A class inherits from a derived class (grandparent → parent → child).

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

class Parent extends GrandParent {
    void display() {
        System.out.println("This is the Parent
class.");
    }
}
```

```
class Child extends Parent {
    void show() {
        System.out.println("This is the Child class.");
    }
}

public class Main {
    public static void main(String[] args) {
        Child obj = new Child();
        obj.greet();   // From GrandParent
        obj.display(); // From Parent
        obj.show();    // Child's own method
    }
}
```

3. Hierarchical Inheritance

- One parent class has multiple child classes.

```
class Parent {
    void display() {
     System.out.println("This is the Parent class.");
    }
}

class Child1 extends Parent {
    void show() {
        System.out.println("This is Child1 class.");
    }
}

class Child2 extends Parent {
    void reveal() {
        System.out.println("This is Child2 class.");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Child1 obj1 = new Child1();
        obj1.display(); // Inherited from Parent
        obj1.show();

        Child2 obj2 = new Child2();
        obj2.display(); // Inherited from Parent
        obj2.reveal();
    }
}
```

4. Multiple Inheritance (Using Interfaces)

Java **does not support multiple inheritance with classes (**to avoid ambiguity (diamond problem)), **but** it allows multiple inheritance **using interfaces.**

The **Diamond Problem** occurs in **multiple inheritance** when a class inherits from two classes that have a common ancestor, leading to **ambiguity** in method resolution.

4. Multiple Inheritance (Using Interfaces)

```
interface A {
    void methodA();
}

interface B {
    void methodB();
}

class Child implements A, B {
    public void methodA() {
    System.out.println("Method from Interface A");
    }
public void methodB() {
System.out.println("Method from Interface B");
    }
}

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

5. Hybrid Inheritance (Using Interfaces)

```
interface A {
    void methodA();
}
interface B extends A {
    void methodB();
}
interface C {
    void methodC();
}
class D implements B, C {
    public void methodA() {
System.out.println("Method from Interface A");
    }
public void methodB() {
    System.out.println("Method from Interface B");
    }
public void methodC() {
     System.out.println("Method from Interface C");
    }
}
public class Main {
    public static void main(String[] args) {
        D obj = new D();
        obj.methodA();
        obj.methodB();
        obj.methodC();
    } }
```

✅ D implements both B and C, and B extends A, forming a hybrid structure.

Summary Table

Type of Inheritance	Description	Example
Single	One child inherits from one parent.	`Child extends Parent`
Multilevel	Grandparent → Parent → Child	`Child extends Parent extends GrandParent`
Hierarchical	One parent, multiple children.	`Child1 extends Parent`, `Child2 extends Parent`
Multiple (Using Interfaces)	A class implements multiple interfaces.	`class Child implements A, B`
Hybrid (Using Interfaces)	Combination of two or more inheritance types.	`interface B extends A`, `class D implements B, C`


Thank
you