



MODULE-3

Inheritance, Packages and Interfaces

Inheritance

Inheritance Basics

Definition: Inheritance is the process by which one class acquires the properties (fields and methods) of another class.

- Superclass / Parent class: The class being inherited.
- Subclass / Child class: The class that inherits.

Syntax

```
class Parent {  
    // fields and methods  
}  
  
class Child extends Parent {  
    // additional members  
}
```

Using super

The super keyword refers to the immediate parent class object.

Uses of super:

- Access parent class variables
- Call parent class methods
- Call parent class constructor

```
class Animal {  
    void eat() {  
        System.out.println("Animal eats");  
    }  
}
```

```
class Dog extends Animal {  
    void bark() {  
        System.out.println("Dog barks");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        d.eat();  
        d.bark();  
    }  
}
```

Output

Animal eats
Dog barks

Advantages

- Code reuse
- Method extension
- Easy maintenance

Creating a Multilevel Hierarchy

A class inherits from another derived class.

```
class A {  
    void showA() {  
        System.out.println("Class A");  
    }  
}  
class B extends A {  
    void showB() {  
        System.out.println("Class B");  
    }  
}  
class C extends B {  
    void showC() {  
        System.out.println("Class C");  
    }  
}  
public class Test {  
    public static void main(String[] args) {  
        C obj = new C();  
        obj.showA();  
        obj.showB();  
        obj.showC();  
    }  
}
```

When Constructors Are Executed

When an object of a subclass is created:

- Parent constructor executes first.
- Child constructor executes next.

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

```
class B extends A {  
    B() {  
        System.out.println("B constructor");  
    }  
}
```

```
class C extends B {  
    C() {  
        System.out.println("C constructor");  
    }  
}
```

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

Method Overriding

A subclass provides its own implementation of a parent class method.

Rules:

- Same method name
- Same parameters
- IS-A relationship

```
class Animal {  
void sound() {  
System.out.println("Animal sound");  
}  
}
```

```
class Dog extends Animal {  
    @Override  
    void sound() {  
        System.out.println("Dog barks");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        d.sound();  
    }  
}
```

Dynamic Method Dispatch (Runtime Polymorphism)

A parent class reference can refer to a child object.

```
class A {  
    void show() {  
        System.out.println("Class A");  
    }  
}
```

```
class B extends A {  
    @Override  
    void show() {  
        System.out.println("Class B");  
    }  
}
```

```
public class Demo {  
    public static void main(String[] args) {  
        A obj = new B();  
        obj.show();  
    }  
}
```

Using Abstract Classes

An abstract class cannot be instantiated.

It may contain:

- Abstract methods
- Concrete methods

```
abstract class Shape {  
    abstract void draw();  
  
    void message() {  
        System.out.println("Drawing shape");  
    }  
}  
  
class Circle extends Shape {  
    @Override  
    void draw() {  
        System.out.println("Drawing Circle");  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        Circle c = new Circle();  
        c.message();  
        c.draw();  
    }  
}
```

Using final with Inheritance

final Variable

```
final int MAX = 100;
```

Cannot be reassigned.

final Method

```
class A {  
    final void display() {}  
}  
  
class B extends A {  
    // display() cannot be overridden  
}
```

final Class

```
final class A {}
```

```
class B extends A {} // Error
```

Local Variable Type Inference and Inheritance (var)

Java allows local variable type inference using var.

```
class Animal {}
```

```
class Dog extends Animal {}
```

```
public class Demo {  
    public static void main(String[] args) {  
        var d = new Dog();  
        d.toString();  
    }  
}
```

Rules:

- Only for local variables
- Type inferred at compile time
- Cannot be used for fields or method parameters

The Object Class

Object is the root class of Java.

Every class implicitly extends Object.

```
class Student {  
    int id = 10;  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student s = new Student();  
        System.out.println(s.toString());  
        System.out.println(s.getClass());  
    }  
}
```

Packages

Packages

A package groups related classes and interfaces.

Benefits:

- Prevent name conflicts
- Access protection
- Easy maintenance

```
package mypack;
```

```
public class Hello {  
    public void display() {  
        System.out.println("Hello Package");  
    }  
}
```

Interfaces

An interface contains method declarations that implementing classes must define.

```
interface Shape {  
    void draw();  
}
```

```
class Circle implements Shape {  
    public void draw() {  
        System.out.println("Circle");  
    }  
}
```

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

Default Interface Methods

Introduced in Java 8.

```
interface A {  
    default void show() {  
        System.out.println("Default Method");  
    }  
}  
  
class B implements A {}  
  
public class Demo {  
    public static void main(String[] args) {  
        B obj = new B();  
        obj.show();  
    }  
}
```

Static Methods in an Interface

Can be called using the interface name

```
interface MathUtil {  
    static int square(int x) {  
        return x * x;  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        System.out.println(MathUtil.square(5));  
    }  
}
```

Private Interface Methods

Introduced in Java 9.

Used internally by default methods.

```
interface A {  
  
    private void helper() {  
        System.out.println("Helper Method");  
    }  
  
    default void show() {  
        helper();  
    }  
}  
  
class B implements A {}  
  
public class Demo {  
    public static void main(String[] args) {  
        B obj = new B();  
        obj.show();  
    }  
}
```



A T M E
College of Engineering

GOLD

QS I-GAUGE
INDEPENDENT RATING



THANK YOU