# Unit 3: Inheritance and Interfaces

## 3.1 Inheritance Basics

### Definition

Inheritance is a mechanism in Java where **one class (child class) inherits the properties and methods of another class (parent class)**.
It helps in:

- **Code reusability**
- **Method sharing**
- **Better program structure**

Java supports **single inheritance** using classes.

### Syntax

```
class ChildClass extends ParentClass {
}
```

### Program

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

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

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

```
}
```

### Output

```
Animal eats food
Dog barks
```

## 3.2 Inheritance and Constructors

### Definition

In inheritance, **parent class constructor is executed first**, then child class constructor.
This ensures that **parent data members are initialized before child members**.

### Syntax

```
class Child extends Parent {
    Child() {
    }
}
```

### Program

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

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

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

### Output

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

## 3.3 super Keyword

### Definition

super keyword refers to the **immediate parent class object**.
It is used to:

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

### Syntax

```
super();
super.variable;
super.method();
```

### Program

```
class Parent {
    int x = 10;

    void show() {
        System.out.println("Parent show method");
    }
}

class Child extends Parent {
    int x = 20;

    void display() {
        System.out.println(super.x);
        super.show();
    }
}

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

```
        c.display();
    }
}
```

### Output

```
10
Parent show method
```

## 3.4 Method Overriding

### Definition

Method overriding occurs when **a child class provides a specific implementation of a parent class method**.
It supports:

- **Runtime polymorphism**
- **Dynamic method calling**

### Syntax

```
class Child extends Parent {
    void method() {
    }
}
```

### Program

```
class Parent {
    void show() {
        System.out.println("Parent version");
    }
}

class Child extends Parent {
    void show() {
        System.out.println("Child version");
    }
}

public class Main {
    public static void main(String[] args) {
```

```
        Child c = new Child();
        c.show();
    }
}
```

### Output

```
Child version
```

## 3.5 Polymorphism

### Definition

Polymorphism means **one object behaving in many forms**.
It allows:

- Parent reference to point to child object
- Same method name with different behavior

### Syntax

```
Parent ref = new Child();
```

### Program

```
class Shape {
    void draw() {
        System.out.println("Drawing shape");
    }
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing circle");
    }
}

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

```
}
```

### Output

```
Drawing circle
```

## 3.6 Dynamic Binding

### Definition

Dynamic binding means **method call is resolved at runtime**, not compile time.
It occurs only with:

- Inheritance
- Method overriding

### Syntax

```
Parent obj = new Child();
```

### Program

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

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

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

### Output

```
Class B
```

## 3.7 `final` Keyword

### Definition

`final` keyword is used to **restrict modification**.
It can be applied to:

- Variable → value cannot change
- Method → cannot be overridden
- Class → cannot be inherited

### Syntax

```
final int x = 10;
final void method() {}
final class ClassName {}
```

### Program

```
class Parent {
    final void show() {
        System.out.println("Final method");
    }
}

class Child extends Parent {
    // cannot override show()
}

public class Main {
    public static void main(String[] args) {
        Parent p = new Parent();
        p.show();
    }
}
```

### Output

```
Final method
```

## 3.8 Abstract Classes

### Definition

An abstract class is a class that **cannot be instantiated** and may contain **abstract and non-abstract methods**.
It is used to **provide a base structure** for child classes.

### Syntax

```
abstract class ClassName {
    abstract void method();
}
```

### Program

```
abstract class Animal {
    abstract void sound();

    void sleep() {
        System.out.println("Animal sleeps");
    }
}

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

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

### Output

```
Dog barks
Animal sleeps
```

## 3.9 Access Specifiers

### Definition

Access specifiers control **where variables and methods can be accessed**.
They help in **data security and encapsulation**.

| Specifier | Access |
|---|---|
| public | Everywhere |
| protected | Same package + subclass |
| default | Same package |
| private | Same class only |

### Program

```
class Test {
    public int a = 10;
    private int b = 20;

    void show() {
        System.out.println(a);
        System.out.println(b);
    }
}

public class Main {
    public static void main(String[] args) {
        Test t = new Test();
        t.show();
    }
}
```

### Output

```
10
20
```

# 3.10 Interfaces

## Definition

An interface is a **collection of abstract methods**.
It supports:

- **Multiple inheritance**
- **100% abstraction**
- Loose coupling

## Syntax

```
interface InterfaceName {
    void method();
}
```

## Program

```
interface Vehicle {
    void start();
}

class Car implements Vehicle {
    public void start() {
        System.out.println("Car starts");
    }
}

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

## Output

```
Car starts
```