



EXPERT OOP & INTERFACE

Advanced Programming

Dr. Mojtaba Vahidi Asl

Ramona Noroozi

Shahid Beheshti University | Fall 2025

TABLE OF CONTENTS

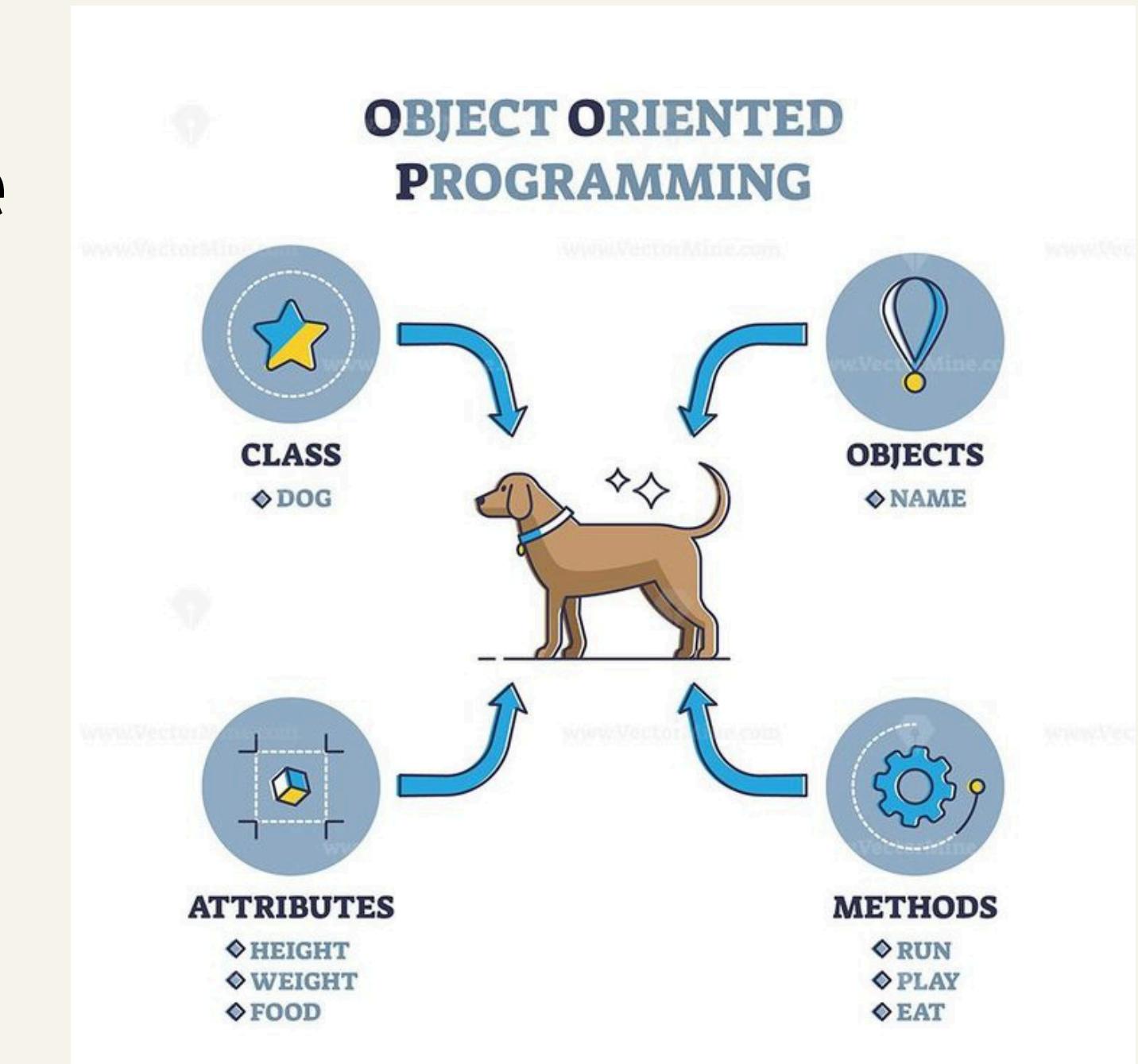
- What is OOP
- Abstraction in Java
- Abstract Classes
- Interfaces
- Inheritance
- Object Class
- Inner Class
- Encapsulation
- Polymorphism
- Binding
- Cloning
- Resources

WHAT IS OOP

- OOP stands for **Object-Oriented Programming**.
- **Object-oriented programming** is about creating objects that contain both data and methods.
- An **Object** represents an entity in real life.

WHAT IS OOP

- Class: Fruit → • Object: Apple
- Class: Vehicle → • Object: Truck
- Class: Car → • Object: Volvo
- Class: Animal → • Object: Dog



WHAT IS OOP

Example

```
public abstract class Animal {  
  
    protected abstract String produceSound();  
  
    protected static void identifyAnimal(String animalSound) {  
        // TODO  
    }  
}
```

ABSTRACTION IN JAVA

Data abstraction is the process of hiding certain details and showing only essential information to the user.

- Example: Driving a car or turning on the lights.

Abstraction in Java can be achieved with either **Abstract classes** or **Interfaces**.

Abstraction focuses on **what** an object does, **not how** it does it.

ABSTRACTION IN JAVA

Abstract Classes

```
abstract class Animal {  
    public abstract void animalSound();  
    public void sleep() {  
        System.out.println("Zzz");  
    }  
}
```

Interfaces

```
interface Animal {  
    public void animalSound();  
    public void sleep();  
}
```

ABSTRACT CLASSES

- Abstract Class is a **restricted** class that cannot be used to create objects (to access it, it must be inherited from another class).
- To access this class, it must be inherited from another class.
- An abstract class can have both **abstract and regular methods**.

ABSTRACT CLASSES

Example

```
abstract class Cat {  
    abstract void eat();  
    void speak() {  
        System.out.println("meow");  
    }  
}
```

INTERFACES

- Like abstract classes, interfaces cannot be used to create objects.
- Interface fields are by default **public, static and final**.
- Interface methods are by default **abstract and public**.
- On implementation of an interface, you must override all of its methods.
- A class can **implement multiple interfaces**.

INTERFACES

Example

```
interface Dog {  
    String word = "woof";  
    void eat();  
    void speak(String word);  
    void speak();  
}
```

INHERITANCE

In Java, it is possible to inherit attributes and methods from one class to another. We group "inheritance" into two categories:

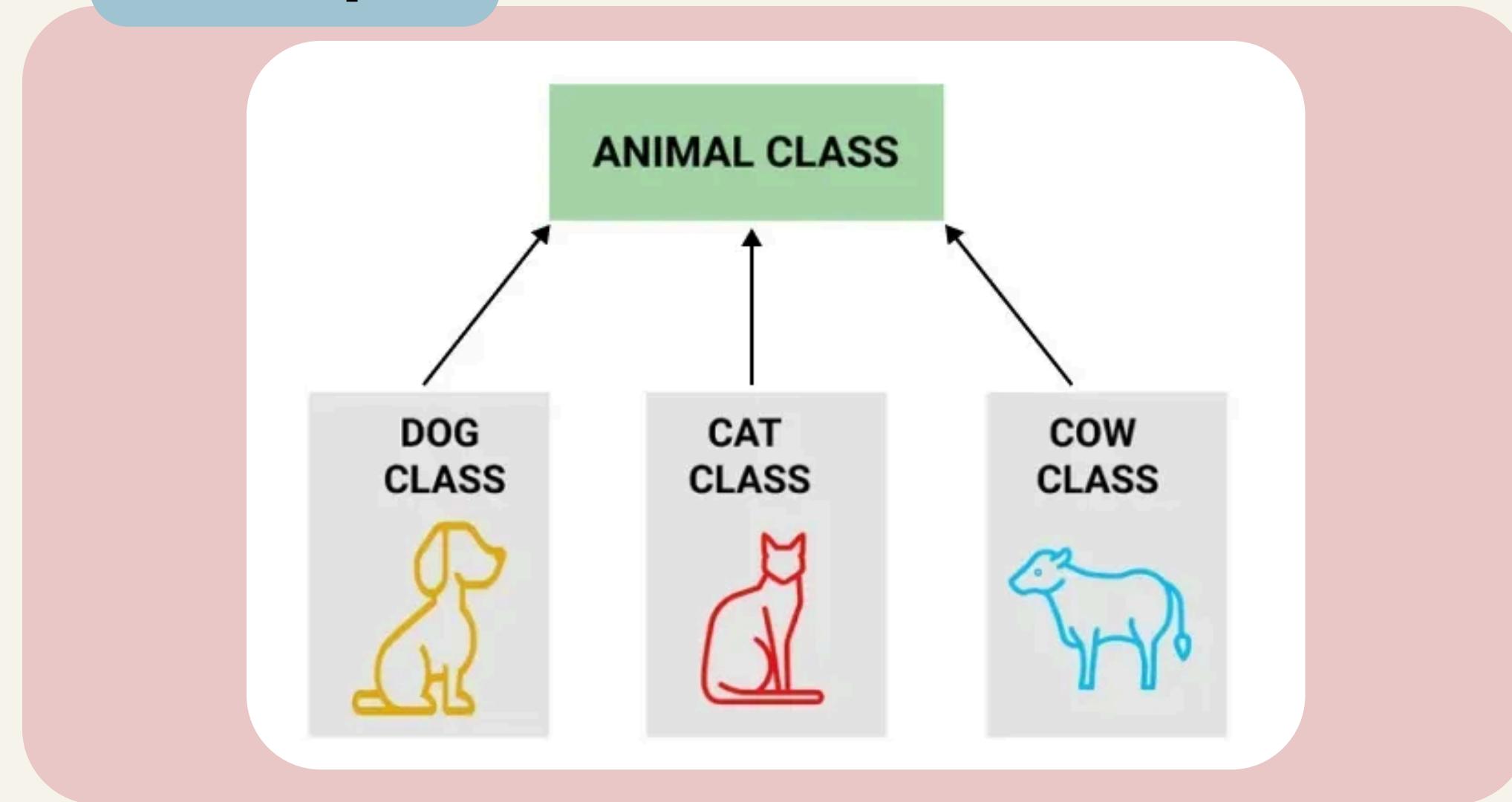
- **subclass (child)** - the class that inherits from another class
- **superclass (parent)** - the class being inherited from

To inherit from a class, use the **extends** keyword.

Java does not support multiple inheritance of classes.

INHERITANCE

Example



INHERITANCE

Example

```
class Vehicle {  
    protected String brand = "Ford";  
    public void honk() {  
        System.out.println("Tuut, tuut!");  
    }  
}  
  
class Car extends Vehicle {  
    private String modelName = "Mustang";  
    public static void main(String[] args) {  
        Car myFastCar = new Car();  
        myFastCar.honk();  
        System.out.println(myFastCar.brand + " " + myFastCar.modelName);  
    }  
}
```

OBJECT CLASS

1

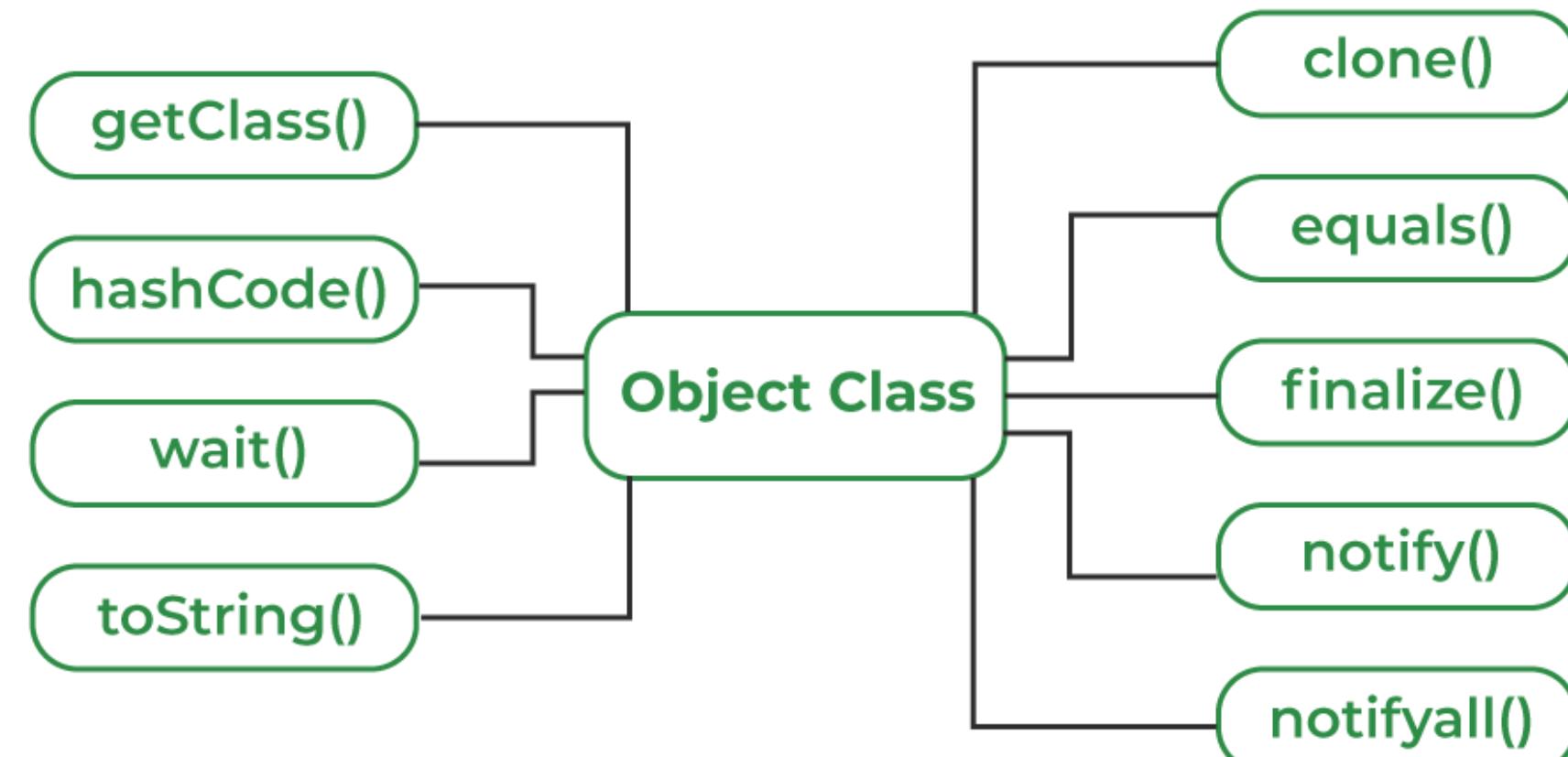
Every class directly or indirectly inherits from the Object class. If a class does not extend any other class then it is a direct child class of the Java Object class and if it extends another class then it is indirectly derived.

2

This class has some methods like hashCode, equals, toString and etc. equals() method compares the given object with the current object. toString() returns this: class name + @ + hash code of the argument object. (Default implementation – can be overridden)

OBJECT CLASS

Object Class Methods



INNER CLASS

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to **group classes that belong together**, which makes your code more readable and maintainable.

To access the inner class, create an object of the outer class, and then create an object of the inner class.

INNER CLASS

Example

```
class OuterClass {  
    int x = 10;  
  
    class InnerClass {  
        int y = 5;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        OuterClass myOuter = new OuterClass();  
        OuterClass.InnerClass myInner = myOuter.new InnerClass();  
        System.out.println(myInner.y + myOuter.x);  
    }  
}
```

ENCAPSULATION

Encapsulation means combining data and the functions that work on that data into a **single unit**, like a class. In Object-Oriented Programming, it helps keep things organized and secure. To achieve this, you must:

- declare class variables/attributes as **private**.
- provide public **get** and **set** methods to access and update the value of a **private variable**.

ENCAPSULATION

Example

```
public class Person {  
    private String name; // private = restricted access  
  
    // Getter  
    public String getName() {  
        return name;  
    }  
  
    // Setter  
    public void setName(String newName) {  
        this.name = newName;  
    }  
}
```

POLYMORPHISM



Polymorphism occurs when we have many classes that are related to each other by inheritance. Polymorphism uses inherited methods to perform different tasks. This allows us to **perform a single action in different ways.**

POLYMORPHISM

- **Multiple Behaviors:** The same method can behave differently depending on the object that calls this method.
- **Method Overriding:** A child class can redefine a method of its parent class.
- **Method Overloading:** We can define multiple methods with the same name but different parameters.
- **Runtime Decision:** At runtime, Java determines which method to call depending on the object's actual class.

POLYMORPHISM

Example

```
class Animal {  
    public void animalSound() {  
        System.out.println("The animal makes a sound");  
    }  
}  
  
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("The pig says: wee wee");  
    }  
}
```

BINDING

- Most generally, "binding" is about associating an identifier to whatever it identifies, be it a method, a variable, or a type.
- All bindings in Java are static ("early") except for bindings of instance methods, which may be **static** or **dynamic** ("late"), depending on method's accessibility.

BINDING

- Static binding: when the type is determined at **compile time**. If there is any private, static, final methods in class, there is static binding.
- Dynamic binding: when the type is determined at **run time**.

BINDING

Static binding

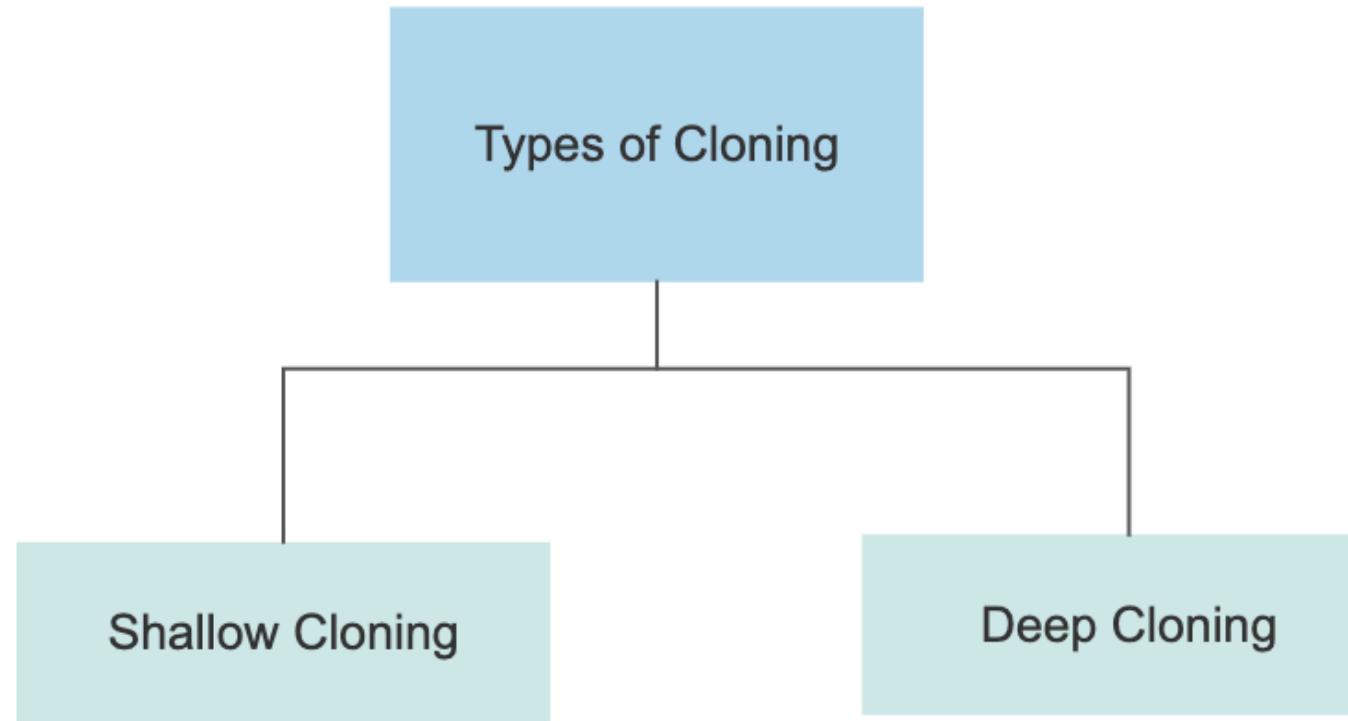
```
public class Main {  
    void addNumbers(int num1, int num2){  
        System.out.println(num1 + num2);  
    }  
  
    void addNumbers(int num1, int num2, int num3){  
        System.out.println(num1 + num2 + num3);  
    }  
  
    public static void main(String args[]){  
        //creating Main class object  
        Main object = new Main();  
        //method call  
        object.addNumbers(10, 20);  
        object.addNumbers(20, 30, 40);  
    }  
}
```

BINDING

Dynamic binding

```
class Test1{  
    void show(){  
        System.out.println("Inside show method of Test1 class");  
    }  
}  
  
public class Main extends Test1{  
    void show(){  
        System.out.println("Inside show method of Main class");  
    }  
  
    public static void main(String args[]){  
        //creating object  
        Test1 object = new Main();  
        //method call  
        object.show();  
    }  
}
```

CLONING



In Java, cloning is a process of creating an **exact copy** of an object. It is useful when you want a new object with the same state as an existing object, but one that is independent of the original object.

CLONING

- Shallow cloning: Shallow cloning creates a new object, but all non-primitive fields inside it still **reference the same objects** as the original. Primitive fields are copied by value, and reference fields are copied by reference.
- Deep cloning: Deep cloning creates a copy of an object along with all the objects it references, ensuring **complete independence** from the original object.

RESOURCES

- Dr. Mojtaba Vahidi Asl Slides
- <https://www.w3schools.com/java/default.asp>
- <https://www.geeksforgeeks.org/java/java/>
- <https://stackoverflow.com/questions/49759384/what-is-binding-in-java-terminology>

SBU | Fall 2025

THANK YOU

Presented By : Ramona Noroozi