25-Method Overriding

Method Overriding in Java

Definition: Method overriding in Java occurs when a subclass (or child class) provides a specific implementation of a method that is already defined in its superclass (or parent class). The method in the subclass must have the same name, parameters (or signature), and return type (or a subtype) as the method in the superclass. When this happens, the method in the subclass overrides the method in the superclass.

Key Points:

- **Run-Time Polymorphism:** Method overriding is a key feature that allows Java to achieve run-time polymorphism. The version of the method that gets executed is determined by the object that is used to invoke it, not by the type of reference variable.
- **Inheritance:** Method overriding is closely related to inheritance, as it allows subclasses to modify or extend the behavior of methods inherited from their parent classes.

Example of Method Overriding:

```
class A {
    // Method in the parent class
    public void show() {
        System.out.println("in A show");
    }
}


class B extends A {
    // Overridden method in the child class
    public void show() {
        System.out.println("in B show");
    }
}


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

```
        B obj = new B(); // Creating an object of subclass B
        obj.show(); // This will call the show() method in class B

        // Output:
        // in B show
    }
}
```

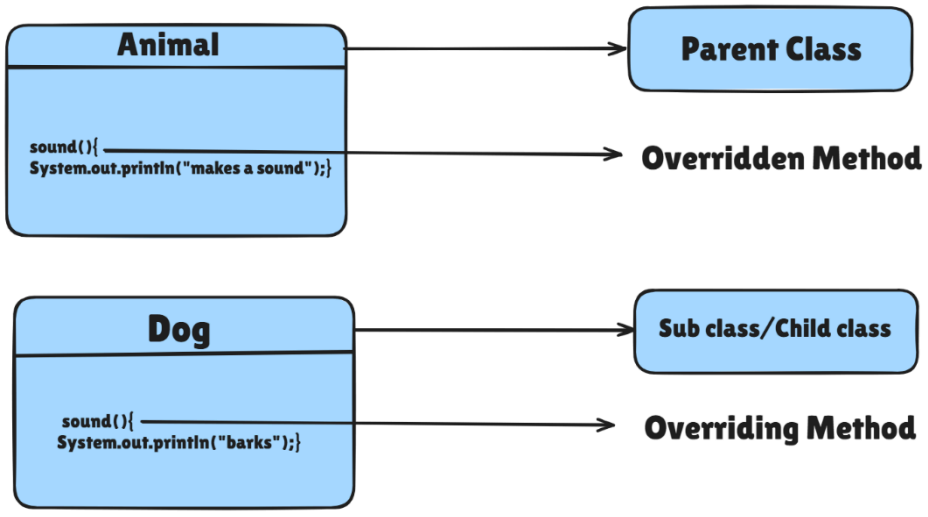
Explanation:

- In this example, class A defines a method show(). Class B, which extends class A, overrides the show() method by providing its implementation.
- When an object of class B is created and its show() method is called, the overridden method in class B is executed, displaying "in B show".
- If class B had not provided its implementation of show(), calling obj.show() would have executed the method from class A, displaying "in A show".

Importance of @Override Annotation:

- When overriding a method, it is a good practice to use the **@Override** annotation above the method. This explicitly tells the compiler that you intend to override a method from the superclass, and the compiler will generate an error if you accidentally do not match the method signature correctly.

```
@Override
public void show() {
    System.out.println("in B show");
}
```



How Method Overriding Works:

- **Dynamic Method Dispatch:** During runtime, the JVM determines which method to execute based on the actual object being referred to, not the reference type. This mechanism is called dynamic method dispatch.
- **Super Keyword:** If you need to call the overridden method from the superclass within the subclass, you can use the super keyword.

```
class B extends A {
    @Override
    public void show() {
        super.show(); // Calls the show() method from class A
        System.out.println("in B show");
    }
}
```