Polymorphism

B.Tech 4th Semester

.

Introduction of Polymorphism

.

Polymorphism means “many forms”. In Java, polymorphism allows methods or objects to behave differently based on the context.

There are three types of polymorphism in Java:

1. Method Overloading
2. Method Overriding
3. Operator Overloading (not supported in Java)

Method Overloading

.

Method Overloading occurs when multiple methods in the same class have the same name but different parameters (different type, number, or order of parameters).

Important Points:

- Used for improving code readability and reusability.
- Achieved by defining multiple methods with the same name but different signatures.
- Happens at **compile time** (also called compile-time polymorphism).

Method Overloading for different number of parameter

```
class MathOperations {
    int add(int a, int b) {
        return a + b;  }
    int add(int a, int b, int c) {
        return a + b + c;
    }
}
public class OverloadingExample {
    public static void main(String[] args) {
        MathOperations obj = new MathOperations();
        System.out.println("Sum (int): " + obj.add(5, 10));
        System.out.println("Sum (three int): " + obj.add(1, 2, 3));
    }
}
```

Method Overriding

Method Overriding occurs when a subclass provides a specific implementation of a method already defined in its parent class.

Important Points:

- The method in the subclass must have the same name, return type, and parameters as the method in the parent class.
- Used for achieving **runtime polymorphism**.
- The overridden method in the subclass must have the same access level or a more permissive access level.
- @Override **annotation** is used to indicate overriding.

Code

```
class Parent {
    void show() {
        System.out.println("This is the parent class method.");
    }
}
class Child extends Parent {
    @Override
    void show() {
        System.out.println("This is the child class method (Overridden).");
    }
}
public class OverridingExample {
    public static void main(String[] args) {
        Parent obj = new Child(); // Upcasting
        obj.show(); // Calls the overridden method in the child class
    }
}
```

Key Differences Between Method Overloading and Method Overriding

Feature	Method Overloading	Method Overriding
Definition	Multiple methods with the same name but different parameter lists in the same class	A subclass provides a new implementation of a method that exists in the parent class
Parameters	Must be different (type, number, or order)	Must be the same as the parent class method
Return Type	Can be the same or different	Must be the same or covariant type
Access Modifier	Can be the same or different	Cannot be more restrictive than the overridden method
Static Methods	Can be overloaded	Cannot be overridden (hides the method if redefined)
Runtime or Compile-time?	Compile-time polymorphism	Runtime polymorphism

Operator overloading

▪

Operator overloading allows operators to have different meanings based on their operands. Java does **not support operator overloading explicitly**, unlike C++.

Important Points About Operator Overloading in Java

1. Java **does not support** operator overloading except for + (String concatenation).

2. The + operator behaves differently for numbers (arithmetic addition) and strings (concatenation).

3. Java provides **method overloading** as an alternative to operator overloading.

4. Wrapper classes (e.g., Integer, Double) use the valueOf method to achieve object representation of primitives

+ Operator for Addition and String Concatenation

```
public class OperatorOverloadingExample {
    public static void main(String[] args) {
        // + as arithmetic addition
        int a = 10, b = 20;
        System.out.println("Sum: " + (a + b)); // Sum: 30

        // + as string concatenation
        String str1 = "Hello";
        String str2 = " World";
        System.out.println(str1 + str2); // Hello World

        // Mixed usage
        System.out.println("Result: " + a + b); // Result: 1020 (concatenation, not addition)
        System.out.println("Result: " + (a + b)); // Result: 30 (addition first)
    }
}
```

Using equals() Instead of == for Object Comparison

- Java does **not allow overloading** == **operator** for object comparison. Instead, it provides .equals() method.

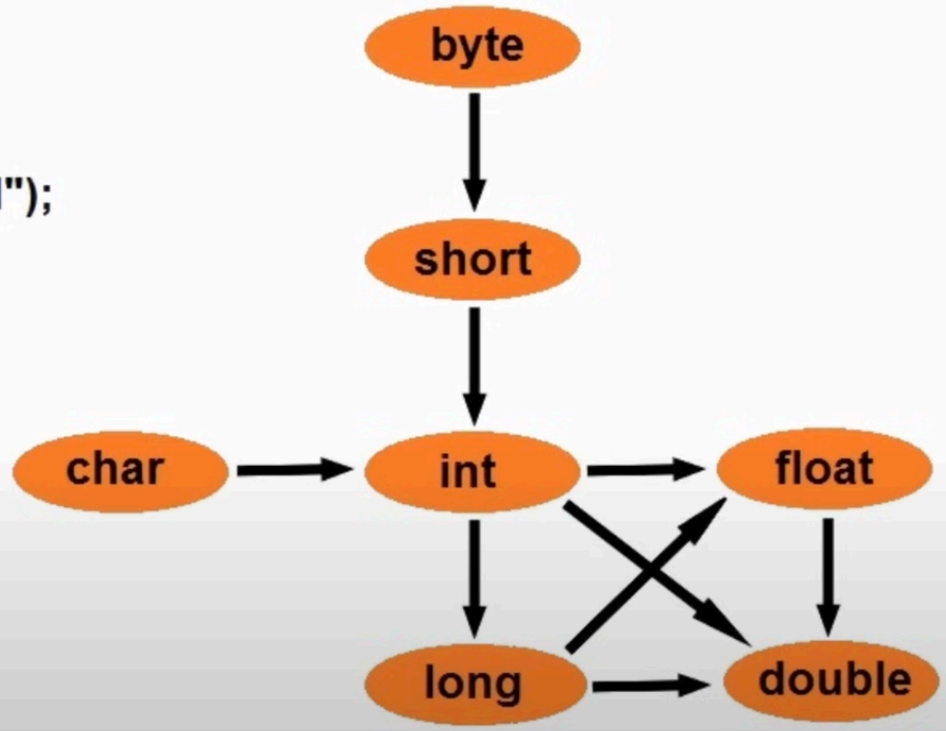
```
class Person {
    String name;
    Person(String name) {
        this.name = name;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true; // Checking reference
        if (obj == null || getClass() != obj.getClass()) return false;
        Person person = (Person) obj;
        return name.equals(person.name);
    }
}
```

Automatic Promotion

-
```
class Test
{
        void show(int a)
        {
                System.out.println("int method");
        }
        void show(String a)
        {
                System.out.println("string method");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                t.show('a');
        }
}
```

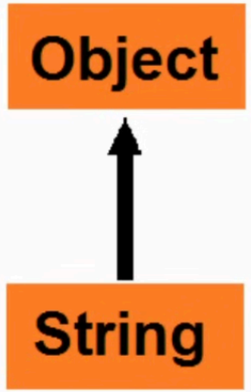
Automatic Promotion

One type is promoted to another implicitly if no matching datatype is found. Below is the diagram :



Case 2

```
class Test
{
        void show(Object a)
        {
                System.out.println("object method");
        }
        void show(String a)
        {
                System.out.println("string method");
        }
        public static void main(String[] args)
        {
                Test t=new Test();
                t.show("abc");
        }
}
```



Object is the parent class of all the classes in java

While resolving Overloaded Methods, Compiler will always give precedence for the child type argument than compared with parent type argument.

Thank You!