Constructors in Java

A **constructor** is a special method that is automatically invoked when an object is created. It is primarily used to **initialize** the newly created object.

```java
class Example {
    Example() {
        // Initialization logic
    }
}
```

> *Although class and method names can be the same in Java, it is not recommended as it may cause confusion.*

Key Properties

- The **constructor name must exactly match** the class name.
- Constructors **do not have a return type**, not even `void`.
- Constructors **cannot be inherited** or **overridden**.

Types of Constructors

1. No-Argument Constructor

A constructor that does not accept any parameters.

```java
class MessageService {
    MessageService() {
        System.out.println("No-argument constructor executed.");
    }
}
```

2. Parameterized Constructor

A constructor that takes parameters to initialize fields.

```java
class Rectangle {
    int length, breadth;

    Rectangle(int length, int breadth) {
        this.length = length;
        this.breadth = breadth;
    }
}
```

Constructor Behavior and Constraints

- The **first statement** in a constructor must be:
    - `super();` – to call the parent class constructor, or
    - `this();` – to call another constructor in the same class.
- If neither is explicitly written, the compiler automatically inserts `super();`.
- `super();` and `this();` are **mutually exclusive** and cannot coexist in the same constructor.

Comparison: `super()` / `this()` vs `super` / `this`

Feature	`super()` / `this()` (Constructor Calls)	`super` / `this` (Keywords)
Type	Constructor calls	Keywords
Purpose	Call parent (`super()`) or sibling (`this()`) constructor	Refer to parent (`super`) or current (`this`) members
Context	Only within a constructor	Usable anywhere except static contexts
Usage Frequency	Only once, must be first statement	Multiple uses allowed

Feature	`super()` / `this()` **(Constructor Calls)**	`super` / `this` **(Keywords)**
Restriction	Cannot combine `super()` and `this()`	Not usable in static contexts

---

Default Constructor

- A **default constructor** is automatically created by the compiler if no constructor is explicitly defined.
- It is always a **no-argument constructor** and includes only one statement: `super();`.
- If a user-defined constructor exists, the compiler **does not** create a default constructor.

```
class Library {
    // Compiler-generated default constructor (if none is defined)
    Library() {
        super(); // Only statement added by the compiler
    }
}
```

- Every class has **either** a user-defined constructor **or** a compiler-generated default constructor.
- *While all default constructors are no-argument, not all no-argument constructors are default constructors.*

Copy Constructor

A **copy constructor** initializes a new object using another object of the same class.

```
class Rectangle {
    int length, breadth;

    // Parameterized Constructor
    Rectangle(int length, int breadth) {
        this.length = length;
```

```java
        this.breadth = breadth;
    }

    // Copy Constructor
    Rectangle(Rectangle source) {
        this.length = source.length;
        this.breadth = source.breadth;
    }

    public static void main(String[] args) {
        Rectangle original = new Rectangle(10, 20);
        Rectangle copy = new Rectangle(original);
        System.out.println(copy.length + " " + copy.breadth);

        original.length = 11;
        copy.breadth = 13;
        System.out.println(copy.length + " " + copy.breadth);
    }
}
```