Constructor Reference in Java

1. Introduction

Constructor references were introduced in Java 8 as part of the functional programming features. They allow you to refer to a constructor using the ::new syntax instead of creating new objects with lambda expressions. Constructor references are a special type of method reference, used to create new instances of a class.

2. Syntax

General Syntax: ClassName::new This syntax refers to a constructor of the class and is used with functional interfaces that return an object.

3. Why Use Constructor References?

- To simplify code and make it more readable. - To avoid writing unnecessary lambda expressions. - To integrate smoothly with functional interfaces like Supplier, Function, and BiFunction. Example: Supplier supplier = () -> new Student(); Supplier supplier = Student::new;

4. Functional Interfaces Used with Constructor References

Supplier → Calls a no-argument constructor. Function → Calls a constructor with one parameter. BiFunction → Calls a constructor with two parameters.

5. Examples

5.1 Using No-Argument Constructor: Supplier supplier = Student::new; Student s = supplier.get(); 5.2 Using Parameterized Constructor (One Argument): Function func = Employee::new; func.apply("Mantu"); 5.3 Using Constructor with Two Parameters: BiFunction func = Product::new; func.apply(101, "Laptop"); 5.4 Constructor Reference with Streams: List students = names.stream().map(Student::new).collect(Collectors.toList());

6. Benefits

■ Cleaner and shorter syntax ■ Readable and expressive code ■ Works seamlessly with functional interfaces and streams ■ Promotes functional programming style in Java

7. Key Points

- Constructor references always end with ::new. - They can refer to any constructor that matches the target functional interface signature. - The compiler automatically chooses the matching constructor based on parameters.

8. Conclusion

Constructor references in Java simplify object creation in functional programming contexts. They remove boilerplate lambda expressions and improve code readability. You can use them effectively with Java Streams and functional interfaces like Supplier, Function, and BiFunction.