Constructor Reference

👉 Constructor Reference

A constructor reference is a special kind of method reference that refers to a class constructor. Just like method references, constructor references provide a shorthand way to write lambda expressions, but specifically for creating new objects. They use the same **::** (double colon) operator syntax followed by the keyword new.

👉 Key Points

- Constructor references use the syntax ClassName::new
- They are used to create new objects in functional programming contexts
- Constructor references work like factory methods for creating objects
- Java automatically selects the appropriate constructor based on the context
- They can be used with any constructor (default, parameterized, etc.)
- Constructor references make object creation code more concise in streams

👉 Constructor Reference Example

➢ First, let's create a Student class and a list of names that we'll work with:

```java
import java.util.Arrays;
import java.util.List;

class Student {
    private String name;
    private Integer age;

    // 0-params constructor
    public Student() {
    }

    // Single parameter constructor
    public Student(String name) {
        this.name = name;
    }

    // Getters and Setters
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    // toString method for printing Student objects
    @Override
    public String toString() {
        return "Student{name='" + name + "', age=" + age + "}";
    }
}

public class ConstructorRefEx {
    public static void main(String[] args) {
        // Create a list of names
        List<String> names = Arrays.asList("Navin", "Harsh", "John");

    }
}
```

➢ Now, let's create a Student object for each name using a traditional for loop:

```java
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;

// Student class as defined above

public class ConstructorReferenceDemo {
    public static void main(String[] args) {
        // Create a list of names
        List<String> names = Arrays.asList("Navin", "Harsh", "John");
        System.out.println("Names: " + names);

        // Create Student objects using a for loop
        List<Student> students = new ArrayList<>();

        for (String name : names) {
            students.add(new Student(name));
        }

        System.out.println("Students (using for loop): " + students);
    }
}
```

Output:

```
Names: [Navin, Harsh, John]
Students (using for loop): [Student{name='Navin', age=0}, Student{name='Harsh', age=0}, Student{name='John', age=0}]
```

- Let's improve our approach by using Stream API with a lambda expression:

```java
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;

// Student class as defined above

public class ConstructorReferenceDemo {
    public static void main(String[] args) {
        // Create a list of names
        List<String> names = Arrays.asList("Navin", "Harsh", "John");
        System.out.println("Names: " + names);

        // Create Student objects using Stream API with lambda
        List<Student> students = names.stream()
                                      .map(name -> new Student(name))
                                      .toList();

        System.out.println("Students (using stream with lambda): " + students);
    }
}
```

Output:

```
Names: [Navin, Harsh, John]
Students (using stream with lambda): [Student{name='Navin', age=0}, Student{name='Harsh', age=0},
Student{name='John', age=0}]
```

- Finally, we can simplify the code further by using a constructor reference:

```
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;

// Student class as defined above

public class ConstructorReferenceDemo {
    public static void main(String[] args) {
        // Create a list of names
        List<String> names = Arrays.asList("Navin", "Harsh", "John");
        System.out.println("Names: " + names);

        // Create Student objects using Stream API with constructor reference
        List<Student> students = names.stream()
                                      .map(Student::new)
                                      .toList();

        System.out.println("Students (using constructor reference): " + students);
    }
}
```

Output:

```
Names: [Navin, Harsh, John]
Students (using constructor reference): [Student{name='Navin', age=0}, Student{name='Harsh',
age=0}, Student{name='John', age=0}]
```

👉 Explanation:

- When you use a constructor reference like Student::new, here's what happens:

1. Java recognizes that you want to create new Student objects
2. It looks for a constructor in the Student class that matches the context
3. In our example, the Stream's map function expects a function that takes a String and returns a Student
4. Java automatically matches this with the Student(String name) constructor
5. Each name from the stream is passed to this constructor, creating a new Student object