# Imperative vs Declarative Programming: Removing Duplicates

## Introduction

In this tutorial, we will code another example to showcase the difference between the \*\***imperative**\*\* and \*\***declarative**\*\* **programming styles**.

### \*\*Use Case\*\*

- Remove duplicates from a list of integers and obtain a list with unique values.

## Project Setup

- The example is implemented using \*\*IntelliJ IDEA\*\*.

- A class named \*\*ImperativeVsDeclarativeExample2\*\* is created.

- The class is made executable by adding \*\*public static void main\*\* method.

## Step 1: Creating the Input List

- A \*\*List<Integer>\*\* named `integerList` is initialized using \*\*Arrays.asList()\*\*.

- The list contains duplicate values:

`[1, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10]`

## Step 2: Imperative Programming Approach

1. Create a new list, `uniqueList`, to store unique values.

2. Iterate through `integerList` using a \*\*for-each loop\*\*.

3. Check if `uniqueList` already contains the integer.

4. If not, add the integer to `uniqueList`.

5. Print the final `uniqueList`.

### \*\*Code Implementation:\*\*

import java.util.\*;  
  
public class ImperativeVsDeclarativeExample2 {  
 public static void main(String[] args) {  
 List<Integer> integerList = Arrays.asList(1, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10);  
   
 List<Integer> uniqueList = new ArrayList<>();  
 for (Integer integer : integerList) {  
 if (!uniqueList.contains(integer)) {  
 uniqueList.add(integer);  
 }  
 }  
  
 System.out.println("Unique List using Imperative approach: " + uniqueList);  
 }  
}

### \*\*Output:\*\*

- The unique list will be: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

## Step 3: Declarative Programming Approach

1. Use \*\*stream()\*\* introduced in Java 8.

2. Apply \*\*distinct()\*\* to filter unique values.

3. Collect the result into a new list using \*\*Collectors.toList()\*\*.

4. Print the final unique list.

### \*\*Code Implementation:\*\*

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class DeclarativeExample2 {  
 public static void main(String[] args) {  
 List<Integer> integerList = Arrays.asList(1, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10);  
   
 List<Integer> uniqueList = integerList.stream() **//This avoids the mutability as returning a new object**.  
 .distinct()  
 .collect(Collectors.toList());  
  
 System.out.println("Unique List using Declarative approach: " + uniqueList);  
 }  
}

### \*\*Output:\*\*

- The unique list will be: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

## Step 4: Comparison - Imperative vs Declarative

### \*\***Imperative Approach** (**How-style programming**) \*\*

- Manually iterates through the list.

- Uses a \*\***for-each loop**\*\* to check and add unique elements.

- Uses \*\***mutable list** (`uniqueList`)\*\*, modifying it during iteration.

- More verbose and requires checking each element manually.

### \*\***Declarative Approach (What-style programming)**\*\*

- Uses \*\*built-in Java 8 functions\*\* to handle duplication.

- Calls \*\*stream() -> distinct() -> collect()\*\*, avoiding explicit iteration.

- Avoids mutability, making it \*\*more readable and maintainable\*\*.

- The underlying system handles filtering, improving efficiency.

## Step 5: Benefits of Declarative Programming

- We \*\*call functions\*\* instead of defining the logic ourselves.

- Code is \*\*shorter, cleaner, and easier to understand\*\*.

- \*\***Performance optimized**\*\* as internal Java methods handle the operations.

- Helps in \*\*writing better, more maintainable\*\* code.

## Conclusion

- Java 8 introduced powerful built-in functions that reduce the need for manual operations.

- The \*\***imperative approach**\*\* is more manual and verbose.

- The \*\***declarative approach**\*\* makes use of \*\*Java 8 Streams API\*\* for cleaner and efficient code.

- The next tutorial will cover another \*\*real-world use case\*\* using Java 8 features.

### \*\*End of the Tutorial\*\*

Thank you for watching!