# Real-World Use Cases of Consumer Functional Interface

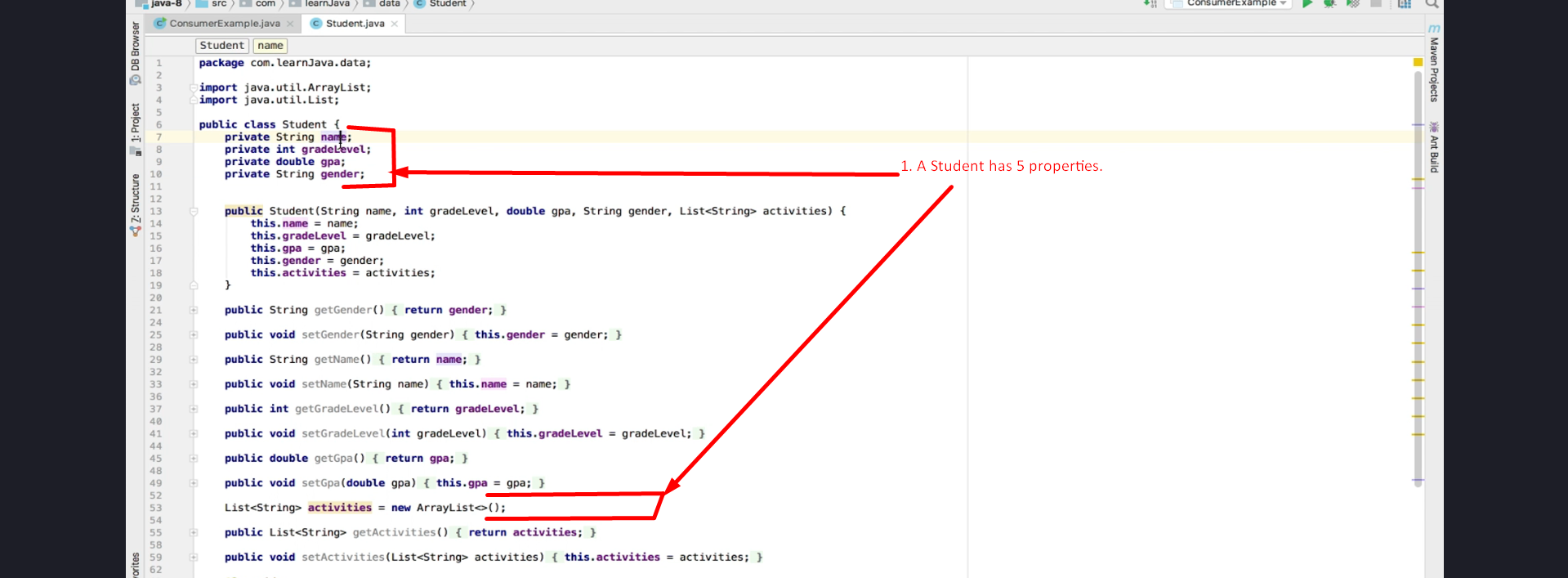
## Introduction

In this tutorial, we will implement \*\*real-time use cases\*\* using the \*\*Consumer Functional Interface\*\* in Java 8.

## Setting Up the Data

1. A new package named \*\*data\*\* is created.

2. The package contains two classes:

- \*\*Student\*\*: Contains four properties: `name`, `gradeLevel`, `GPA`, and `gender` with appropriate getter, setter, and `toString()` methods.  


- \*\*StudentDataBase\*\*: Contains a static method `getAllStudents()` that returns a list of six students (two from each grade level: 2nd, 3rd, and 4th).  


## Implementing Consumer Functional Interface

1. A new class \*\*ConsumerExample\*\* is created to demonstrate the use of the \*\*Consumer Functional Interface\*\*.

2. The method \*\*printName()\*\* is created to iterate over and print the names of students.

## Using forEach() Method

- Java 8 introduced \*\*forEach()\*\*, a \*\*default method\*\* in the `List` interface.

- It accepts a \*\*Consumer Functional Interface\*\* as an argument.

- Instead of using an enhanced `for` loop, we use `forEach()` to iterate through student data.

## Implementing Consumer Using Lambda

1. \*\*Consumer<Student> c2\*\* is created to print each student's details using `toString()`.

2. The `forEach()` method accepts `c2`, which prints all students from `StudentDataBase`.

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

public class ConsumerExample {  
 public static void main(String[] args) {  
 List<Student> studentList = StudentDataBase.getAllStudents();  
  
 Consumer<Student> c2 = (student) -> System.out.println(student);  
  
 studentList.forEach(c2);  
 }  
}

## Printing Student Name and Activities

1. \*\*Method printNameAndActivities()\*\* is created to print student names and activities.

2. Each student has a \*\*list of activities\*\* such as `swimming, basketball, volleyball`.

3. Two separate consumers are created:

- \*\*c3\*\*: Prints the student's name.

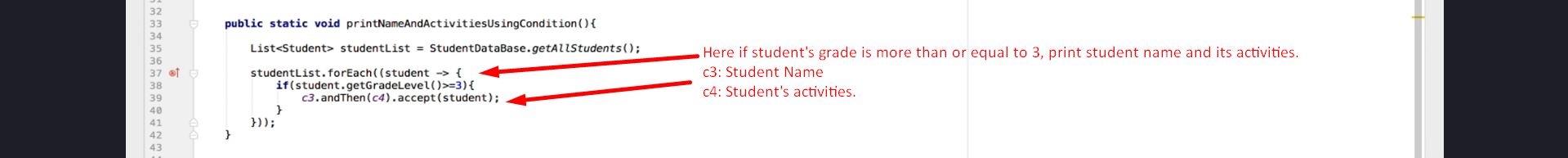
- \*\*c4\*\*: Prints the student's activities.

4. The \*\*andThen()\*\* method is used to \*\*chain\*\* consumers to print both name and activities in one execution.

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

Consumer<Student> c3 = (student) -> System.out.print(student.getName() + " - ");  
Consumer<Student> c4 = (student) -> System.out.println(student.getActivities());  
  
studentList.forEach(c3.andThen(c4)); // **Consumer Chaining** to print student’s name and activities.

## Filtering Students Based on Grade



1. A new method \*\*printNameAndActivitiesUsingCondition()\*\* is created.

2. Students with a \*\*grade level of 3.0 or higher\*\* are filtered.

3. Instead of duplicating code, a \*\*Lambda expression\*\* is used within `forEach()` to filter students before printing.

4. The reusable consumer \*\*c3\*\* (printing names) is used to avoid code duplication.

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

studentList.forEach(student -> {  
 if(student.getGradeLevel() >= 3) {  
 c3.accept(student);  
 c4.accept(student);  
 }  
});

## Filtering Based on GPA and Grade Level

1. A new condition is added to filter students with \*\*GPA >= 3.9\*\*.

2. The same consumer `c3` is reused to print the names of selected students.

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

studentList.forEach(student -> {  
 if(student.getGradeLevel() >= 3 && student.getGPA() >= 3.9) {  
 c3.accept(student);  
 }  
});

## Need for Predicate Functional Interface

1. If multiple conditions need to be applied dynamically, \*\*Predicate Functional Interface\*\* is a better alternative.

2. Instead of duplicating filtering conditions, \*\*Predicate\*\* can encapsulate the logic.

3. This will be covered in upcoming tutorials.

## Chaining Multiple Consumer Interfaces

1. We previously used \*\*andThen()\*\* to chain two consumers.

2. Java allows chaining \*\*N number of Consumer Functional Interfaces\*\* using `andThen()`.

3. This feature helps perform multiple operations sequentially.

## Conclusion

- Java 8 introduced powerful functional interfaces like \*\*Consumer\*\* to simplify operations on collections.

- The \*\*forEach()\*\* method, introduced in Java 8, takes a \*\*Consumer Functional Interface\*\* as input.

- We implemented real-world use cases to \*\*filter\*\*, \*\*print\*\*, and \*\*chain\*\* consumer functions.

- We also saw how \*\*Lambda expressions\*\* help in avoiding \*\*code duplication\*\* and improving readability.

- In the next tutorial, we will explore \*\*Predicate Functional Interface\*\* for advanced filtering.

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

Thank you for watching! Thanks a lot.