Object Oriented Programming in Java

Lambda Expressions

Suhel Hammoud

Writing Your First Lambda Expression

Java SE 8 introduced lambda expressions as a cleaner alternative to anonymous classes for specific use cases.

If you're unfamiliar with anonymous classes, don't worry — you can learn lambdas without them.

Writing a lambda involves three steps:

1. Identify the lambda type
2. Find the method to implement
3. Implement that method

Let’s explore each step.

▒▒▒ Identifying the Lambda Type

A lambda must implement a **functional interface** — an interface with **exactly one abstract method**.

Since Java 8, interfaces can have default and static methods, but only abstract methods count.

Note: `@FunctionalInterface` is optional but helps catch mistakes.

🟩 Examples of Functional Interfaces

```
@FunctionalInterface
public interface Runnable {
    void run();
}
```

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);
    default Consumer<T> andThen(
...) { }
}
```

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
    default Predicate<T> and(...) { }
    static <T> Predicate<T> isEqual(...) { }
}
```

Each has a single abstract method, making them valid functional interfaces.

▒▒ Finding the Method to Implement

Once you know the functional interface, find its abstract method:

- `Runnable`: `void run()`
- `Predicate<T>`: `boolean test(T t)`
- `Consumer<T>`: `void accept(T t)`

Your lambda is just an inline implementation of this method.

Writing Your First Lambda

To create a `Predicate<String>` that checks for 3-character strings:

```java
Predicate<String> predicate =
    (String s) -> {
        return s.length() == 3;
    };
```

Simplifying the Syntax

Java allows simplification:

```java
Predicate<String> predicate = s -> s.length() == 3;
```

- No need to declare types or use braces for single-line bodies
- Keep lambdas short and readable

▨ Other Examples

Consumer that prints a string:

```
Consumer<String> print = s -> System.out.println(s);
```

Runnable that prints a message:

```
Runnable runnable = () -> System.out.println("I am running");
```

Using Lambdas in Methods

Example using a lambda in a method:

```
List<String> retainStringsOfLength3(List<String> strings) {
    Predicate<String> predicate = s -> s.length() == 3;
    return strings.stream()
                  .filter(predicate)
                  .collect(Collectors.toList());
}
```

Lambdas implement an interface, so you can call its methods like `test()`.

🟩 Variable Capture in Lambdas

Lambdas can read **final or effectively final** variables but can't modify them.

Example:

```
int totalPrice = 0;
Consumer<Product> consumer = p -> totalPrice += p.getPrice(); // Error
```

Use an external mutable structure (e.g., `AtomicInteger`) for updates.

🟩 Lambda Serialization

Lambdas can be serialized, enabling their use in serializable objects.

This supports backward compatibility in classes that store functional fields.