Java Lambda Expressions

Streamline Your Code: A Guide to Java Lambdas

What is a Lambda?

A **lambda expression** is a short block of code which takes in parameters and returns a value.

- Simplifies code for single-use functions
- Often used in functional programming styles
- Helps eliminate boilerplate code

```
(parameter1, parameter2) -> expression
```

Why Use Lambdas?

- Makes code **more readable**
- **Reduces verbosity** by eliminating unnecessary classes
- Improves **code reusability**

Traditional Approach vs Lambda

Traditional Anonymous Class:

```
Comparator<Integer> comparator = new Comparator<Integer>() {
    @Override
    public int compare(Integer o1, Integer o2) {
        return o1.compareTo(o2);
    }
```

Anatomy of a Lambda Expression

A lambda expression is defined by three components:

1. **Parameters**: `(parameter1, parameter2, ...)`
2. **Arrow Operator**: `->`
3. **Body**: `expression` or `{ statements }`

Example:

```
(int a, int b) -> a + b
```

Functional Interfaces

A lambda expression can only be used with **functional interfaces**.
A **functional interface** is an interface with **one abstract method**.

```
@FunctionalInterface
interface MyFunctionalInterface {
    void myMethod();
}
```

Example with `Runnable`:

```
Runnable r = () -> System.out.println("Hello Lambda!");
```

Lambda in Collections

Lambdas can be used to process collections.

Example: Filtering a List

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.stream()
     .filter(name -> name.startsWith("A"))
     .forEach(System.out::println);
```

Output:

```
Alice
```

Method References

Simplify lambdas further using **method references**:

```
// Lambda
list.forEach(s -> System.out.println(s));

// Method Reference
list.forEach(System.out::println);
```

Use cases:

- `ClassName::staticMethod`
- `object::instanceMethod`
- `ClassName::new` (Constructor reference)

Real-World Example: Sorting

Traditional Sorting:

```
List<String> list = Arrays.asList("D", "B", "A");
Collections.sort(list, new Comparator<String>() {
    public int compare(String s1, String s2) {
        return s1.compareTo(s2);
    }
});
```

With Lambdas:

```
list.sort((s1, s2) -> s1.compareTo(s2));
```

With Method Reference:

Use Cases for Lambdas

- Sorting Collections
- Event handling in GUI applications
- Filtering and transforming data streams
- Runnable tasks in multithreading

Common Mistakes

- Using lambdas with non-functional interfaces
- Misunderstanding scoping rules
- Overcomplicating simple expressions

Practice Exercise

1. Convert the following code to use a lambda:

```java
ActionListener listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
};
```

2. Simplify using a method reference where possible.

Resources

- Java Lambda Basics
- Java Streams and Lambdas
- Common Lambda Use Cases

🎉

Now You're Ready for Lambdas!