CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Lambda Expressions in Java

Department of Computer Science
University of Maryland, College Park

Functional Interfaces in Java

- **Definition:** A functional interface has exactly one abstract method.
- **Key Point:** Can have multiple default or static methods.
- **Purpose:** They are the foundation for lambda expressions.
- **Common Examples:**
  - Runnable
  - Comparator<T>
  - Callable<T>
- Predefined interfaces in java.util.function:
  - Predicate<T>
  - Function<T, R>
  - Consumer<T>
- **Optional Declaration:**
  - You can declare a functional interface using the @FunctionalInterface annotation (this is not mandatory, but ensures the interface meets the requirement).

Motivation - Why Use Lambda Expressions?

- **Problems with Anonymous Inner Classes Before Java 8:**
  - Creating instances of **functional interfaces** (e.g., Runnable, Comparator<T>) required **verbose anonymous classes**.
  - Too much **boilerplate code** for simple operations.
- **How Lambda Expressions Help:**

  Reduces verbosity – Eliminates the need for anonymous classes.

  Improves readability – Code becomes more **concise and easier to understand**.

  Enables functional programming – Works well with **Streams API**, higher-order functions, and method references.

  - **See: BeforeJava8 and AfterJava8**

What is a Lambda Expression?

- **Definition:**
  - A **lambda expression** is an **anonymous function** that can be used to implement a **functional interface**.
  - It provides a **clear and concise** way to represent a **single method** implementation.
- **Syntax:** `(parameters) -> { body }`
- **Lambda Components:**

1. **Parameters** – Input values (can be zero, one, or multiple).
2. **Arrow (->)** – Separates parameters from the body.
3. **Body** – Defines the implementation logic (can be a single expression or a block {} for multiple statements).

Functional Interfaces & Lambda Expressions

- ***Reminder:*** **What is a Functional Interface?**
  - A **functional interface** is an interface that contains **exactly one abstract method**.
  - Lambda expressions can be used to implement them **without needing a class or an object**.
- **Examples of Functional Interfaces in Java:**
  - **Predefined Interfaces in java.util.function package**
    - Consumer<T> – Accepts an argument but returns nothing.
    - Supplier<T> – Returns a value but takes no argument.
    - Function<T, R> – Takes an argument of type T and returns R.
    - Predicate<T> – Evaluates a condition and returns true or false.
      - **See** `FunctionalInterfacesExample`

Lambda Expression Syntax Breakdown

- **Basic Syntax Variations**
- **Zero parameters:**

```
() -> System.out.println("Hello, World!");
```

- **One parameter (parentheses optional): `x -> x * x`**
- // If only one parameter, parentheses can be omitted
- **Multiple parameters: `(x, y) -> x + y`**
- **Block body (for multiple statements):**

```
(x, y) -> { int sum = x + y;
            return sum; }
```

Using a return statement:

- **Single-line expressions automatically return a value** (no need for return).
- **If using {} (block body), you must explicitly use return.**

See `LambdaVariations`