



Java Training

Jan - Feb 2023

Lambda Expressions

- Helps us to **write** our code in **functional** style.
- Describe what you want, rather than how to get it.
- Provides a clear way to **implement Single Abstract Method** by using an expression
- It is very **useful** in collection library in which it helps to **iterate, filter and extract** data.

Lambda Expressions Syntax

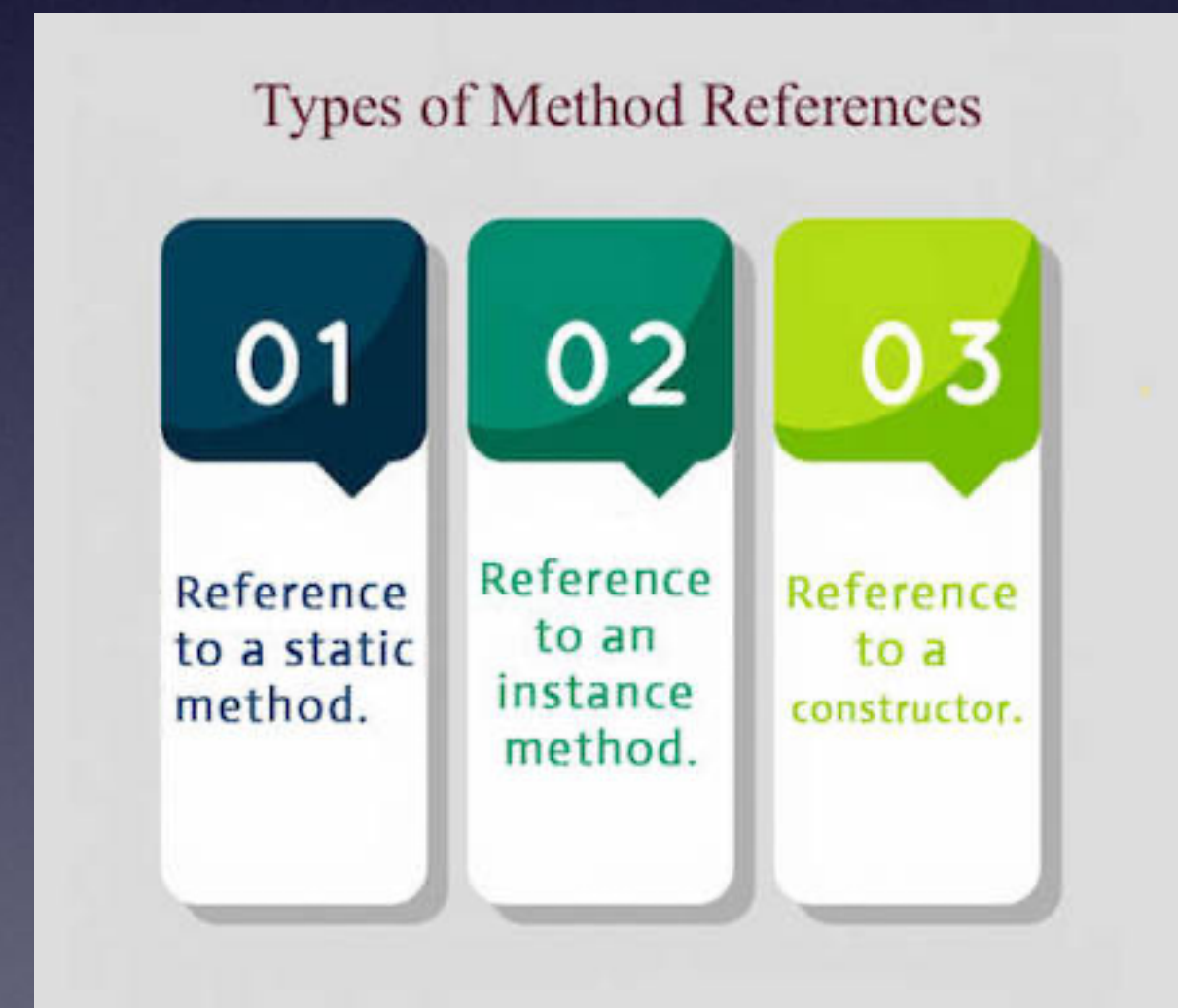
- Standard Syntax
- Parameter Type
- Multiple Lines of Code
- Single Parameter with Inferred Type
- Method References
- No Parameter

The diagram illustrates the standard syntax of a lambda expression. It shows the code `(int arg1, String arg2) -> {System.out.println("Two arguments "+arg1+" and "+arg2);}` with three components labeled below:
1. **Argument List**: Points to the parameter list `(int arg1, String arg2)`.
2. **Arrow token**: Points to the lambda arrow `->`.
3. **Body of lambda expression**: Points to the code block `{System.out.println("Two arguments "+arg1+" and "+arg2);}`.

Method Reference

- It is used to refer method of **functional interface** .
- It is **compact** and easy form of lambda expression.

- 01: ContainingClass::staticMethodName
- 02: containingObject::instanceMethodName
- 03: ClassName::**new**



Functional Interface

- An **Interface** that contains only **one** abstract **method**.
- It **can** have **any** number of **default** and static **methods**.
- It **can** also declare **methods** of **object class**.
- Also known as Single Abstract Method Interfaces.

StringJoiner

- Used for **joining** Strings making use of a **delimiter**, **prefix**, & **suffix**.
- The **default** value is returned only when the StringJoiner is **empty**.
- **merge()** : It **adds** the contents of the given StringJoiner **without prefix and suffix** as the **next element**.
- **Collectors.joining()** **internally** uses StringJoiner to perform the joining operation.

Optional

- It is a `public final` class.
- Which is used to deal with `NullPointerException` in Java application.
- Provides methods to `check` the presence of `value`.

forEach

- Java provides a new method `forEach()` to iterate the elements.
- It is defined in `Iterable` and `Stream` interfaces.
- It is a default method defined in the `Iterable` interface.
- `Collection` classes which extends `Iterable` interface can use `forEach()` method to iterate elements.
- This method takes a single parameter which is a functional interface.
- So, you can pass lambda expression as an argument.

Date/Time API

- Java has introduced a new Date and Time API since Java 8.
- The `java.time` package contains Java 8 Date and Time classes.

Drawbacks of existing Date/Time API's

- **Thread safety:** The existing classes such as Date and Calendar does not provide thread safety. Hence it leads to hard-to-debug concurrency issues that are needed to be taken care by developers. The new Date and Time APIs of Java 8 provide thread safety and are immutable, hence avoiding the concurrency issue from developers.
- **Bad API designing:** The classic Date and Calendar APIs does not provide methods to perform basic day-to-day functionalities. The Date and Time classes introduced in Java 8 are ISO-centric and provides number of different methods for performing operations regarding date, time, duration and periods.
- **Difficult time zone handling:** To handle the time-zone using classic Date and Calendar classes is difficult because the developers were supposed to write the logic for it. With the new APIs, the time-zone handling can be easily done with Local and ZonedDateTime APIs.

Default Methods

- Java provides a facility to create default **methods** inside the **interface**.
- Methods which are defined inside the interface and **tagged** with **default keyword** are known as default methods.
- These methods are **non-abstract** methods and can **have** method **body**.

Why Default Method ?

- Before Java 8, interfaces could have only abstract methods.
- The implementation of these methods has to be provided in a separate class.
- So, if a new method is to be added in an interface, then its implementation code has to be provided in the class implementing the same interface.
- To overcome this issue, Java 8 has introduced the concept of default methods which allow the interfaces to have methods with implementation without affecting the classes that implement the interface.

Nashorn JavaScript Engine

- Nashorn is a JavaScript engine.
- It is used to execute JavaScript code dynamically at JVM.
- You can execute JavaScript code by two ways
 - Using jjs command-line tool, and
 - By embedding into Java source code.

Nashorn cont.

- The new default JavaScript engine for the JVM as of Java 8.
- Many sophisticated techniques have been used to make Nashorn orders of magnitude more performant than its predecessor called Rhino, so it is a worthwhile change.

Collectors

- Collectors is a final class that extends Object class.
- It provides reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria etc.

Stream

- Java 8 `java.util.stream` package consists of classes, interfaces and an enum to allow functional-style operations on the elements.
- It performs lazy computation. So, it executes only when it requires.

Java Base64

- Basic Encoding and Decoding
 - A-Za-z0-9+/_
- URL and Filename Encoding and Decoding
 - A-Za-z0-9+_
• url and file name safe
- MIME
 - MIME friendly format
 - Represented in lines of no more than 76 characters each