Java 8

Content

- Introduction
- Lambda
- Functional IFC
- Method References
- Default Method
- Static Method
- Optional Class
- Stream API
- Base64
- Date and Time API

Java 8

Introduction

- Released in March 2014.
- Major update with several new features and enhancements.
- Focus on improving developer productivity and language capabilities.
- Emphasis on functional programming.

Java 8

Lambda

- Enables functional programming in Java.
- Removes boilerplate code for anonymous inner classes.
- Useful in collection operations.
- Simplifies the implementation of functional interfaces.

- Syntax:
  - (parameters) -> expression
  - (parameters) -> { statements; }

```
(int arg1, String arg2) -> {System.out.println("Two arguments "+arg1+" and "+arg2);}
```

Argument List — Arrow token — Body of lambda expression

Java 8

Lambda

Lambda Expression Syntax Parameters:

- **Zero Parameter :**

```
() -> System.out.println("Zero parameter lambda");
```

- **Single Parameter:**

```
(p) -> System.out.println("One parameter: " + p);
```

- **Multiple parameters**

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);
```

Java 8

Functional IFC

- A functional interface is an interface that contains only one single abstract method.
- Functional interfaces are a key part of the Java 8 features, particularly when working with lambda expressions.
- Functional interfaces can have multiple default or static methods.
- Annotated with @FunctionalInterface for clarity.

Java 8

Functional IFC

- Common Functional Interfaces:

1. **Supplier<T>:** Represents a supplier of results.
2. **Consumer<T>**: Represents an operation that accepts a single input argument and returns no result.
3. **Predicate<T>**: Represents a predicate (boolean-valued function) of one argument.
4. **Function<T, R>**: Represents a function that takes one

Java 8

Method References

Short-hand for calling a method via lambda expression.
Types:

- Static methods: ClassName::staticMethodName

```
// Static method reference
Consumer<List<Integer>> sortList = Collections::sort;
```

- Instance methods: instance::instanceMethodName

```
// Instance method reference
Function<String, String> toUpper = String::toUpperCase;
```

- Constructors: ClassName::new

```
// Constructor reference
Supplier<List<String>> listSupplier = ArrayList::new;
```

Java 8

Default Method

- Default methods provide a way to add new functionality to interfaces without forcing all classes that implement the interface to provide an implementation for the new methods.

- A default method is defined using the default keyword followed by the method signature and implementation.

- The implementation of the default method in the implemented classes is **optional in General** .

- The implementation of the default method is **mandatory** just in case of **class implements multiple interfaces that have default methods with the same signature .**

Java 8

Static Method

- Java 8 allows interfaces to have static methods with method bodies.
- These static methods can be called directly on the interface, without the need for an implementing class.
- Unlike abstract methods, static methods in interfaces can provide a default implementation.
- If a class implementing the interface does not provide its own implementation, the default implementation in the interface is used.

Java 8

Optional

- **Optional** class was introduced to provide a more robust way of handling situations where a value may or may not be present.
- **Optional** is designed to address the problem of dealing with null values and to encourage better practices in handling the absence of a value (**NullPointerException**) .
- **Optional** is an **immutable** object, and its methods do not modify the instance but instead return a new one.
- **Optional** often used as a return type for methods to indicate that the result may or may not be present.

Java 8

Optional

Purpose:

- Container for optional values.
- Avoids NullPointerException.

Common Methods:

- isPresent(): checks if a value is present.
- ifPresent(): executes a block of code if a value is present.
- orElse(): provides a default value if a value is not present.

Example :

```
Optional<String> optional = Optional.ofNullable(null);
optional.ifPresent(System.out::println);
String value = optional.orElse("Default Value");
System.out.println(value); // Output: Default Value
```

Java 8

Streams

- **Streams** is a new abstraction introduced in Java 8 to express operations on data in a concise and functional manner.
- **Streams** is a sequence of elements that can be processed in a functional style.
- **Streams** allow for processing collections of data in a declarative way, similar to SQL queries.
- **Streams methods** :

Filter	Map
ForEach	Collect
Reduce	Sorted
Count	Distinct

Java 8

Base64

- Base64 is a binary-to-text encoding scheme that represents binary data in an ASCII string format.

- Base64 is used to encode binary data, such as images, audio files, or any binary data, into a text-based format.

- Base64 encoding is not a form of encryption; it's simply a way to represent binary data in a format that is safe for transportation in text-based protocols like
    - email
    - HTML
    - XML documents

- Base64 encoding uses a set of 64 characters :
    - The 26 uppercase letters
    - The 26 lowercase letters
    - The 10 digits
    - The '+'
    - The '/' characters.

Java 8

Date and Time API

- Modern API for date and time manipulation.
- Solves issues with the old java.util.Date and java.util.Calendar.
- Thread-safe and immutable.
- **Key Classes:**
    - LocalDate: Date without time.
    - LocalTime: Time without date.
    - LocalDateTime: Date and time.
    - ZonedDateTime: Date and time with time zone.

Thank You !!