Java 8 Features Documentation

1. Lambda Expressions

Lambda expressions introduce functional programming to Java. They are essentially anonymous methods that allow you to treat code as data or pass functions as arguments.

- **Syntax:** (parameters) -> { body }
- **Why use it?** It significantly reduces "boilerplate" code, especially when working with collections and listeners.

---

2. Functional Interfaces

A functional interface is an interface that has **exactly one abstract method**. They are the "target types" for lambda expressions.

- **Annotation:** @FunctionalInterface (Optional, but recommended to prevent adding more abstract methods).
- **Common Examples:** * Predicate<T>: Takes an input and returns a boolean.
    - Function<T, R>: Takes an input T and returns a result R.
    - Consumer<T>: Takes an input and returns nothing (void).

---

3. Method References

Method references are a shorthand for lambdas that simply call an existing method by name. They make the code much more readable.

Type	**Syntax**	**Lambda Equivalent**
Static	ClassName::methodName	(x) -> ClassName.methodName(x)
Instance	instance::methodName	(x) -> instance.methodName(x)
Constructor	ClassName::new	() -> new ClassName()

---

4. Stream API

The Stream API is a powerful way to process sequences of elements (like Collections) in a declarative way. It supports "filter-map-reduce" transformations.

- **Intermediate Operations:** (Lazy) filter(), map(), sorted(), distinct().
- **Terminal Operations:** (Triggers execution) collect(), forEach(), count(), reduce().

---

5. Collectors API

Collectors are utility methods used at the end of a Stream to transform the result into a specific structure, like a List, Set, or a Map.

- **Grouping:** Collectors.groupingBy() allows you to categorize data (e.g., grouping employees by department).
- **Joining:** Collectors.joining(", ") concatenates string elements with a delimiter.

---

6. Optional Class

java.util.Optional is a container object used to represent a value that might be null. It provides a safer alternative to returning null and prevents NullPointerException.

- **Key Methods:** ofNullable(), isPresent(), ifPresent(), and orElse().

---

7. Default and Static Methods in Interfaces

Before Java 8, interfaces could only have abstract methods.

- **Default Methods:** Defined with the default keyword. They allow you to add new methods to interfaces without breaking the classes that already implement them.
- **Static Methods:** Utility methods that belong to the interface itself, not the implementing object.

---

8. Parallel Streams

Parallel streams automatically partition the stream into multiple chunks, processing them on different CPU cores using the **Fork/Join framework**.

- **Usage:** Call .parallelStream() instead of .stream().
- **When to use:** Use for very large datasets and operations that are CPU-intensive and independent.

---

9. Date & Time API (java.time)

A complete overhaul of the old, buggy java.util.Date. The new API is immutable, thread-safe, and follows ISO standards.

- **Local Classes:** LocalDate, LocalTime, LocalDateTime (no timezone).
- **Zoned Classes:** ZonedDateTime (handles timezones).
- **Periods & Durations:** To measure the distance between two points in time.

---

10. Spliterator & Lambda Internals

- **Spliterator:** An interface for traversing and partitioning sequences of elements. It is the core engine that makes Parallel Streams possible by "splitting" the work.
- **Internals (invokedynamic):** Unlike anonymous classes, Lambdas are not compiled into separate .class files. The JVM uses a special instruction called invokedynamic to create the function at runtime, which is much more memory-efficient.