Key Java 8 features :

🔶 1. Lambda Expressions

- **What**: A lambda expression is essentially an anonymous function — a concise block of code that can be passed around as data.
- **Why**: It simplifies the syntax for functional programming and eliminates boilerplate code like anonymous class declarations.
- **Where**: Perfect for event listeners, sorting, filtering collections, and passing behavior as parameters to methods.

🔶 2. Functional Interfaces

- **What**: Interfaces that have just **one abstract method** are called functional interfaces (e.g., `Runnable`, `Predicate`, `Function`).
- **Why**: They enable lambdas and method references, ensuring type safety while allowing functional programming.
- **Where**: Used extensively with Java 8's built-in functions (`java.util.function`) and custom behavior abstractions.

🔶 3. Stream API

- **What**: A declarative API to process data from collections (like Lists or Sets) using a sequence of operations — like `filter`, `map`, `reduce`.
- **Why**: Replaces verbose for-loops with readable, functional-style operations, and enables parallel processing.
- **Where**: Common in data processing pipelines, filtering large lists, transforming data structures, or aggregating results.

🔶 4. Method References

- **What**: A shorthand notation to refer directly to methods or constructors using `::` syntax (e.g., `System.out::println`).
- **Why**: Reduces verbosity when the lambda is only calling an existing method.
- **Where**: Inside stream operations or anywhere a lambda is used to invoke a method.

🔶 5. Optional Class

- **What**: A container for optional (possibly null) values — instead of returning `null`, you return `Optional.empty()` or `Optional.of()`.

- **Why**: Eliminates common `NullPointerExceptions` and encourages safe value-checking using methods like `isPresent()`, `orElse()`, and `map()`.
- **Where**: Frequently used in service and repository layers when a method may or may not return a value.

🔶 6. Default & Static Methods in Interfaces

- **What**: Java 8 allows interfaces to have `default` and `static` method implementations.
- **Why**: Helps in interface evolution without breaking existing implementations — especially useful for adding new features in APIs.
- **Where**: Framework interfaces like `List`, `Map`, `Comparator` use this to offer utility and sorting methods directly.

🔶 7. New Date and Time API (java.time)

- **What**: A modern, immutable date-time API introduced in `java.time` package, replacing the problematic `java.util.Date`.
- **Why**: Offers thread-safety, better design, and fluent API for manipulating time zones, formatting, parsing, etc.
- **Where**: Everywhere you need to handle date and time in apps — from logs to expiration dates to time zone conversions.

🔶 8. Collectors & Terminal Operations

- **What**: Tools that allow you to **gather**, **group**, or **summarize** data from streams using `Collectors.toList()`, `groupingBy()`, etc.
- **Why**: Makes aggregation logic more readable and efficient in one-liners.
- **Where**: When working with large data sets or transforming the result of stream operations into collections or maps.