Java 8 introduced a lot of exciting features that brought functional programming and cleaner code to Java. Here's a breakdown of the key features:

**1. Lambda Expressions**

Lambda expressions enable functional programming, making code more concise, readable, and expressive. It allows you to pass behavior as parameters to methods.

**Syntax:**

java

CopyEdit

(parameters) -> expression

**Example:**

java

CopyEdit

List<String> names = Arrays.asList("John", "Sarah", "Michael");

names.forEach(name -> System.out.println(name));

Here, name -> System.out.println(name) is the lambda expression. It replaces the need for a full loop.

**2. Functional Interfaces**

Functional interfaces have exactly one abstract method and are used primarily with lambda expressions.

**Example:**

java

CopyEdit

@FunctionalInterface

interface Calculator {

int add(int a, int b); // Single abstract method

default void greet() {

System.out.println("Hello!");

}

}

Calculator calc = (a, b) -> a + b;

System.out.println(calc.add(5, 3)); // 8

calc.greet(); // Hello!

Java provides built-in functional interfaces like:

* **Predicate**: Takes an argument and returns a boolean.
* **Function**: Takes one argument and returns a result.
* **Consumer**: Accepts an argument and performs an action.
* **Supplier**: Supplies a result.

**3. Stream API**

The Stream API provides functional-style operations to process collections, enabling cleaner and more readable code.

**Example:**

java

CopyEdit

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

int sum = numbers.stream()

.filter(n -> n % 2 == 0) // Filter even numbers

.mapToInt(Integer::intValue) // Convert Integer to int

.sum(); // Terminal operation: sum the values

System.out.println("Sum of even numbers: " + sum); // Output: 6

* **Intermediate operations**: .filter(), .map()
* **Terminal operations**: .collect(), .forEach(), .reduce()

**4. Default and Static Methods in Interfaces**

Interfaces can now have default and static methods.

* **Default Methods**: Methods with a body that can be used without overriding.
* **Static Methods**: Can be called directly from the interface.

**Example:**

java

CopyEdit

interface MyInterface {

default void greet() {

System.out.println("Hello from default method!");

}

static void staticMethod() {

System.out.println("Hello from static method!");

}

}

MyInterface.staticMethod(); // Calling static method

MyInterface obj = new MyInterface() {};

obj.greet(); // Calling default method

**5. Optional Class**

Optional is a container object which may or may not contain a non-null value. It's a safer way to handle null and avoid NullPointerException.

**Example:**

java

CopyEdit

Optional<String> optionalName = Optional.ofNullable(null);

optionalName.ifPresent(name -> System.out.println("Name: " + name));

String result = optionalName.orElse("Default Name");

System.out.println("Result: " + result); // Default Name

**6. New Date and Time API (java.time)**

The new Date and Time API provides a better way to handle dates and times. It follows the ISO-8601 standard and addresses many issues of the older Date and Calendar classes.

**Example:**

java

CopyEdit

LocalDate today = LocalDate.now();

LocalDateTime now = LocalDateTime.now();

System.out.println("Today's Date: " + today);

System.out.println("Current Date and Time: " + now);

LocalDateTime futureDate = now.plusDays(10).plusHours(5);

System.out.println("Future Date and Time: " + futureDate);

LocalDate parsedDate = LocalDate.parse("2025-05-20");

System.out.println("Parsed Date: " + parsedDate);

**Summary of Key Java 8 Features:**

* **Lambda Expressions**: Concise and functional programming style.
* **Functional Interfaces**: Interfaces with a single abstract method, used with lambdas.
* **Stream API**: Enables functional operations on collections, making them easier to process.
* **Default and Static Methods**: Interfaces can have methods with a body.
* **Optional Class**: Provides a safer way to handle null values.
* **New Date and Time API**: A modern and more reliable API for working with dates and times.

These features make Java much more expressive, efficient, and functional! Let me know if you'd like to explore any of these in more detail.