Sure, here are explanations and examples of the Java 8 features you mentioned:

\*\*1. Functional Interfaces:\*\*

Functional interfaces are interfaces that have exactly one abstract method. They are a core concept in Java 8, as they enable the use of lambda expressions and method references. Java provides several built-in functional interfaces, such as `Predicate`, `Function`, and `Consumer`.

Example using `Predicate` functional interface:

```java

import java.util.function.Predicate;

public class FunctionalInterfaceExample {

public static void main(String[] args) {

Predicate<Integer> isEven = num -> num % 2 == 0;

System.out.println(isEven.test(4)); // Output: true

System.out.println(isEven.test(7)); // Output: false

}

}

```

\*\*2. Lambda Expressions:\*\*

Lambda expressions are a concise way to express instances of functional interfaces. They provide a way to define anonymous functions in a compact and readable manner.

Example using lambda expression with `Comparator`:

```java

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class LambdaExpressionExample {

public static void main(String[] args) {

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

// Using lambda expression for sorting

Collections.sort(names, (name1, name2) -> name1.compareTo(name2));

System.out.println(names); // Output: [Alice, Bob, Charlie]

}

}

```

\*\*3. Stream API methods with examples:\*\*

The Stream API introduced in Java 8 provides a powerful way to work with collections and perform various operations in a functional style. Streams allow for concise and parallel processing of data.

Example using Stream API for filtering and mapping:

```java

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class StreamAPIExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

List<Integer> evenSquares = numbers.stream()

.filter(num -> num % 2 == 0)

.map(num -> num \* num)

.collect(Collectors.toList());

System.out.println(evenSquares); // Output: [4, 16, 36, 64, 100]

}

}

```

These examples showcase some of the key features introduced in Java 8: functional interfaces, lambda expressions, and the Stream API. These features have significantly enhanced Java's capabilities for expressing functional programming concepts and simplifying the manipulation of data and collections.