Java 8 Features - Interview Question

Java 8 introduced several significant enhancements to the language and the Java Development Kit (JDK). Below is a detailed explanation of its major features.

1. Lambda Expressions

- Description: Lambda expressions allow you to write concise and flexible code by providing a way to express instances of single-method interfaces (functional interfaces) in a more compact form.
- Syntax: (parameters) -> expression or (parameters) -> { statements }

Example:

```java
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.forEach(name -> System.out.println(name));
```

2. Functional Interfaces

- Description: A functional interface is an interface with exactly one abstract method. Java 8 introduced the @FunctionalInterface annotation to ensure that an interface meets this condition.

Example:

```java
@FunctionalInterface
interface MyFunctionalInterface {
    void myMethod();
}
```

3. Streams API

- Description: The Streams API allows you to process sequences of elements (such as collections) in a functional style. It supports operations such as filtering, mapping, and reducing.

Example:

```java
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
List<String> filteredNames = names.stream()
                                  .filter(name -> name.startsWith("A"))
                                  .collect(Collectors.toList());
```

4. Default Methods

- Description: Interfaces can now have default methods with a body. This allows you to add new methods to interfaces without affecting the classes that implement the interface.

Example:

```java
interface MyInterface {
    default void defaultMethod() {
        System.out.println("Default method");
    }
}
```

5. Static Methods in Interfaces

- Description: Java 8 allows interfaces to have static methods. These methods belong to the interface itself, not to instances of classes implementing the interface.

Example:

```java
interface MyInterface {
    static void staticMethod() {
        System.out.println("Static method");
    }
}
```

6. Method References

- Description: Method references provide a way to refer to methods or constructors directly using their names. They can be used to simplify lambda expressions.
- Syntax: ClassName::methodName

Example:

```java
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.forEach(System.out::println);
```

7. Optional Class

- Description: The Optional class is a container object which may or may not contain a value. It is used to avoid null references and to represent the presence or absence of a value.

Example:

```java
Optional<String> optional = Optional.of("Hello");
optional.ifPresent(value -> System.out.println(value));
```

8. New Date and Time API

- Description: Java 8 introduced a new Date and Time API in java.time package, which is more comprehensive and user-friendly compared to the old java.util.Date and java.util.Calendar classes.

Example:

```java
LocalDate date = LocalDate.now();
LocalDate futureDate = date.plusDays(10);
System.out.println(futureDate);
```

9. Nashorn JavaScript Engine

- Description: Nashorn is a new JavaScript engine that replaces the older Rhino engine. It provides better performance and compliance with the JavaScript standard.

Example:

```java
ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");
engine.eval("print('Hello from Nashorn!');");
```

10. Parallel Streams

- Description: The Streams API supports parallel processing of data, allowing for concurrent execution of operations on data streams.

Example:

```java
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.parallelStream().forEach(name -> System.out.println(name));
```

11. CompletableFuture

- Description: CompletableFuture is part of the java.util.concurrent package and provides a way to write asynchronous, non-blocking code. It supports a variety of methods for combining and managing asynchronous tasks.

Example:

```java
CompletableFuture.supplyAsync(() -> {
    return "Hello";
}).thenAccept(result -> {
    System.out.println(result);
});
```

12. New Functional Interfaces in java.util.function

- Description: Java 8 introduced several new functional interfaces in the java.util.function package, such as Function, Predicate, Consumer, Supplier, and others, which facilitate functional programming.

Example:

```java
Function<String, Integer> stringToLength = String::length;
System.out.println(stringToLength.apply("Hello")); // Output: 5
```