**Java 8 Features**

**Create a new branch feature-java8 as part of your github repo rg-assignments**

**and save/push all your coding solutions in the same branch**

**Please share your output screenshots in the assignment document along with the github link for each question. Provide an explanation wherever possible as part of your response :-)**

1. **List the features of Java 8**

* Lambda Expressions – Enables functional-style programming and simplifies code.
* Functional Interfaces – Interfaces with a single abstract method, used with lambdas.
* Stream API – Processes collections of objects in a functional and parallelizable way.
* Default & Static Methods in Interfaces – Allows methods in interfaces with implementation.
* Optional Class – Avoids null checks and NullPointerException.
* Date and Time API – Provides a modern date/time handling mechanism
* Method References – Shortcut for calling methods via lambdas.
* Collectors – Helps in collecting elements from streams into lists, sets, etc.
* Nashorn JavaScript Engine – Embeds JavaScript execution within Java.
* New APIs & Enhancements – Like forEach, removeIf, Comparator changes, etc.

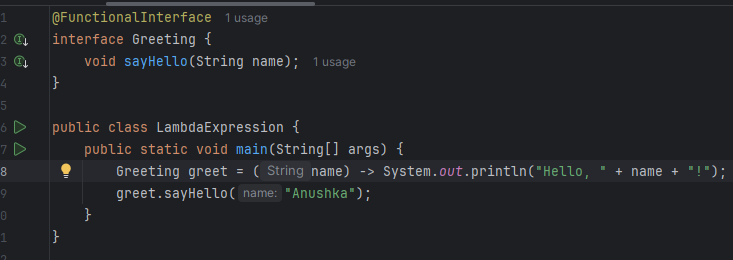
1. **What is a Lambda Expression, and why do we use them? Explain with a coding example and share the output screenshot.**

A Lambda Expression is a feature in Java 8 that allows us to write anonymous methods (functions) in a concise way. It's mostly used to implement functional interfaces (interfaces with a single abstract method).

Using Lambda Expressions also:

* Reduces boilerplate code
* Makes code more readable

**Code:**

****

**Output:**

****

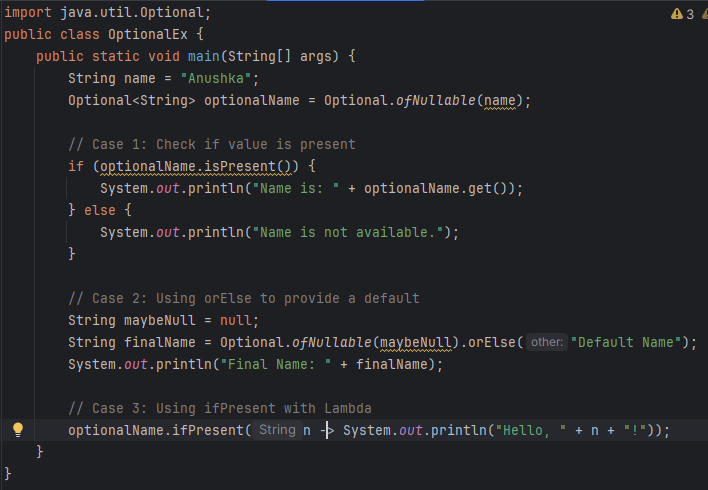
Lambda expressions in Java 8 allow us to treat functionality as a method argument. In this example, instead of creating an implementation class for the ***Greeting*** interface, we directly provide a concise implementation using a lambda expression. It improves code clarity and reduces overhead.

**Github link :** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java8/LambdaExpression.java>

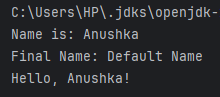
1. **What is optional, and what is it best used for? Explain with a coding example and share the output screenshot.**

Optional<T> is a container object used to handle null values. It helps avoid NullPointerException by explicitly representing the presence or absence of a value. **Github link:** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java8/OptionalEx.java>

**Code:**

****

**Output:**

****

Optional is a container class to represent optional values. It makes the code more readable and robust by avoiding unnecessary null checks. In the example:

* used isPresent() and get() to check and retrieve the value.
* orElse() helps provide a fallback
* ifPresent() executes a lambda if the value exists

1. **What is a functional interface? List some examples of predefined functional interfaces.**

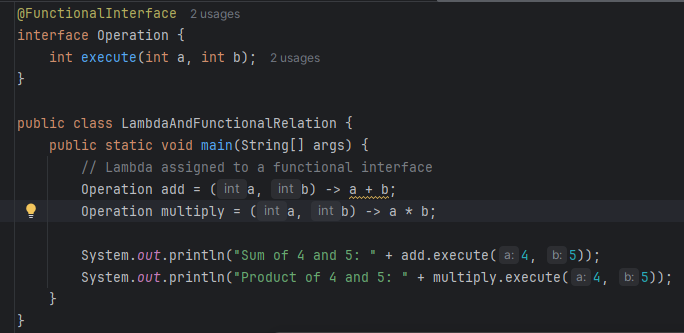
A Functional Interface in Java is an interface that contains only one abstract method. These interfaces can have multiple default or static methods, but only one abstract method. They enable writing cleaner functional code.

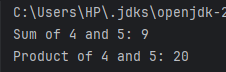
Some predefined functional interfaces are as follows:

| **Interface** | **Package** |  |
| --- | --- | --- |
| Runnable | java.lang |  |
| Callable<T> | java.util.concurrent |  |
| Predicate<T> | java.util.function |  |
| Function<T,R> | java.util.function |  |
| Consumer<T> | java.util.function |  |
| Supplier<T> | java.util.function |  |

1. **How are functional interfaces and Lambda Expressions related?**

A lambda expression provides an implementation for a method defined in a functional interface. Java uses type inference to map a lambda expression to a compatible functional interface. Without functional interfaces, lambda expressions would have no target to bind to.

**Code:**

**Output**

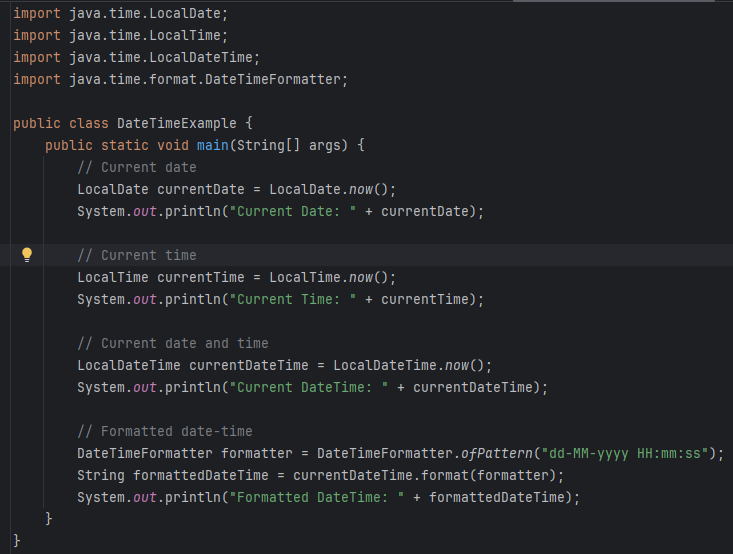
Lambda expressions work by providing an implementation for a method defined in a functional interface. In this example, **Operation** is a functional interface. We use lambda expressions to define two operations—addition and multiplication—demonstrating the direct link between lambdas and functional interfaces.

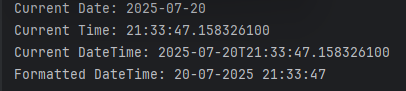
**Github link:** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java8/LambdaAndFunctionalRelation.java>

1. **List some Java 8 Date and Time API’s. How will you get the current date and time using Java 8 Date and Time API? Write the implementation and share the output screenshot.**

We can use the newly introduced a new Date and Time API in **the java.time** package.

**Github link:** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java8/DateTimeExample.java>

**Code:**

**Output:**

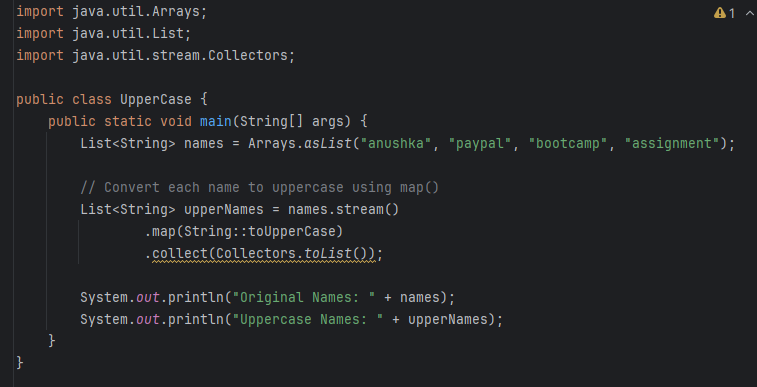
Java 8 introduced immutable date/time API under *java.time*. It improves thread-safety, clarity, and ease of use. This example shows how to fetch and format current date and time using **LocalDate, LocalTime, LocalDateTime**, and **DateTimeFormatter**.

1. **How to use map to convert objects into Uppercase in Java 8? Write the implementation and share the output screenshot.**

We can use *map()* to convert a list of strings to uppercase.

**Github link :** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java8/UpperCase.java>

**Code:**

****

**Output:**



1. **Explain how Java 8 has enhanced interface functionality with default and static methods. Why were these features introduced, explain with a coding example?**

Prior to Java 8, interfaces could only have abstract methods. Java 8 introduced:

* default methods: Provide a method body inside interfaces.
* static methods: Define utility methods in interfaces.

These features were introduced to:  
 1. add new functionality to interfaces without breaking existing implementation classes.

2. allow shared logic within the interface itself.

3. To support functional programming and better API evolution.

**Github Link:** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java8/InterfaceDefaultStatic.java>

**Code:**

**interface Vehicle {**

**void start();**

***// Default method***

**default void fuelType() {**

**System.*out*.println("Default fuel type: Petrol");**

**}**

***// Static method***

**static void serviceReminder() {**

**System.*out*.println("Static: Service due in 6 months.");**

**}**

**}**

**class Car implements Vehicle {**

**@Override**

**public void start() {**

**System.*out*.println("Car is starting...");**

**}**

**@Override**

**public void fuelType() {**

**System.*out*.println("Fuel type: Diesel");**

**}**

**}**

**public class InterfaceDefaultStatic {**

**public static void main(String[] args) {**

**Car myCar = new Car();**

**myCar.start(); *// abstract method***

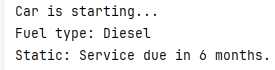
**myCar.fuelType(); *// overridden default method***

**Vehicle.*serviceReminder*();**

**}**

**}**

**Code:**

****

Java 8 allows default and static methods in interfaces. This enables backward-compatible interface enhancements. In the example:

* fuelType() is a default method that can be overridden.
* serviceReminder() is a static utility method, accessed using the interface name.

### **Discuss the significance of the Stream API introduced in Java 8 for data processing. How does it improve application performance and developer productivity?**

The Stream API in Java 8 allows you to process collections functionally and declaratively, without writing explicit loops. This ensures cleaner and concise code and also enables parallel processing.

Some Common Stream Operations:

* filter() – filters elements based on a condition
* map() – transforms each element
* sorted() – sorts the stream
* collect() – collects results into a list/set/map
* forEach() – performs an action on each element
* count(), min(), max() – terminal operations

### **What are method references in Java 8, and how do they complement the use of lambda expressions? Provide an example where a method reference is more suitable than a lambda expression. Explain with a coding example and share the output screenshot.**

Method references are a shorthand notation of lambda expressions to call a method directly.

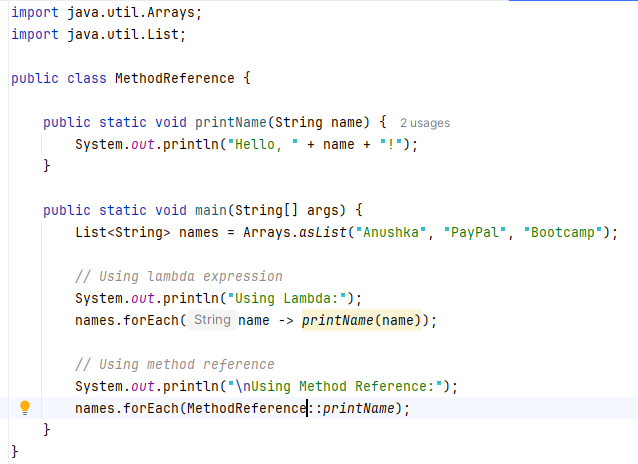
Instead of writing: list.forEach(str -> System.out.println(str));

We can write: llist.forEach(System.out::println);

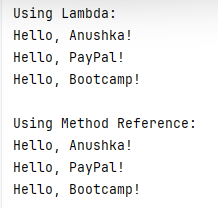
**Types of Method References:**

| **Syntax** | **Description** |
| --- | --- |
| ClassName::staticMethod | Reference to a static method |
| objectRef::instanceMethod | Reference to an instance method |
| ClassName::instanceMethod | Reference to an instance method of arbitrary object |
| ClassName::new | Reference to a constructor |

**Code:**

****

**Output:**

****

**Github link :** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java8/MethodReference.java>