1. **List the features of Java 8?**
2. **Lambda Expressions** → Enables writing anonymous methods in a concise way.  
    → Helps pass behavior as arguments (functional programming).
3. **Functional Interfaces** → Interfaces with a **single abstract method** (e.g., Runnable, Comparator).  
    → Required to use lambda expressions.
4. **Streams API** → Processes collections (like List, Set) using operations like map, filter, reduce.  
    → Supports parallel and sequential data processing.
5. **Default Methods in Interfaces** → Interfaces can now have method bodies using the default keyword.  
    → Helps evolve interfaces without breaking existing implementations.
6. **Static Methods in Interfaces** → Interfaces can now include static utility methods.
7. **Method References** → Shorthand for calling methods using ::.  
    → Example: System.out::println.
8. **Optional Class** → Helps handle null values gracefully without NullPointerException.  
    → Acts as a container object.
9. **New Date and Time API (java.time)** → Modern replacement for old Date and Calendar classes.  
    → Immutable, thread-safe, and easier to use.
10. **Collectors (Stream terminal operations)** → Used to collect results from Stream operations (e.g., toList(), toMap()).
11. **Nashorn JavaScript Engine** → Allows executing JavaScript code within Java applications (deprecated in later versions).

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

**Sure! Here's a crisp and graduate-level explanation of Lambda Expressions:**

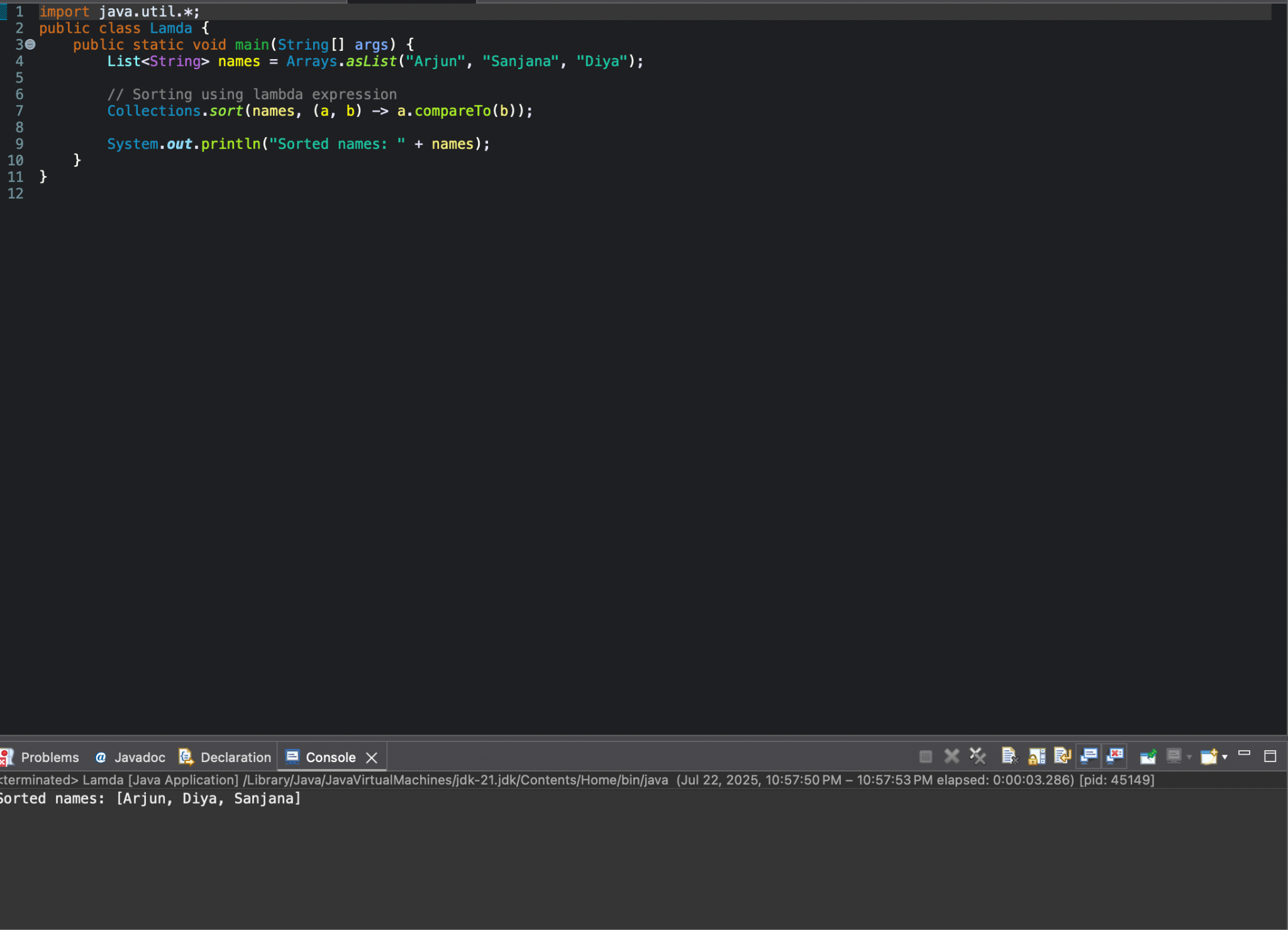
What is a Lambda Expression?

**A Lambda Expression in Java is an anonymous function—a concise way to represent a method using just parameters and a body.**

**It enables functional programming by treating behavior as data, allowing functions to be passed as arguments.**

Why Do We Use Them?

* **To write cleaner, shorter, and more readable code.**
* **To implement functional interfaces (single-method interfaces) efficiently.**
* **To enable higher-order functions like map(), filter(), forEach() in the Streams API.**
* **Reduces boilerplate compared to anonymous inner classes.**

****

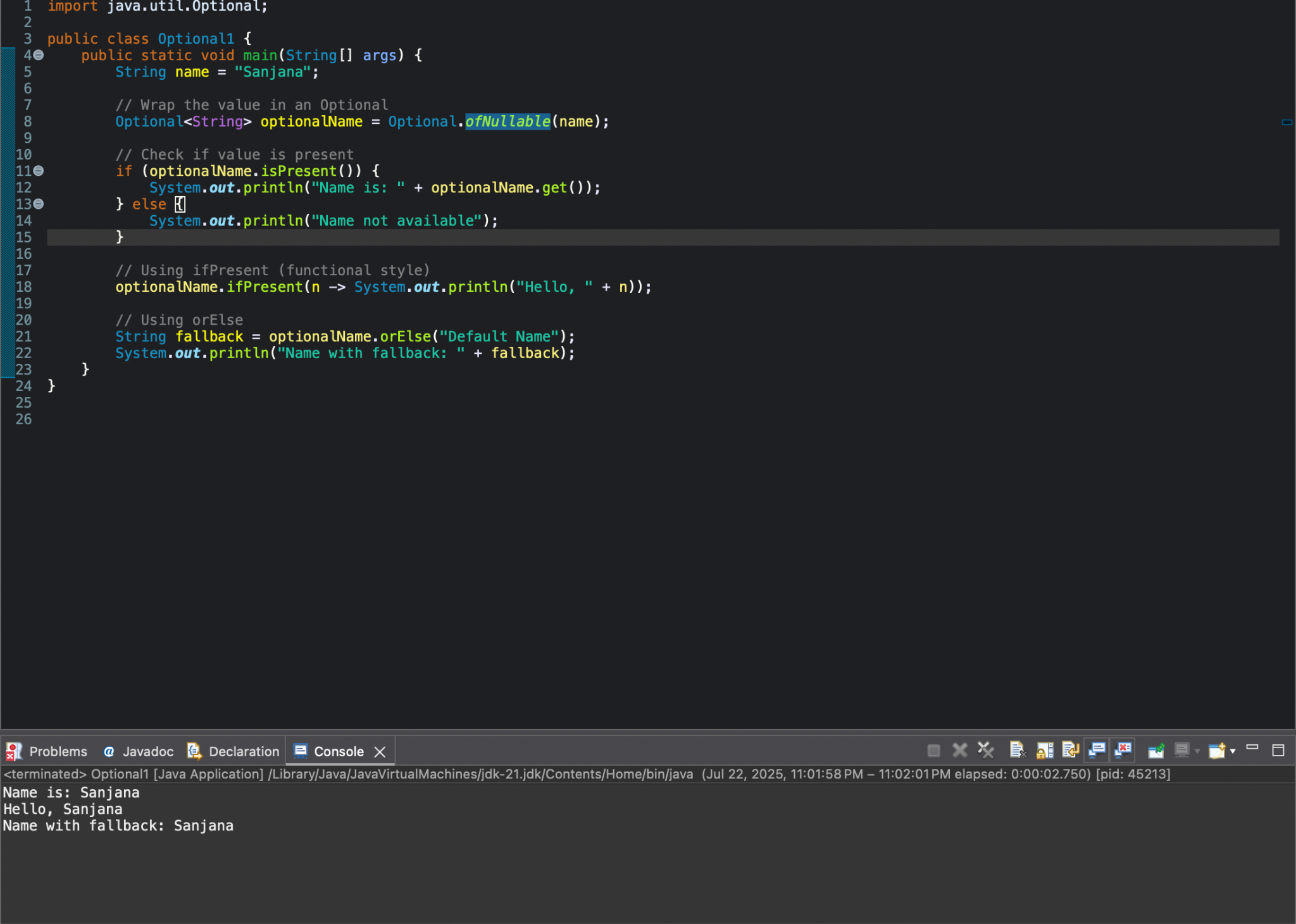
**3)**What is optional, and what is it best used for? **Explain with a coding example and share the output screenshot.**

What is Optional in Java?

**Optional is a container object introduced in Java 8 to represent the presence or absence of a value. It’s a better alternative to using null and helps avoid NullPointerException.**

Why Use Optional?

* **To explicitly handle cases where a value might be null**
* **To write cleaner, safer code using functional constructs**
* **To make the code more readable and intention-revealing**

****

4.)What is a functional interface? List some examples of predefined functional interfaces.

A **functional interface** is an interface with **only one abstract method**.  
 It’s the **target type for lambda expressions** and supports functional programming in Java.

Annotated with @FunctionalInterface for clarity and compile-time safety.

Why Use It?

* Enables use of **lambda expressions** and **method references**
* Supports **cleaner, concise code**
* Used in **Streams, callbacks, event handling**

Example:

Predicate<String> isEmpty = s -> s.isEmpty();

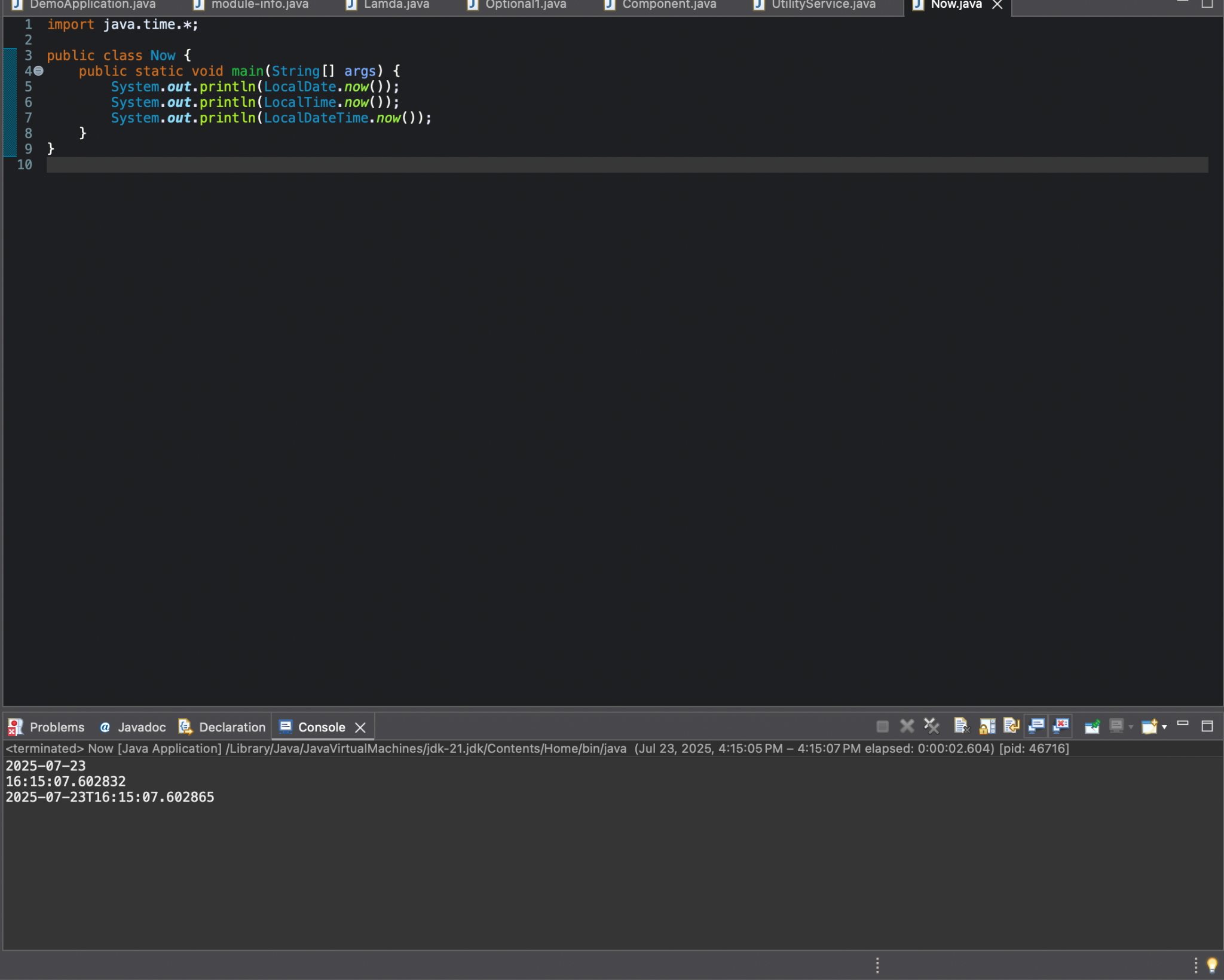
System.out.println(isEmpty.test("")); // true

5. How are functional interfaces and Lambda Expressions related?

Functional interfaces provide the target type for lambda expressions to implement in Java.

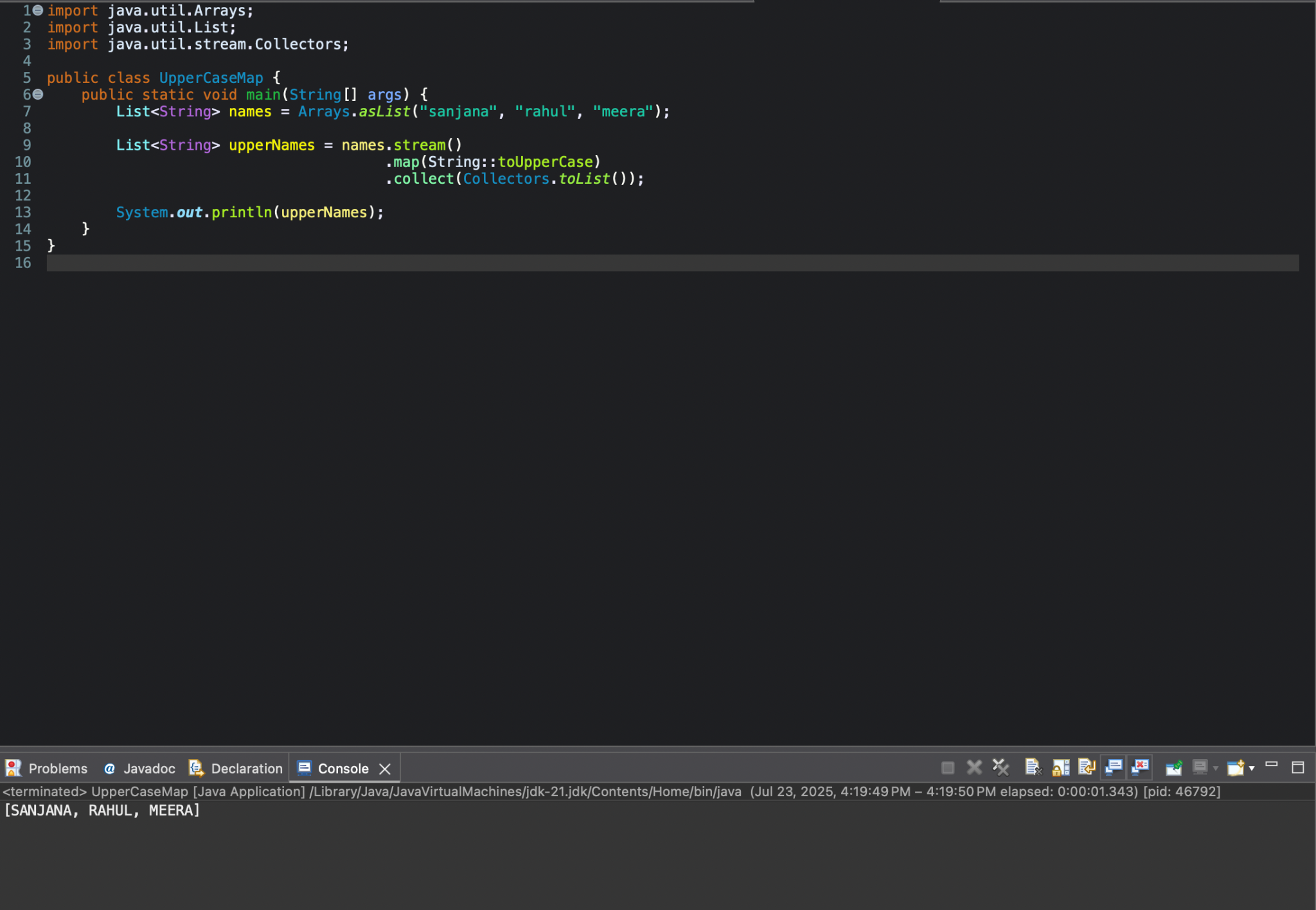
6.List some Java 8 Date and Time API’s. **How will you get the current date and time using Java 8 Date and Time API? Write the implementation and share the output screenshot.**

**Java 8 Date-Time APIs: LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Instant**

****

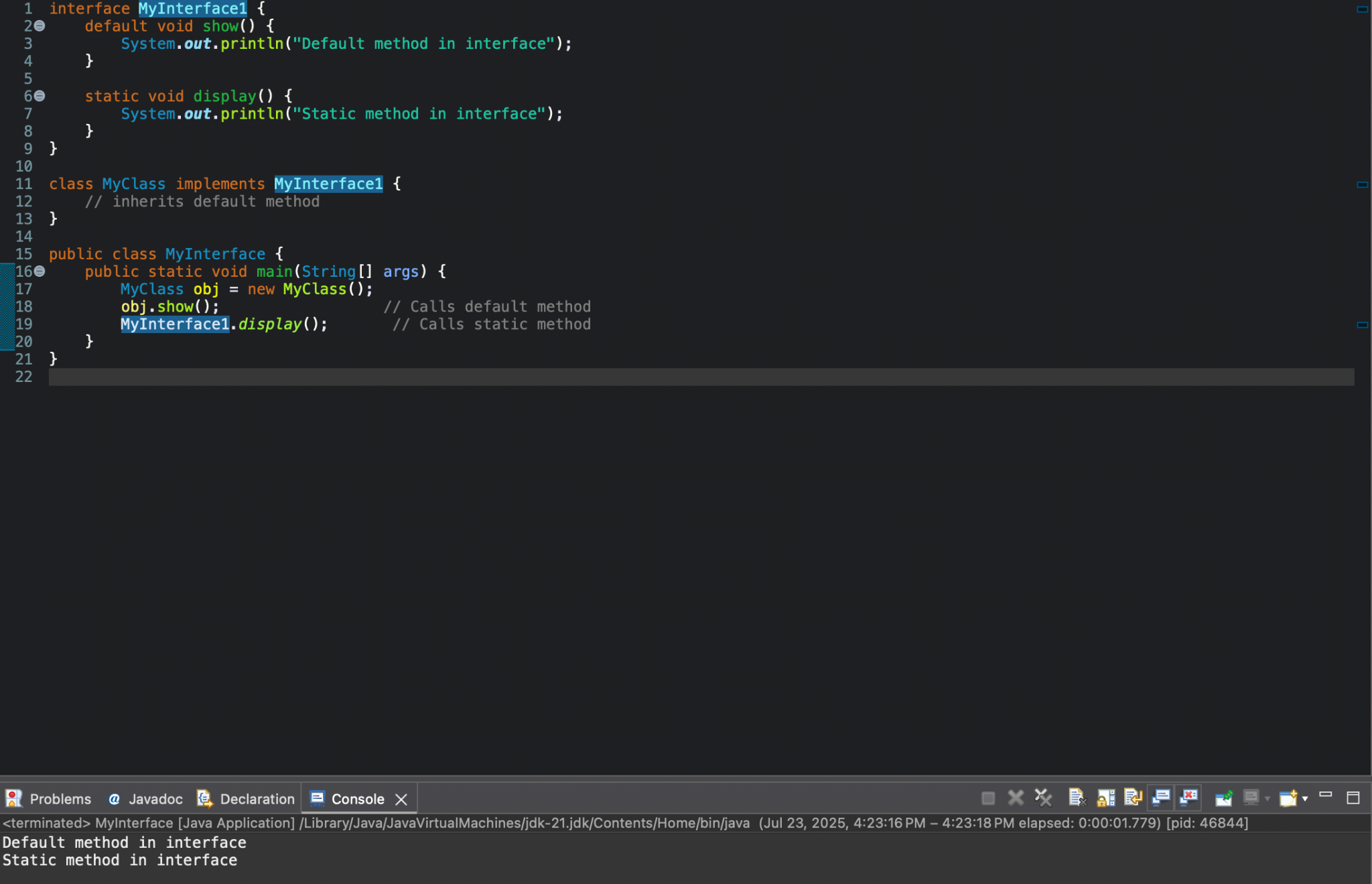
**7.How to use map to convert objects into Uppercase in Java 8? Write the implementation and share the output screenshot.**

1. **Java 8 Stream.map() is used to transform each element in a collection.**
2. **String::toUpperCase is passed to convert each string to uppercase.**
3. **The result is collected back into a list using collect(Collectors.toList()).**

****

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

* Java 8 introduced default and static methods in interfaces to add method implementations without breaking existing classes.
* Default methods allow adding new functionality to interfaces while maintaining backward compatibility.
* Static methods can be called using the interface name and are not inherited by implementing classes



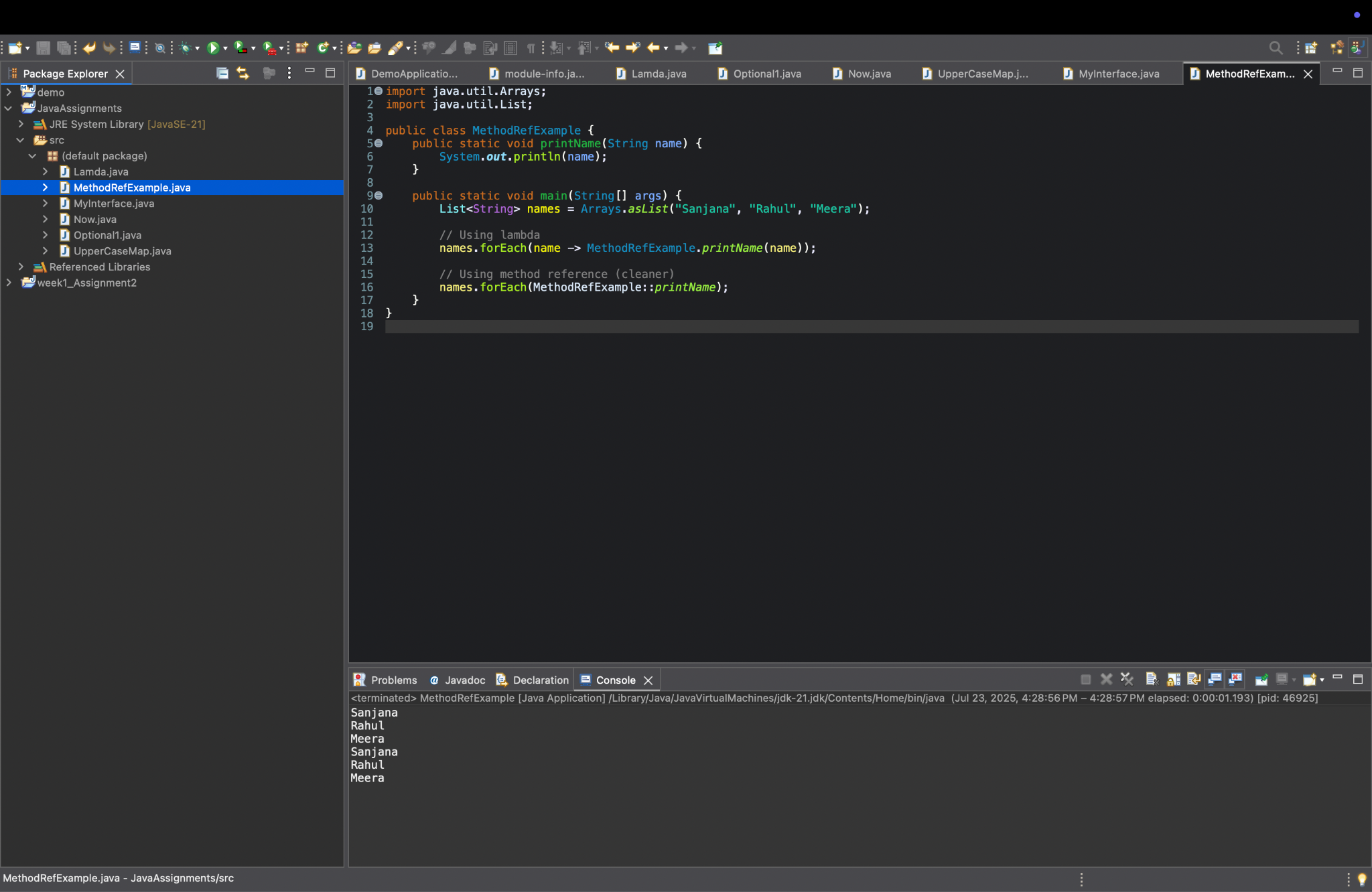
**9.** Discuss the significance of the Stream API introduced in Java 8 for data processing. How does it improve application performance and developer productivity?

* **Stream API** enables functional-style, declarative processing of collections (like filtering, mapping, reducing).
* It improves **developer productivity** by simplifying code with powerful operations like map(), filter(), and collect().
* It boosts **performance** using **lazy evaluation** and **parallel streams**, allowing efficient and scalable data processing.

10. 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.**

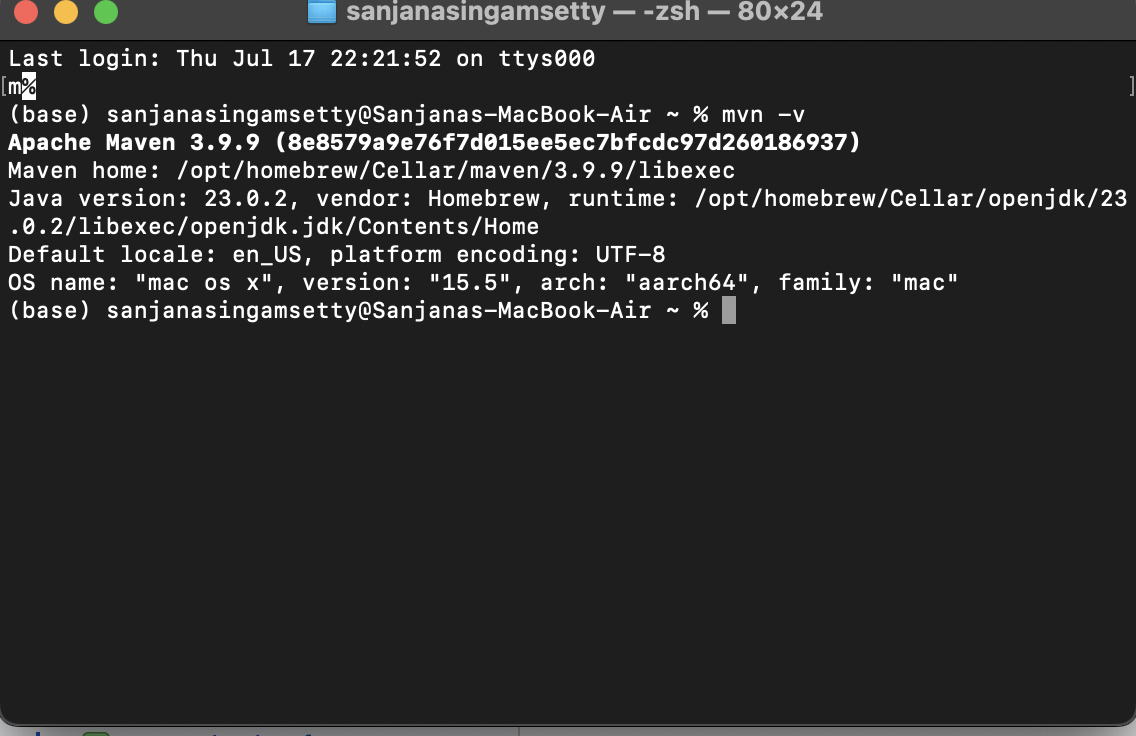
✅ 10. Method References in Java 8

**Crisp Explanation:**

* **Method references are a shorthand for lambda expressions that call existing methods.**
* **They improve readability and reduce boilerplate when the lambda just calls a method.**
* **Syntax: ClassName::methodName or object::methodName  
  **

SPRINGBOOT

1. **Install maven 3.6 or above. Execute mvn -v in the local terminal/command prompt and share the screenshot**

****

1. **What is the difference between maven central repository and local repository?**

**Difference between Maven Central and Local Repository (Crisp Points):**

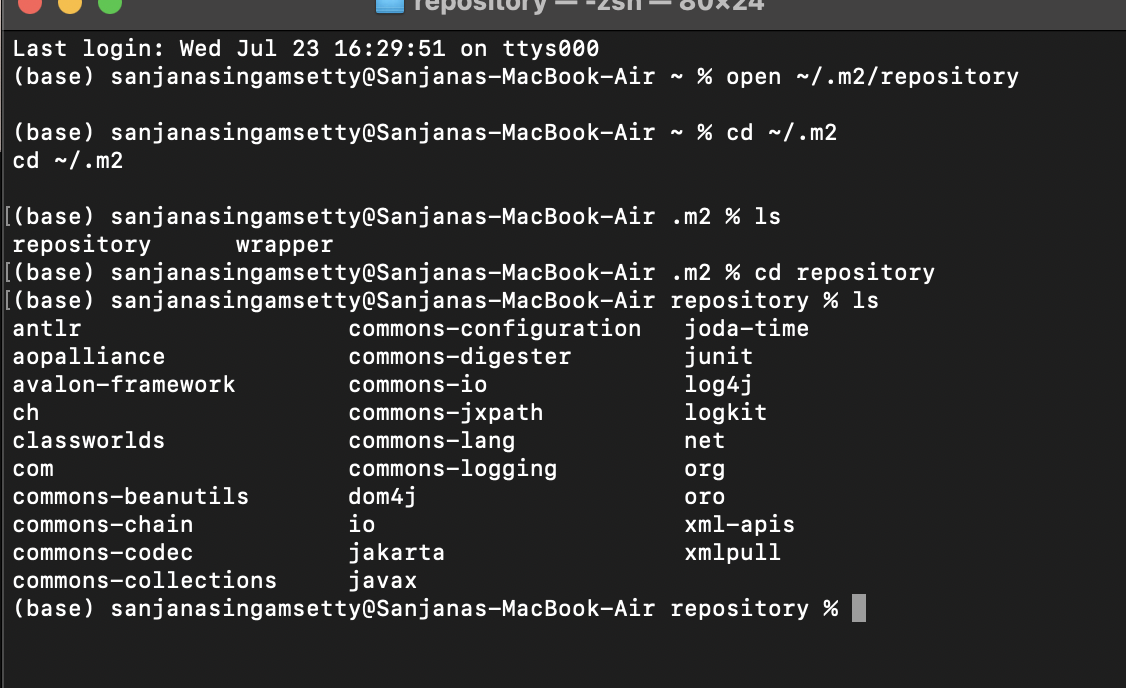
* **Local Repository:**
  1. **Located on your machine (~/.m2/repository)**
  2. **Stores downloaded and built dependencies**
  3. **Fast access, no internet needed once downloaded**
* **Maven Central Repository:**
  1. **Public online repository (https://repo.maven.apache.org)**
  2. **Contains thousands of open-source libraries**
  3. **Requires internet to fetch dependencies first time**

1. **Maven commands**
   1. **To build the maven project**
   2. **To run the maven tests**

**Maven Commands:**

* **To build the project:  
   mvn clean install**
* **To run tests:  
   mvn test**

1. **Please locate the maven settings.xml file and local maven repository in your machine and share the screenshot**

****

5)**The basic principle behind Dependency Injection(DI) is that the objects define their dependencies .What are the different ways in which an object can define its dependency ?**

Ways an object can define its dependencies (DI):

1. **Constructor Injection** → Dependencies are passed via constructor.
2. **Setter Injection** → Dependencies are set via public setters.
3. **Field Injection** → Dependencies are injected directly into fields (e.g., using @Autowired).

**6)What is the difference between the @Autowired and @Inject annotation?**

@Autowired

* **Spring-specific**
* required = true by default
* Supports @Qualifier, @Lazy, etc.

@Inject

* **Java standard (JSR-330)**
* No required attribute
* Needs @Named for qualifiers

7)**Explain the use of @Respository, @Component, @Service and @Controller**

@Component

Generic Spring bean.  
 **Used for:** Any general-purpose class.

@Component

public class Helper {}

@Repository

Marks **DAO classes**, handles DB exceptions.

@Repository

public class UserRepo {}

@Service

Marks **business logic** classes.

@Service

public class UserService {}

@Controller

Handles **web requests (MVC)**.

@Controller

public class UserController {}

**8.)Fix the code and explain why?**

**The following code tries to inject a property from application.properties, but the appName field is always null. Identify and fix the issue.**

**@Component**

**public class AppNamePrinter {**

**@Value("app.name")**

**private String appName;**

**public void printAppName() {  
 System.out.println("Application Name: " + appName);  
 }  
 }**

Problem:

**@Value("app.name")**

**private String appName;**

**This injects the literal string "app.name", not the property value.**

Fix:

**Use ${...} to reference properties:**

**@Value("${app.name}")**

**private String appName;**

Reason:

**"${app.name}" tells Spring to fetch the value from application.properties. Without ${}, it treats it as a plain string.**

Example in application.properties:

**app.name=My Spring App**

**9. What does the @SpringBootApplication annotation do?**

**@SpringBootApplication is a convenience annotation that combines:**

* **@Configuration – Marks the class as a source of Spring bean definitions**
* **@EnableAutoConfiguration – Enables auto-configuration based on dependencies**
* **@ComponentScan – Scans for components in the package**

**It bootstraps the Spring Boot application with minimal setup.**

**10. What is the maven command to start the SpringBootApplication?**

**mvn spring-boot:run**

**This compiles the project and runs the main() method annotated with @SpringBootApplication.**

**11.Implement EmployeeCRUD using Spring and JDBC with the below Employee class. In the branch feature-spring, create a folder Employee-Spring. Push the solution to the branch and share the link.**

**class Employee{**

**private int id;**

**private String name;**

**private String department;**

**}**

[**https://github.com/sanjana-singamsetty/rg-assignments/tree/feature-spring/week2**](https://github.com/sanjana-singamsetty/rg-assignments/tree/feature-spring/week2)

**12. mplement EmployeeCRUD using SpringBoot and Spring Data JPA with the below Employee class. In the branch feature-spring, create a folder Employee-SpringBoot-JPA. Push the solution to the branch and share the link.**

**class Employee{**

**private int id;**

**private String name;**

**private String department;**

**}**

[**https://github.com/sanjana-singamsetty/rg-assignments/tree/feature-spring/week2**](https://github.com/sanjana-singamsetty/rg-assignments/tree/feature-spring/week2)

**13. Follow the demo in the pre-work link** [**https://www.youtube.com/watch?v=hr2XTbKSdAQ&t=18s**](https://www.youtube.com/watch?v=hr2XTbKSdAQ&t=18s) **and create a Spring Batch application that processes customer data. In the branch feature-spring, create a folder Customer-SpringBatch. Push the solution to the branch and share the link**

[**https://github.com/sanjana-singamsetty/rg-assignments/tree/feature-spring/week2**](https://github.com/sanjana-singamsetty/rg-assignments/tree/feature-spring/week2)