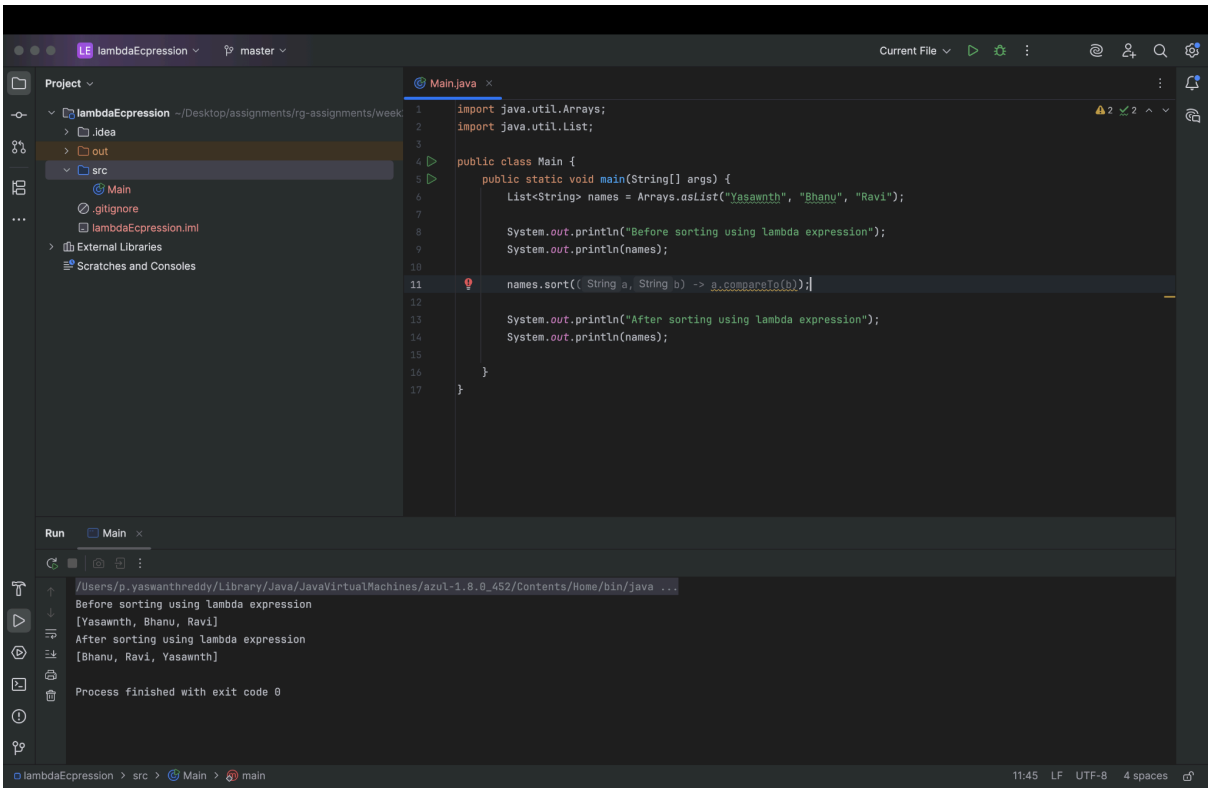
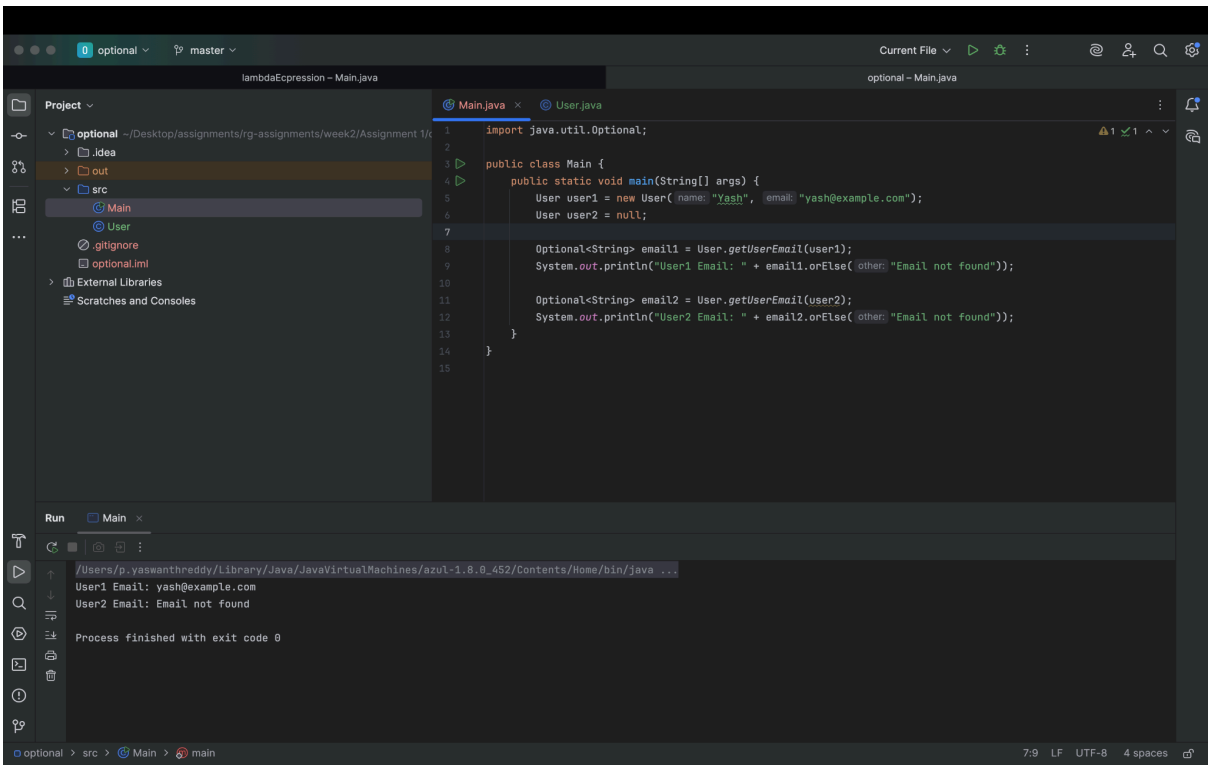
1. Features of Java 8 are as follows:
    a. Default & Static methods in interface
    b. Functional Interfaces
    c. Lambda Expression
    d. Method & Constructor reference
    e. forEach() method
    f. Stream API
    g. Functional API (Predicate, Consumer, Supplier)
    h. Date and Time API (joda time)
    i. Optional Class
    j. Nashorn JavaScript Engine

2. A Lambda Expression is a concise way to represent an anonymous function that can be passed as an argument to a method or stored in a variable. It was introduced in Java 8 to enable functional programming in Java.
   We use Lambda Expression to reduce boilerplate code, to improve readability, to enable functional-style programming and to make code more concise.



Github link :
https://github.com/PathireddyYaswanthReddy/rg-assignments/blob/master/week2/Assignment%201/lambdaEcpression/src/Main.java

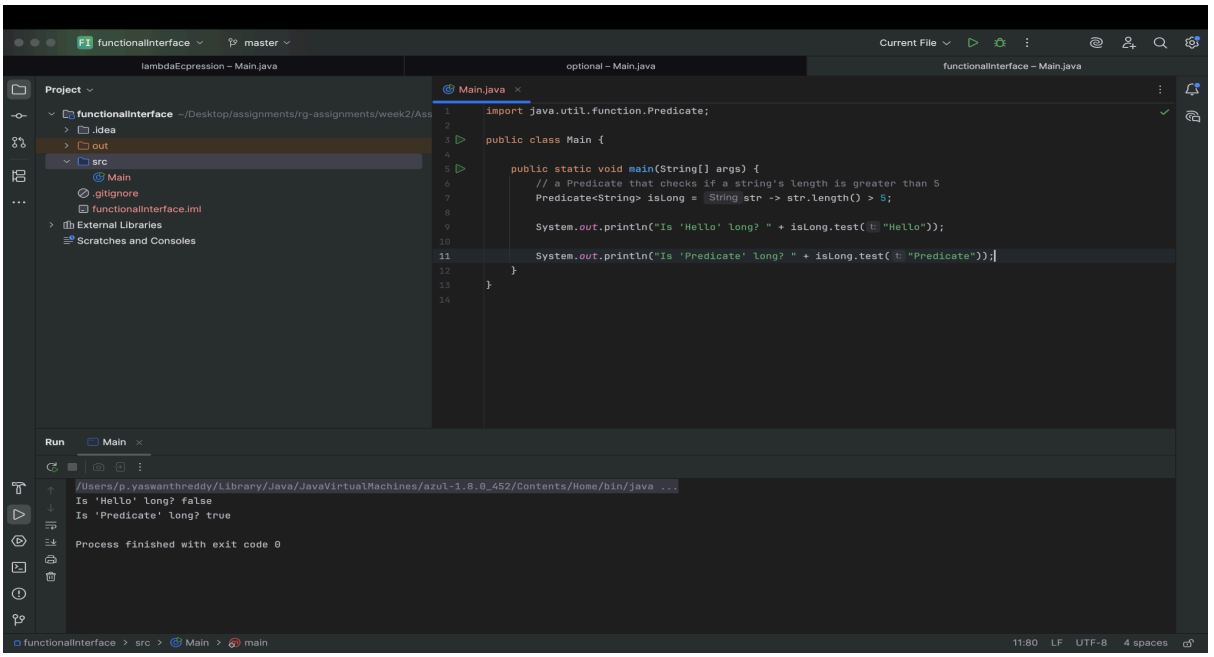
3. Optional<T> is a container object introduced in Java 8 to represent a value that may or may not be present. It helps to avoid **NullPointerException** and makes our code more readable and safer.
   We use Optional to avoid null checks, make it clear that a value might be absent and to provide a clean, functional-style API to handle missing values.



Github link:
https://github.com/PathireddyYaswanthReddy/rg-assignments/blob/master/week2/Assignment%201/optional/src/Main.java

4. A functional interface is an interface that contains only one abstract method. It can have default or static methods, but only one abstract method.
Some of the predefined functional interfaces in Java 8 are

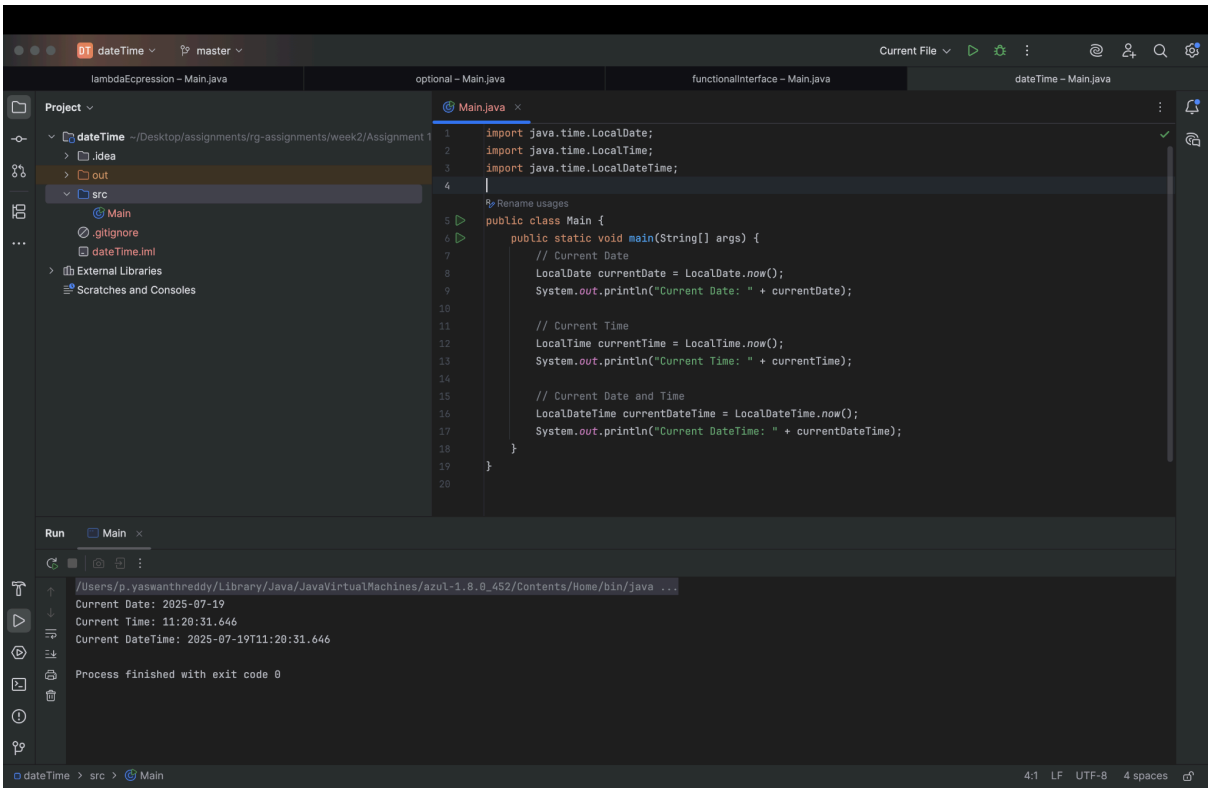
a. Function<T, R>
b. Predicate<T>
c. Consumer<T>
d. Supplier<T>



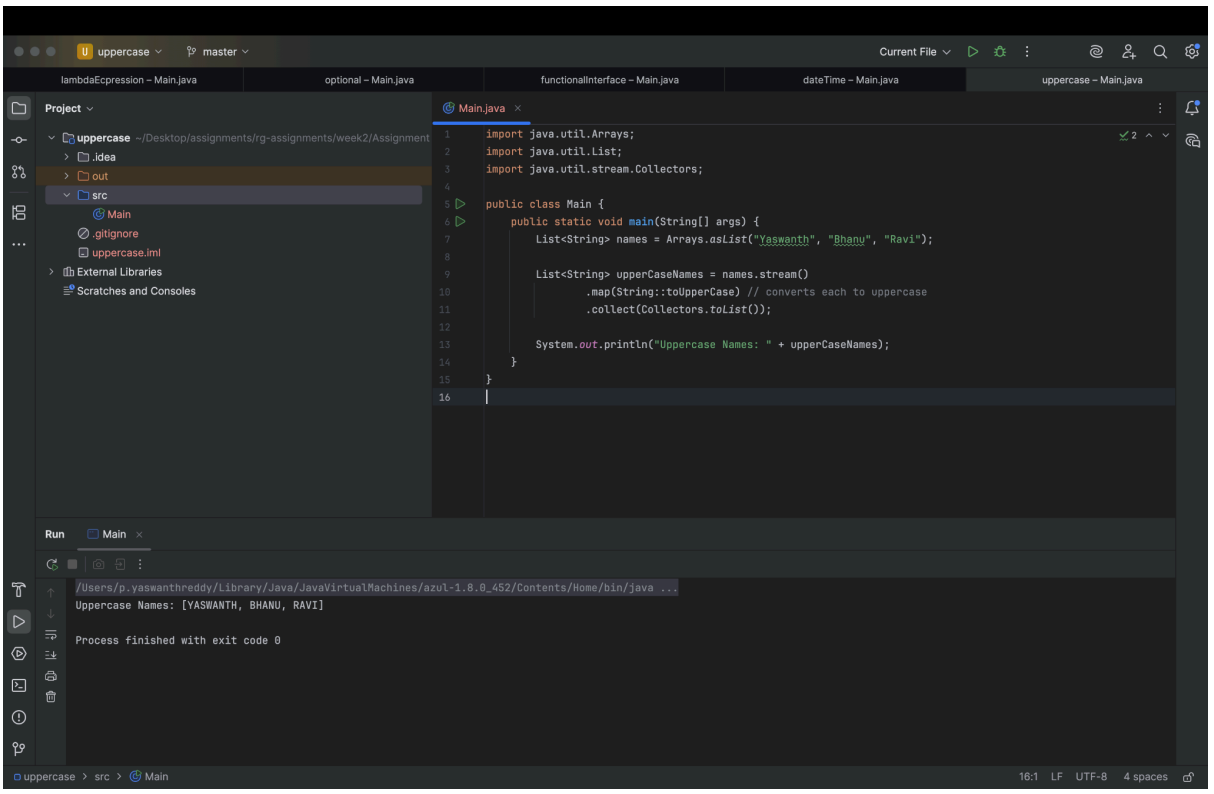
Github link:
https://github.com/PathireddyYaswanthReddy/rg-assignments/blob/master/week2/Assignment%201/functionalInterface/src/Main.java

5. Functional interfaces and lambda expressions are closely related (tightly coupled). Functional interfaces, which have a single abstract method, serve as the target for lambda expressions. Lambda expressions provide a concise way to implement the abstract method of a functional interface, essentially treating behavior as code that can be passed around and used as a value.

6. Some of the Java 8 Date and Time APIs (java.time package) are as follows:
    a. LocalDate - Date only
    b. LocalTime - Time only
    c. LocalDateTime - Date and Time
    d. ZonedDateTime - Date / Time with time zone info
    e. Instant - Timestamp of machine
    f. Period / Duration - Difference between dates/times



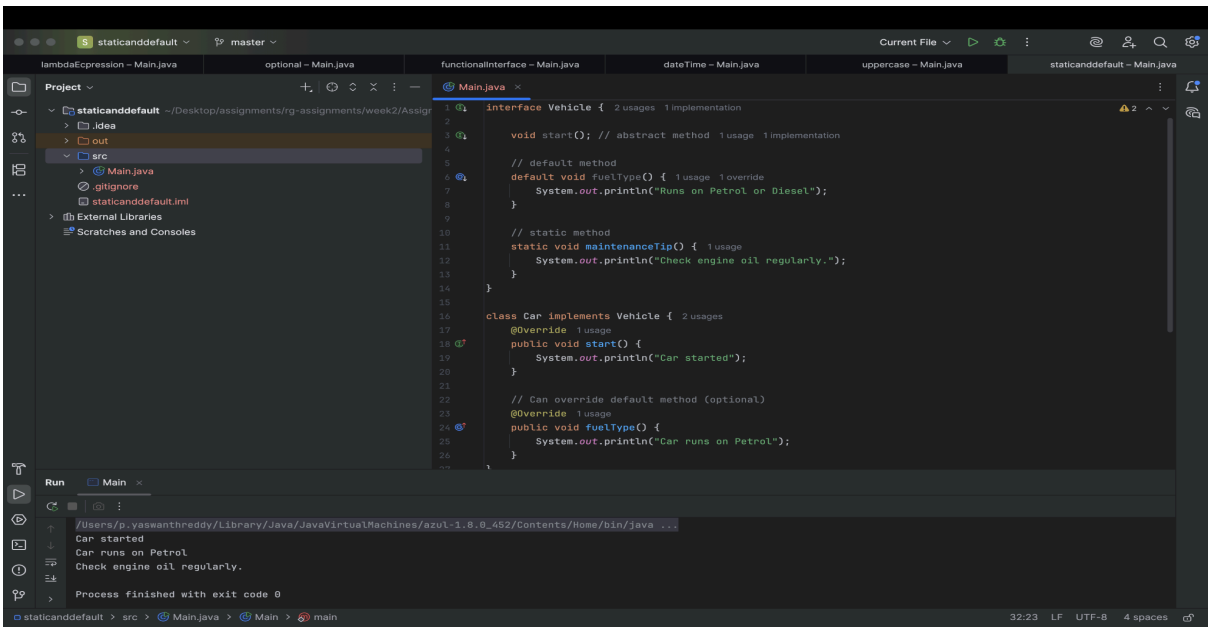
Github link:
https://github.com/PathireddyYaswanthReddy/rg-assignments/blob/master/week2/Assignment%201/dateTime/src/Main.java

7. 

Github link:
https://github.com/PathireddyYaswanthReddy/rg-assignments/blob/master/week2/Assignment%201/uppercase/src/Main.java

8. Before Java 8, Interfaces could only contain abstract methods. If an interface was changed (e.g., adding a new method), all implementing classes would break.
   To fix this and maintain backward compatibility, Java 8 introduced default methods and static methods.
   default Method:
   a. Allows you to provide implementation inside an interface.
   b. Implementing classes can override it.
   static Method:
   a. Belongs to the interface itself, not to implementing classes.
   b. Called using the interface name.

Github link:
https://github.com/PathireddyYaswanthReddy/rg-assignments/blob/master/week2/Assignment%201/staticanddefault/src/Main.java



9. The Stream API, introduced in Java 8, is a powerful abstraction for processing sequences of elements (like collections) in a functional style. It improves performance by enabling parallel streams, which automatically divide workloads across CPU cores for faster processing. Its lazy evaluation mechanism ensures that operations are only performed when necessary, reducing unnecessary computations. Additionally, pipelining allows multiple intermediate operations like filter() and map() to be combined efficiently, minimizing overhead and enhancing execution speed. The Stream API improves productivity by reducing boilerplate code and promoting the use of lambda expressions and functional programming, making the code more concise.

10. Method references are a shorthand notation of lambda expressions to call a method directly by its name.

Type	Syntax
Reference to a static method	ClassName::staticMethod
Reference to an instance method	object::instanceMethod
Reference to a constructor	ClassName::new

Github link:
https://github.com/PathireddyYaswanthReddy/rg-assignments/blob/master/week2/Assignment%201/methodReference/src/MethodReferenceExample.java

methodReference | master | Current File

lambdaEcpression – Main.java | optional – Main.java | functionalInterface – Main.java | dateTime – Main.java | uppercase – Main.java | staticanddefault – Main.java | methodReference – Main.java

Project

- methodReference ~/Desktop/assignments/rg-assignments/week2/Assi
  - .idea
  - out
  - src
    - MethodReferenceExample
  - .gitignore
  - methodReference.iml
- External Libraries
- Scratches and Consoles

MethodReferenceExample.java

```java
import java.util.Arrays;
import java.util.List;

public class MethodReferenceExample {

    public static boolean startsWithA(String name) {  2 usages
        return name.startsWith("A");
    }

    public static void main(String[] args) {
        List<String> names = Arrays.asList("Anu", "Ravi", "Arjun", "Yaswanth");

        // Using lambda expression
        names.stream()
                .filter( String name -> startsWithA(name))
                .forEach( String name -> System.out.println(name));

        // Using method reference
        names.stream()
                .filter(MethodReferenceExample::startsWithA)
                .forEach(System.out::println);
    }
```

Run | MethodReferenceExample

```
/Users/p.yaswanthreddy/Library/Java/JavaVirtualMachines/azul-1.8.0_452/Contents/Home/bin/java ...
Anu
Arjun
Anu
Arjun

Process finished with exit code 0
```

methodReference > src > MethodReferenceExample > main | 11:77 | LF | UTF-8 | 4 spaces