

Week 2: Assignment 1 - Java 8 Features

1. List the features of Java 8

Ans.)

1. Lambda Expressions – Introduces a new way to write anonymous methods using a clean and concise syntax. It helps to pass behavior as parameters to methods.
2. Functional Interfaces – An interface with a single abstract method (SAM). It is used with lambda expressions to provide target types for the methods.
3. Stream API – Provides a powerful way to process collections of data (like filtering, sorting, mapping) in a functional style with less code.
4. Default Methods – Allows interfaces to have methods with a body (implementation) so existing code does not break when new methods are added.
5. Optional Class – A container object that may or may not hold a non-null value. It is used to avoid NullPointerExceptions and handle missing values properly.

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

Ans.) A lambda expression in Java 8 is a short way to write anonymous methods. It allows you to write cleaner and more concise code, especially for functional interfaces.

We use lambda expressions to:

- Reduce boilerplate code (no need for separate class or method).
- Pass behavior as a parameter (functional programming).
- Make code more readable and maintainable.

Syntax of Lambda Expression: (parameter1, parameter2) -> { body of method }

```
import java.util.*;
```

```
public class LambdaExample {
```

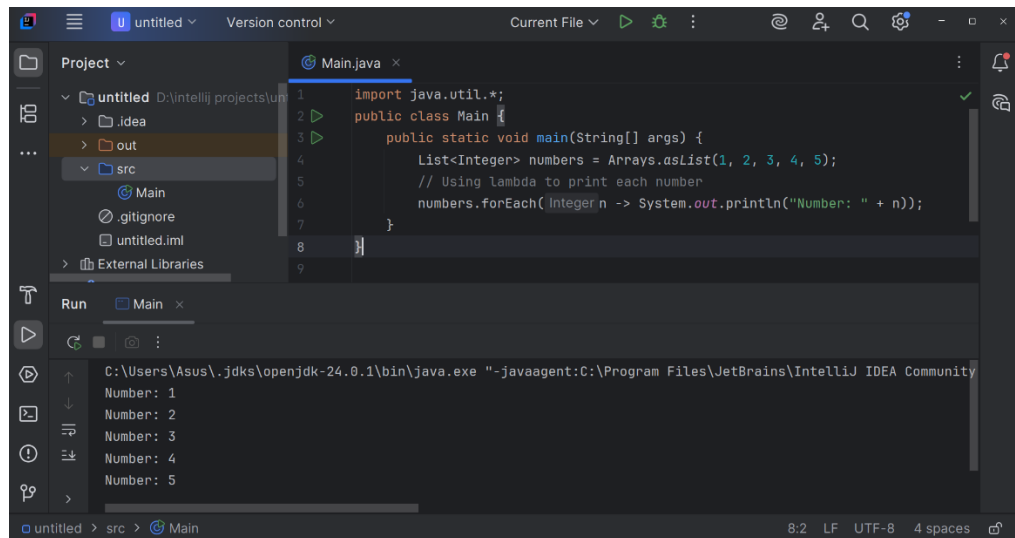
```
    public static void main(String[] args) {
```

```
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
```

```
        numbers.forEach(n -> System.out.println("Number: " + n));
```

```
    }
```

```
}
```



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

Ans.) Optional is a class introduced in Java 8. It acts as a container for a value that might be present or absent (null).

It is mainly used to avoid NullPointerException and to write safer, cleaner code when dealing with nullable values.

It is best used for:

- To handle null values without explicit null checks.
- To return values safely from methods that may not have a result.

```
import java.util.Optional;
```

```
public class OptionalExample {
```

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

```
        String name = "Harshit";
```

```
        Optional<String> optionalName = Optional.ofNullable(name);
```

```
        if (optionalName.isPresent()) {
```

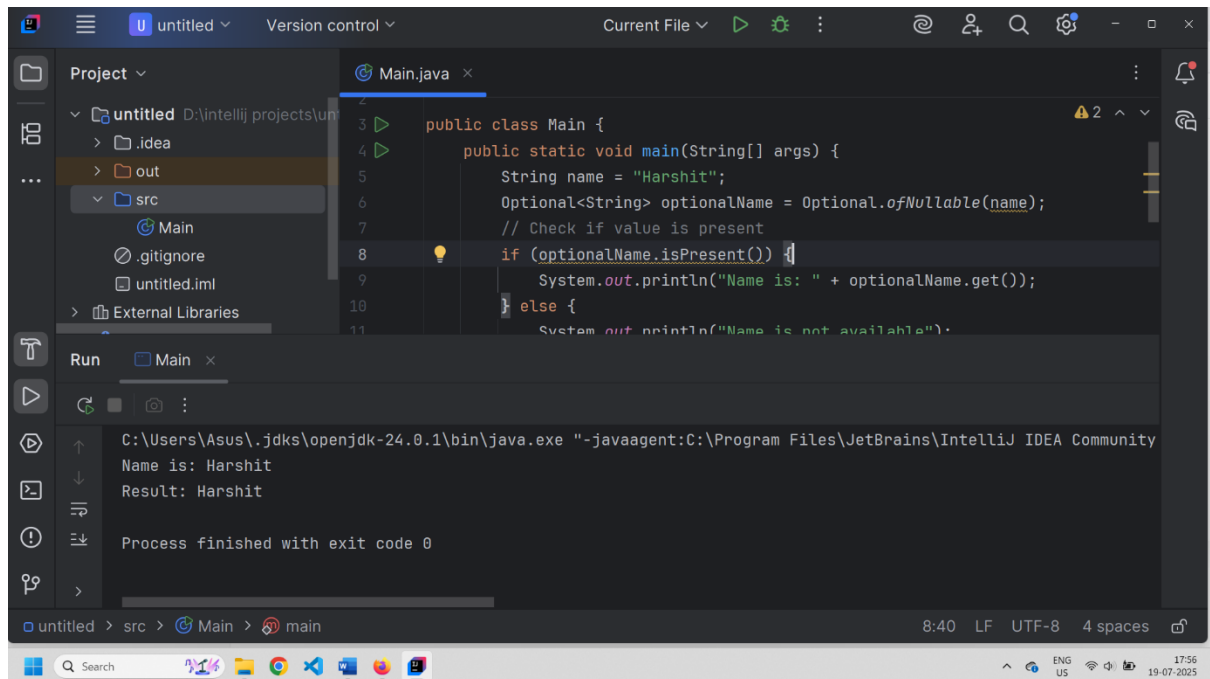
```
            System.out.println("Name is: " + optionalName.get());
```

```
        } else {
```

```
            System.out.println("Name is not available");
```

```
        }
```

```
}  
  
}
```



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

Ans.) A functional interface in Java is an interface that has only one abstract method. It can have multiple default or static methods, but only one abstract method.

Examples of Predefined Functional Interfaces in Java 8:

1. Runnable – Represents a task that can be executed (no input, no output).
2. Callable – Represents a task that returns a result.
3. Comparator – Used to compare two objects.
4. Consumer – Accepts a single argument and returns no result.
5. Supplier – Provides a result but takes no input.
6. Function – Accepts one argument and produces a result.
7. Predicate – Accepts one argument and returns a boolean (true/false).

5. How are functional interfaces and Lambda Expressions related?

Ans.) A functional interface is an interface with only one abstract method, and a lambda expression essentially provides an implementation for that single method.

Lambda expressions simplify the process of implementing functional interfaces, making code more concise and readable.

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.

Ans.)

- LocalDate – Represents a date (year, month, day) without time.
- LocalTime – Represents only time (hour, minute, second, nanosecond).
- LocalDateTime – Represents both date and time.
- ZonedDateTime – Represents date and time with timezone.
- Instant – Represents a timestamp in UTC.

```
import java.time.LocalDate;

import java.time.LocalTime;

import java.time.LocalDateTime;

public class DateTimeExample {

    public static void main(String[] args) {

        LocalDate currentDate = LocalDate.now();

        System.out.println("Current Date: " + currentDate);

        LocalTime currentTime = LocalTime.now();

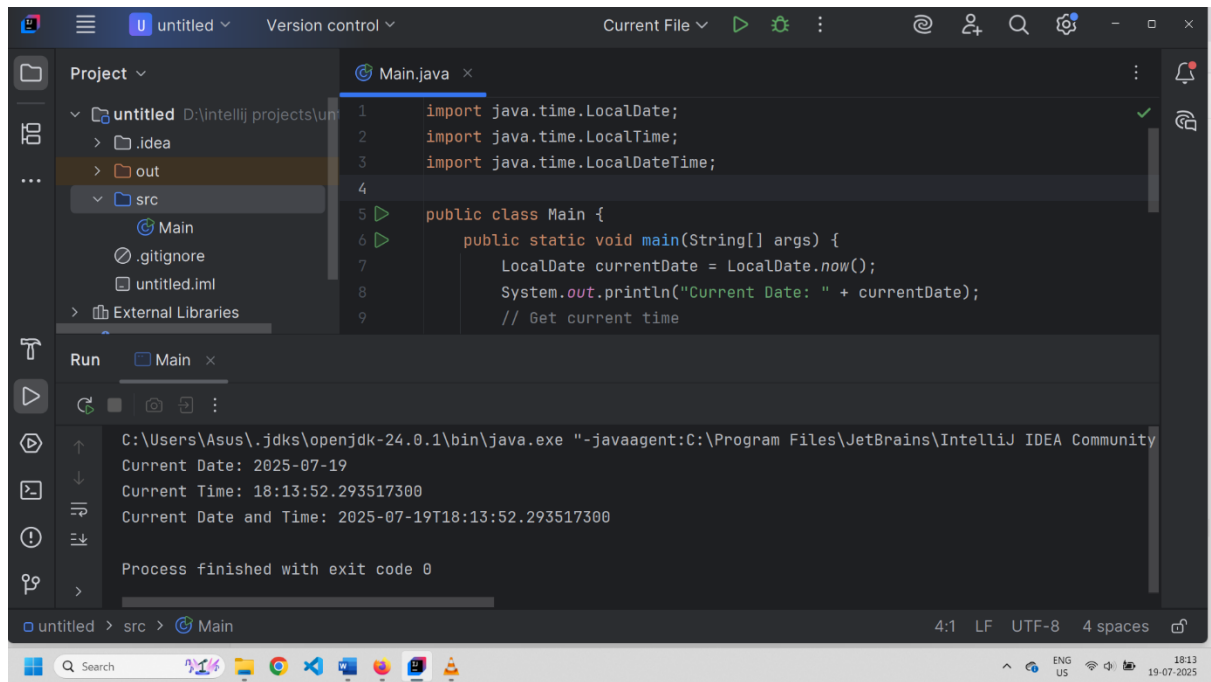
        System.out.println("Current Time: " + currentTime);

        LocalDateTime currentDateTime = LocalDateTime.now();

        System.out.println("Current Date and Time: " + currentDateTime);

    }

}
```



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

Ans.) The map() method in Java 8 Stream API is used to transform each element of a collection. To convert all strings to uppercase, we can use map(String::toUpperCase).

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

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

```
import java.util.stream.Collectors;
```

```
public class MapToUppercaseExample {
```

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

```
        List<String> names = Arrays.asList("harshit", "raj", "java", "stream");
```

```
        // Convert to uppercase using map()
```

```
        List<String> upperCaseNames = names.stream()
```

```
            .map(String::toUpperCase)
```

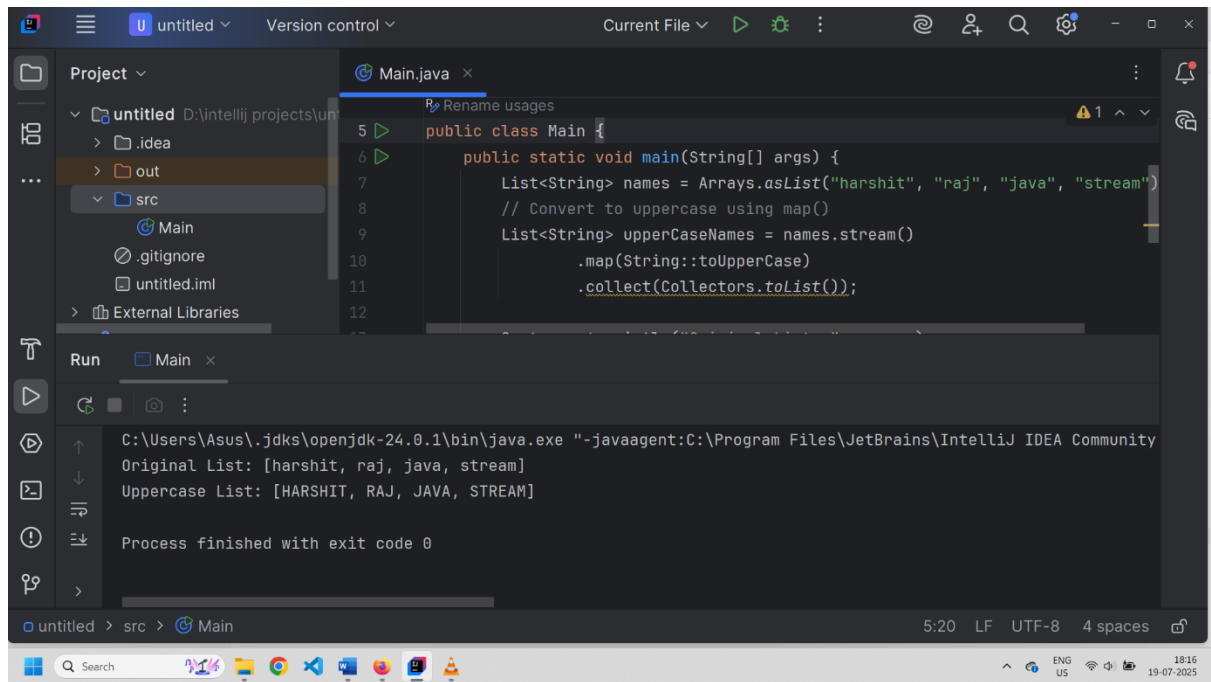
```
            .collect(Collectors.toList());
```

```
        System.out.println("Original List: " + names);
```

```
        System.out.println("Uppercase List: " + upperCaseNames);
```

```
    }
```

```
}
```



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

Ans.) In Java 8, interfaces were enhanced by introducing:

1. Default Methods

- Interfaces can now have methods with a body using the default keyword.
- It allows adding new functionality to interfaces without breaking existing classes that already implement the interface.

2. Static Methods

- Interfaces can also have static methods with a body.
- These methods belong to the interface itself and can be called without creating an object.

Before Java 8:

- Adding a new method to an interface forced all implementing classes to override it.
- Java 8 solved this with default methods, allowing backward compatibility and providing a way to extend interfaces easily.
- Static methods help keep utility methods inside interfaces rather than separate utility classes.

interface Calculator {

void add(int a, int b); // Abstract method

```

// Default method
default void show() {
    System.out.println("This is the default method in interface.");
}

// Static method
static void display() {
    System.out.println("This is the static method in interface.");
}

public class MyCalculator implements Calculator {

    public void add(int a, int b) {
        System.out.println("Addition: " + (a + b));
    }

    public static void main(String[] args) {
        MyCalculator calc = new MyCalculator();
        calc.add(5, 10); // Calls abstract method
        calc.show();     // Calls default method
        Calculator.display(); // Calls static method
    }
}

```

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

Ans.) The Stream API introduced in Java 8 provides a powerful and declarative way to process collections of data (like Lists, Sets, etc.). It helps developers work with data in a functional style (using map, filter, reduce, etc.).

How It Improves Application Performance:

1. **Efficient Processing:** Streams use lazy evaluation, meaning operations are only executed when needed.

2. **Parallel Streams:** Allows processing data in parallel using multiple CPU cores, improving performance for large datasets.
3. **Reduced Memory Usage:** Intermediate operations like map and filter don't create new collections; they process data on the fly.

How It Improves Developer Productivity:

1. **Less Boilerplate Code:** Stream API eliminates the need for loops and manual iteration.
2. **Readable and Clean Code:** Uses method chaining and functional operations for clarity.
3. **Powerful Functional Operations:** Provides ready-to-use operations like filter, map, sorted, collect, etc.

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.

Ans.) Method references in Java 8 are a shorthand way of writing lambda expressions that simply call an existing method.

They make the code shorter and more readable when the lambda expression is just forwarding its parameters to a method.

Lambda expressions define the behavior inline. Method references reuse existing methods instead of writing a lambda. Both achieve the same goal but method references are cleaner when you don't need extra logic.

Example Where Method Reference is Better than Lambda

```
import java.util.Arrays;

import java.util.List;

public class MethodReferenceExample {

    public static void main(String[] args) {

        List<String> names = Arrays.asList("Harshit", "Raj", "Java", "Stream");

        // Using lambda expression

        System.out.println("Using Lambda:");

        names.forEach(name -> System.out.println(name));
```



```

// Using method reference

System.out.println("\nUsing Method Reference:");

names.forEach(System.out::println);

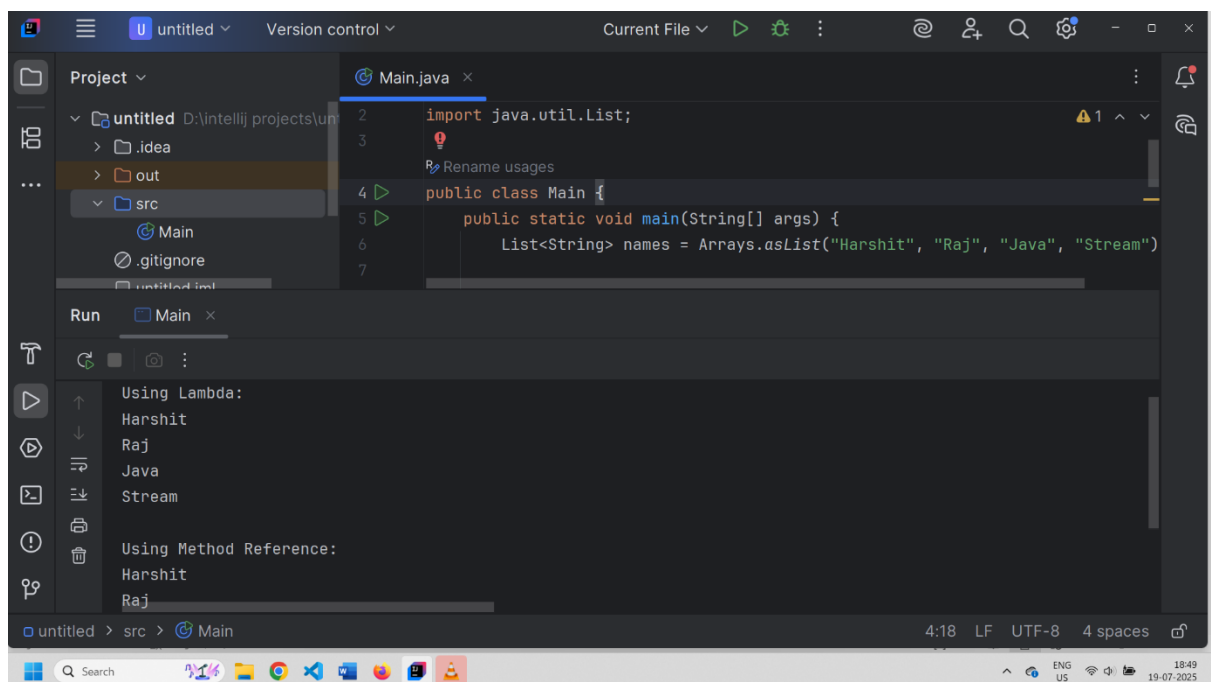
}

}

```

The reason Method Reference is Better Here because:

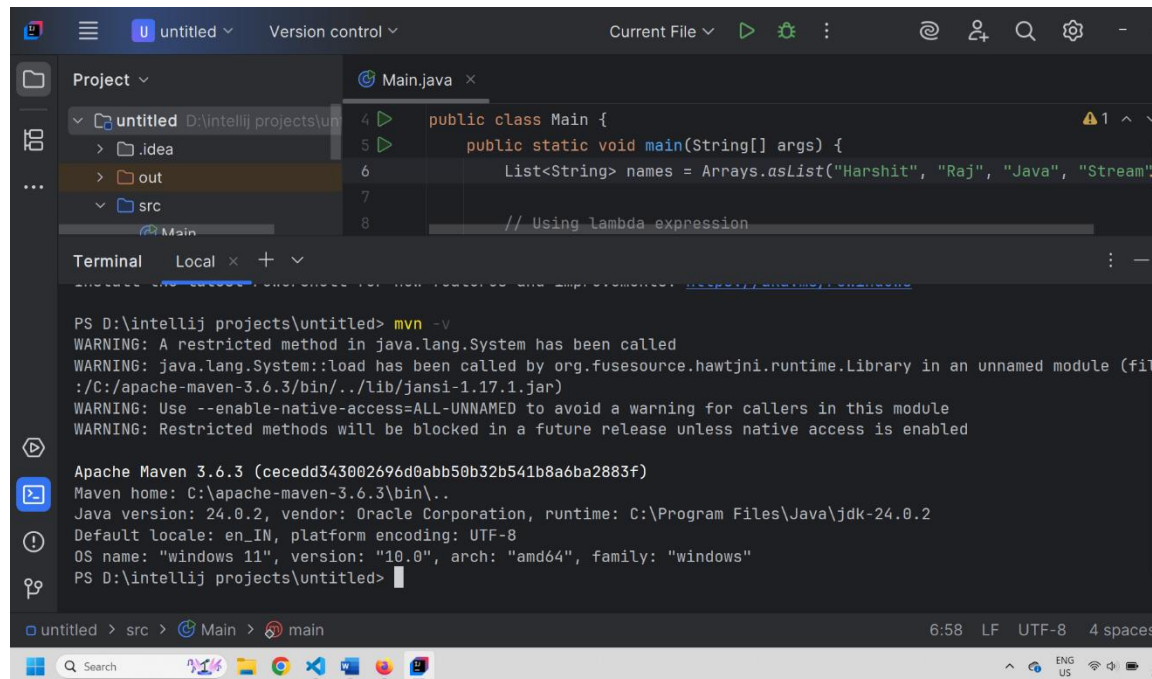
It avoids extra syntax (name ->) and directly points to System.out.println, making the code shorter and cleaner.



Week 2: Assignment 1 - Maven, Spring and Spring Boot

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

Ans.)



The screenshot shows an IDE interface with a project named 'untitled' and a file 'Main.java'. The terminal window displays the output of the command `mvn -v`. The output shows the Apache Maven version (3.6.3), the Maven home directory, the Java version (24.0.2), the vendor (Oracle Corporation), the runtime (C:\Program Files\Java\jdk-24.0.2), the default locale (en_IN), the platform encoding (UTF-8), the OS name (windows 11), the version (10.0), the arch (amd64), and the family (windows).

```
PS D:\intellij projects\untitled> mvn -v
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::load has been called by org.fusesource.hawtjni.runtime.Library in an unnamed module (file
:/C:/apache-maven-3.6.3/bin/../lib/jansi-1.17.1.jar)
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled

Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\apache-maven-3.6.3\bin\..
Java version: 24.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-24.0.2
Default locale: en_IN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS D:\intellij projects\untitled>
```

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

Ans.) The Maven Central Repository is a large online library of all the commonly used Java libraries, plugins, and dependencies. It is hosted on the internet at <https://repo.maven.apache.org/maven2>. When we build a project, Maven automatically downloads any required dependencies from the Central Repository if they are not available on your computer.

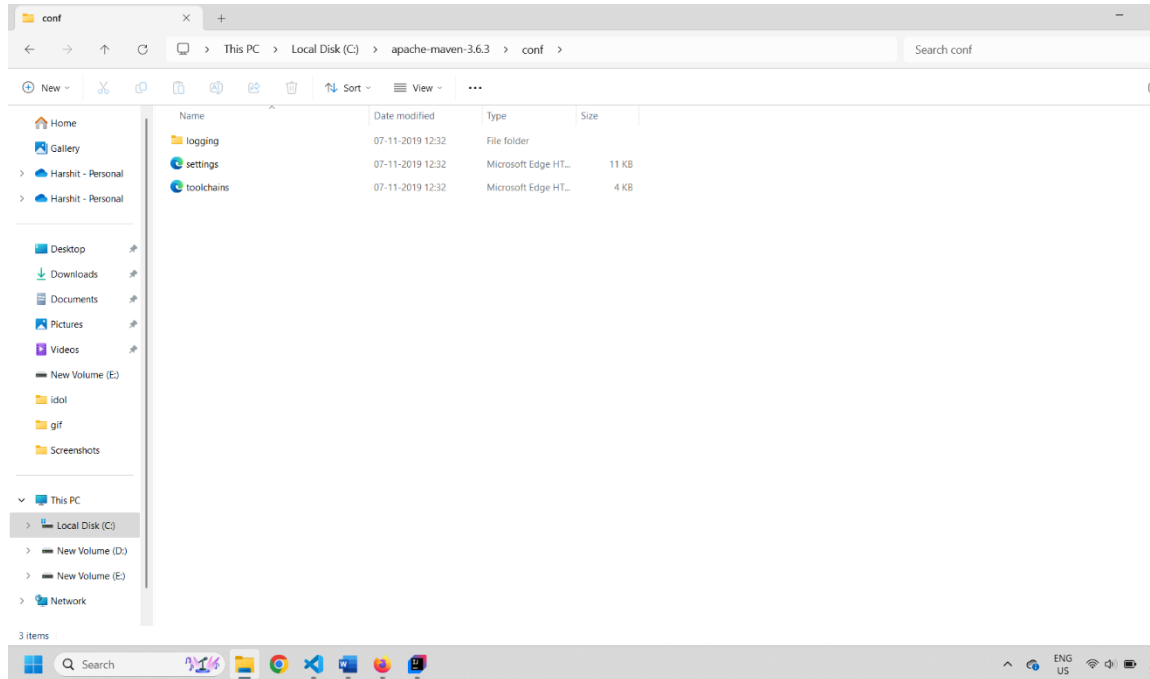
The Maven Local Repository is a folder on your own computer where Maven stores all the downloaded dependencies from the Central Repository. The default location of the local repository is `.m2/repository` in your user directory. Before downloading from the internet, Maven first checks this local repository. If the dependency is already there, it uses it directly without downloading again.

3. Maven commands

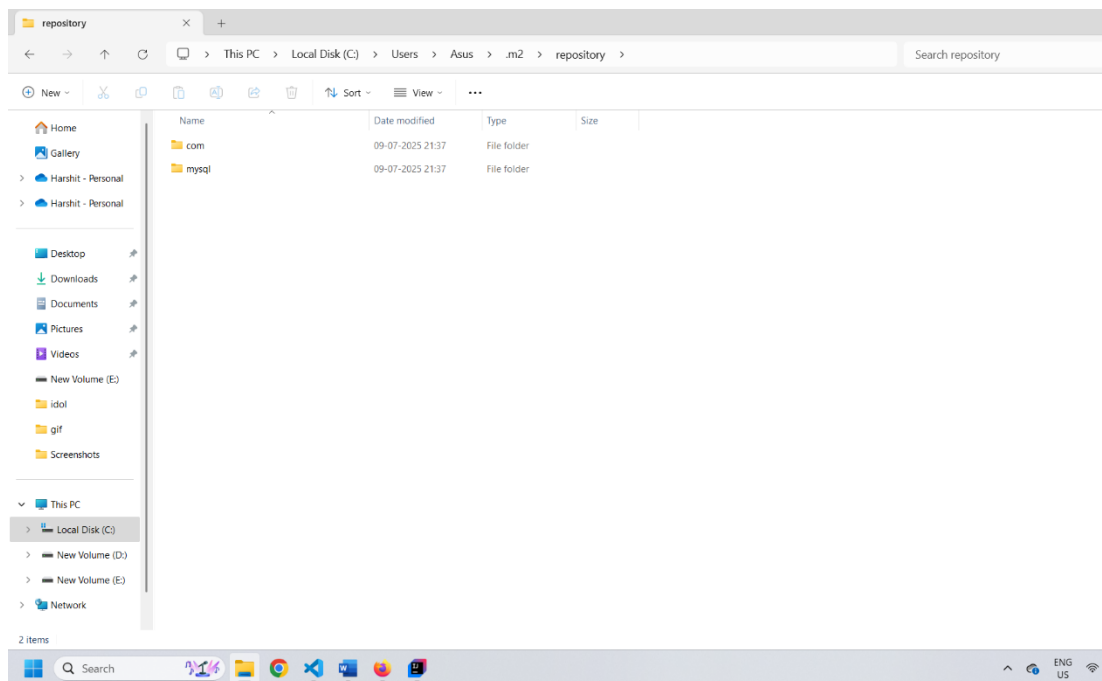
- a. To build the maven project - `mvn clean install`
- b. To run the maven tests - `mvn test`

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

Ans.) settings.xml file location



local maven repository location



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?

Ans.) Dependency Injection is a design pattern where the dependencies (objects a class needs) are provided from outside rather than the class creating them itself. This makes the code more flexible, testable, and easier to maintain.

There are three main types of Dependency Injection:

Constructor Injection - Dependencies are provided through the class constructor.

Example:

```
class Car {  
    private Engine engine;  
  
    Car(Engine engine) { // dependency injected via constructor  
        this.engine = engine;  
    }  
}
```

Setter Injection - Dependencies are provided using public setter methods after the object is created.

Example:

```
class Car {  
    private Engine engine;  
  
    public void setEngine(Engine engine) { // dependency injected via setter  
        this.engine = engine;  
    }  
}
```

Interface Injection - The dependency provides an injector method that will inject the dependency into any client passed to it.

Example:

```
interface EngineSetter {  
    void setEngine(Engine engine);  
}
```

```

class Car implements EngineSetter {
    private Engine engine;

    public void setEngine(Engine engine) {
        this.engine = engine;
    }
}

```

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

Ans.)

@Autowired	@Inject
Comes from Spring Framework (org.springframework.beans.factory.annotation.Autowired).	Comes from Javax (JSR-330) (javax.inject.Inject).
Spring-specific annotation with extra features.	Standard Java annotation (works in multiple frameworks).
By default, it requires the dependency (can set required=false for optional).	No required attribute; optional handled with @Nullable or Optional.
Supports @Qualifier for selecting specific beans.	Does not support @Qualifier directly; uses standard annotations.
Used only in Spring applications.	Can be used in any JSR-330 supported framework (Spring, Java EE, etc.).

7. Explain the use of @Respository, @Component, @Service and @Controller annotations with an example for each.

Ans.)

1. @Component

- **What it is:**
A generic stereotype annotation for any Spring-managed component.
- **Use case:**
Marks a class as a Spring bean (general-purpose).
- **Example:**

```
import org.springframework.stereotype.Component;
```

@Component

```
public class MyBean {
    public void display() {
        System.out.println("Hello from @Component bean!");
    }
}
```

2. @Repository - Specialized version of @Component for **DAO (Data Access Object)** classes.

- **Use case:** Marks a class that interacts with the database. It also provides automatic exception translation for persistence-related exceptions.
- **Example:**

```
import org.springframework.stereotype.Repository;

@Repository
public class UserRepository {
    public void saveUser() {
        System.out.println("User saved to database!");
    }
}
```

3. @Service - Specialized version of @Component for **service layer** classes.

- **Use case:** Marks a class that holds business logic.
- **Example:**

```
import org.springframework.stereotype.Service;

@Service
public class UserService {
    public void processUser() {
        System.out.println("Processing user in service layer!");
    }
}
```

4. @Controller - Specialized version of @Component for **web controllers** in Spring MVC.

- **Use case:** Marks a class as a controller that handles HTTP requests.
- **Example:**

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class HomeController {

    @GetMapping("/")

    public String home() {

        return "home"; // returns view name

    }

}
```

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);
    }

}
```

Ans.)

The issue is with this line: `@Value("app.name")`

The, "app.name" is treated as a **hardcoded String**, not as a property key from `application.properties`.

That's why appName is always null.

Fix : Use `${}` to tell Spring to fetch the value from `application.properties`:

@Component

```
public class AppNamePrinter {  
    @Value("${app.name}") // Fetches property value correctly  
    private String appName;  
  
    public void printAppName() {  
        System.out.println("Application Name: " + appName);  
    }  
}
```

9. What does the @SpringBootApplication annotation do?

Ans.) The SpringBootApplication annotation is a meta-annotation in Spring Boot that combines Configuration, EnableAutoConfiguration, and ComponentScan. It simplifies application setup by configuring Spring Beans, enabling auto-configuration, and scanning for components in the specified package.

Uses :

- It simplifies Spring Boot configuration.
- Instead of writing all three annotations separately, you just use @SpringBootApplication.
- @SpringBootApplication makes it easy to start a Spring Boot application by enabling **configuration**, **auto-configuration**, and **component scanning** in a single step.

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

Ans.) mvn spring-boot:run

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;  
}
```

Ans.) GitHub link - <https://github.com/Harshit-Raj-14/PayPal-RG-Assignment-Harshit-Raj/tree/main/Week%202/Assignment%201/Maven%2C%20Spring%20and%20Spring%20Boot/Employee-Spring>

12. Implement 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;  
}
```

Ans.) GitHub link – <https://github.com/Harshit-Raj-14/PayPal-RG-Assignment-Harshit-Raj/tree/main/Week%202/Assignment%201/Maven%2C%20Spring%20and%20Spring%20Boot/Employee-SpringBoot-JPA>

13. Follow the demo in the pre-work link

<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.

Ans.) GitHub link - <https://github.com/Harshit-Raj-14/PayPal-RG-Assignment-Harshit-Raj/tree/main/Week%202/Assignment%201/Maven%20C%20Spring%20and%20Spring%20Boot/Custom-SpringBatch>