

# Spring Core\_Maven Exercises

## Exercise 1: Configuring a Basic Spring Application

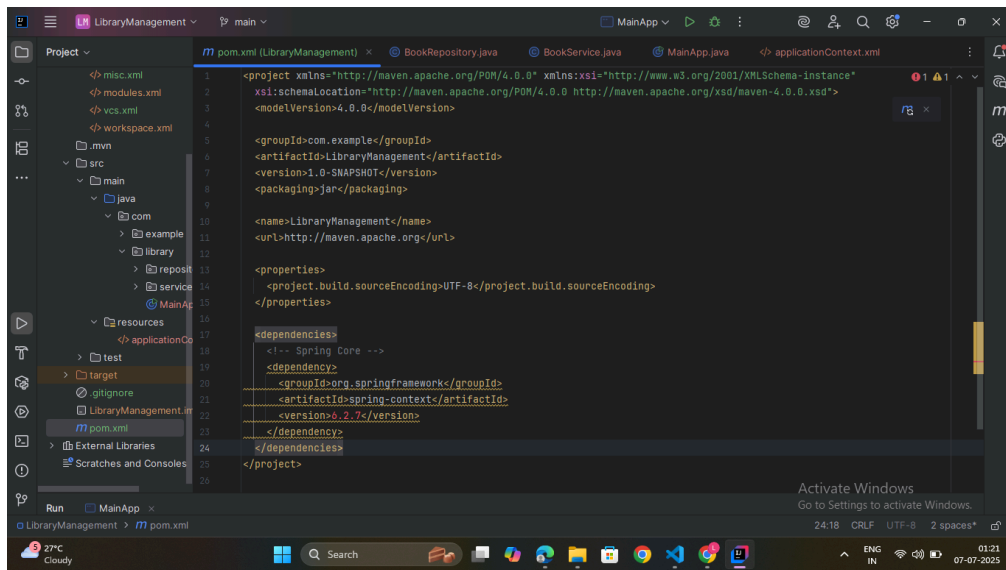
### Scenario:

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

### Solution:-

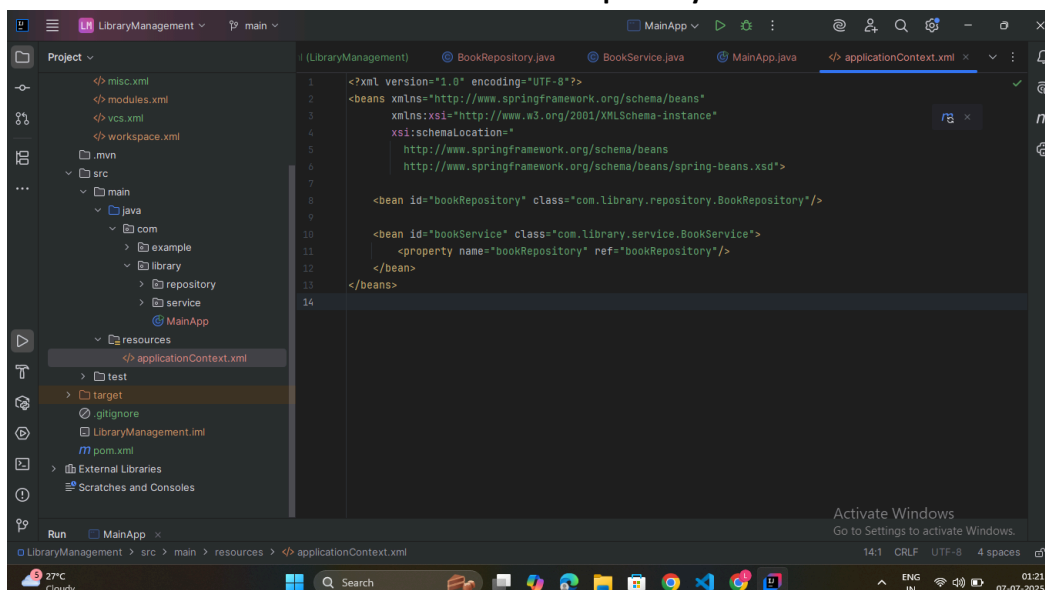
#### 1. Set Up a Spring Project:

- Create a Maven project named **LibraryManagement**.  
Add Spring Core dependencies in the **pom.xml** file.



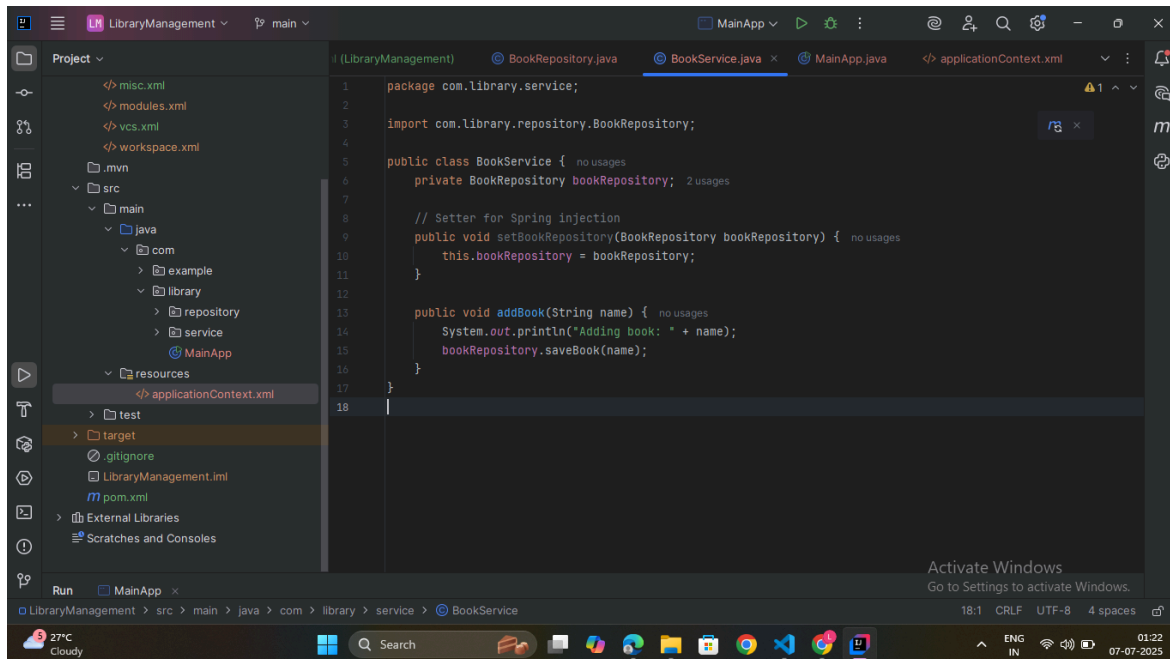
#### 2. Configure the Application Context:

- Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
- Define beans for **BookService** and **BookRepository** in the XML file.



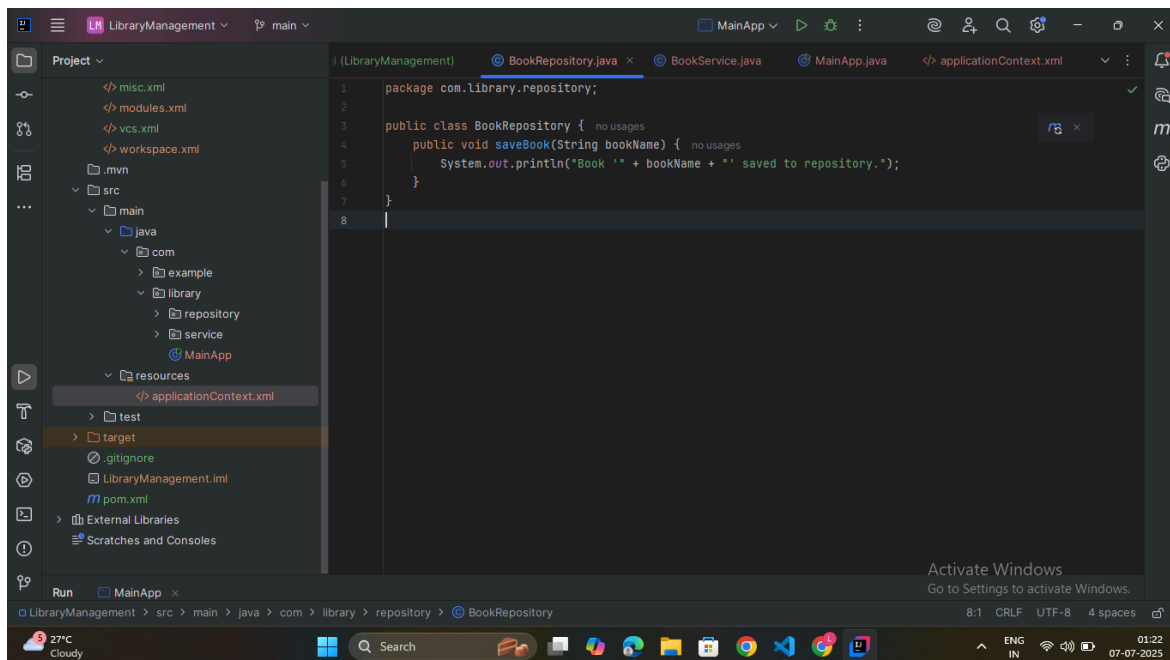
### 3. Define Service and Repository Classes:

- Create a package **com.library.service** and add a class **BookService**.
- Create a package **com.library.repository** and add a class **BookRepository**.



This screenshot shows an IDE window with the `BookService.java` file open. The package is `com.library.service`. It imports `com.library.repository.BookRepository`. The class `BookService` has a private field `bookRepository` and two methods: `setBookRepository` for Spring injection and `addBook` which prints the book name and saves it to the repository.

```
1 package com.library.service;
2
3 import com.library.repository.BookRepository;
4
5 public class BookService {
6     private BookRepository bookRepository;
7
8     // Setter for Spring injection
9     public void setBookRepository(BookRepository bookRepository) {
10         this.bookRepository = bookRepository;
11     }
12
13     public void addBook(String name) {
14         System.out.println("Adding book: " + name);
15         bookRepository.saveBook(name);
16     }
17 }
18
```

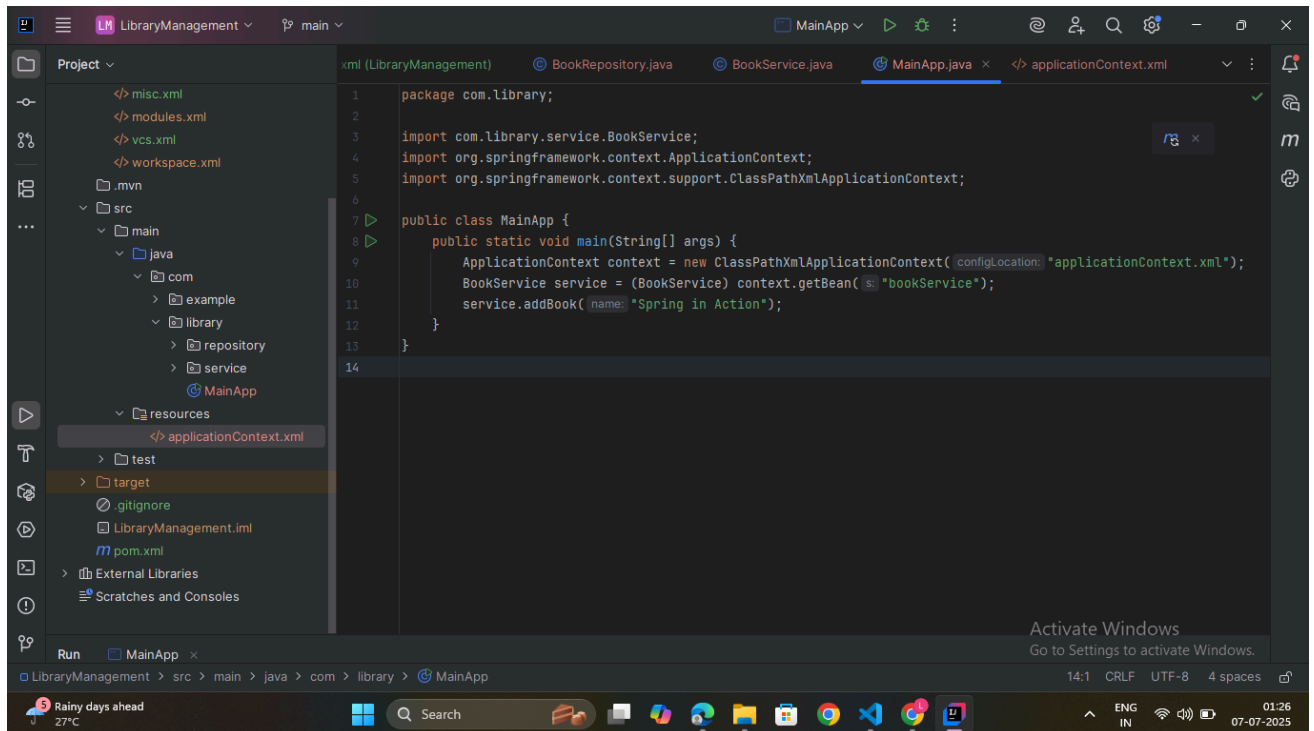


This screenshot shows the same IDE window with the `BookRepository.java` file open. The package is `com.library.repository`. The class `BookRepository` has a single method `saveBook` which prints the book name and saves it to the repository.

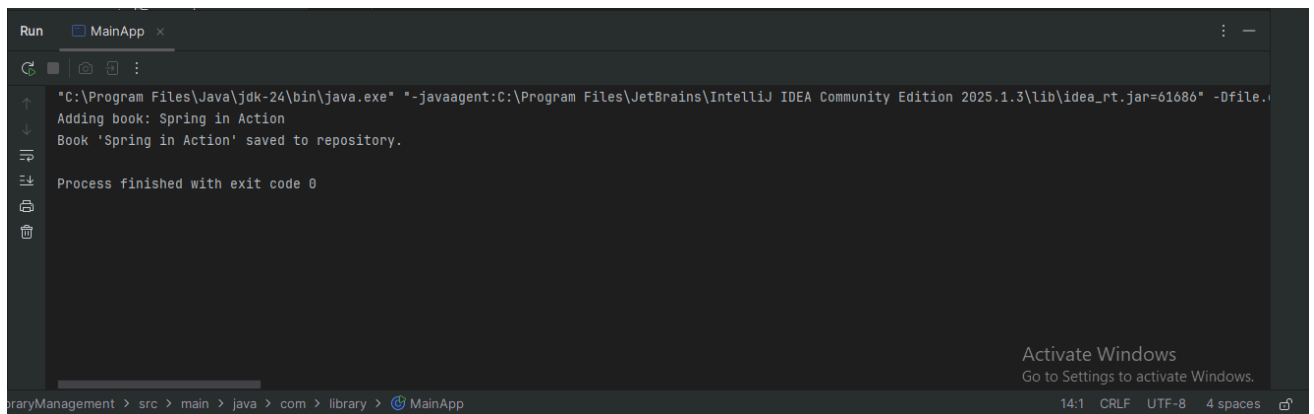
```
1 package com.library.repository;
2
3 public class BookRepository {
4     public void saveBook(String bookName) {
5         System.out.println("Book " + bookName + " saved to repository.");
6     }
7 }
8
```

#### 4. Run the Application:

- Create a main class to load the Spring context and test the configuration.



#### OUTPUT:-



## Exercise 2: Implementing Dependency Injection

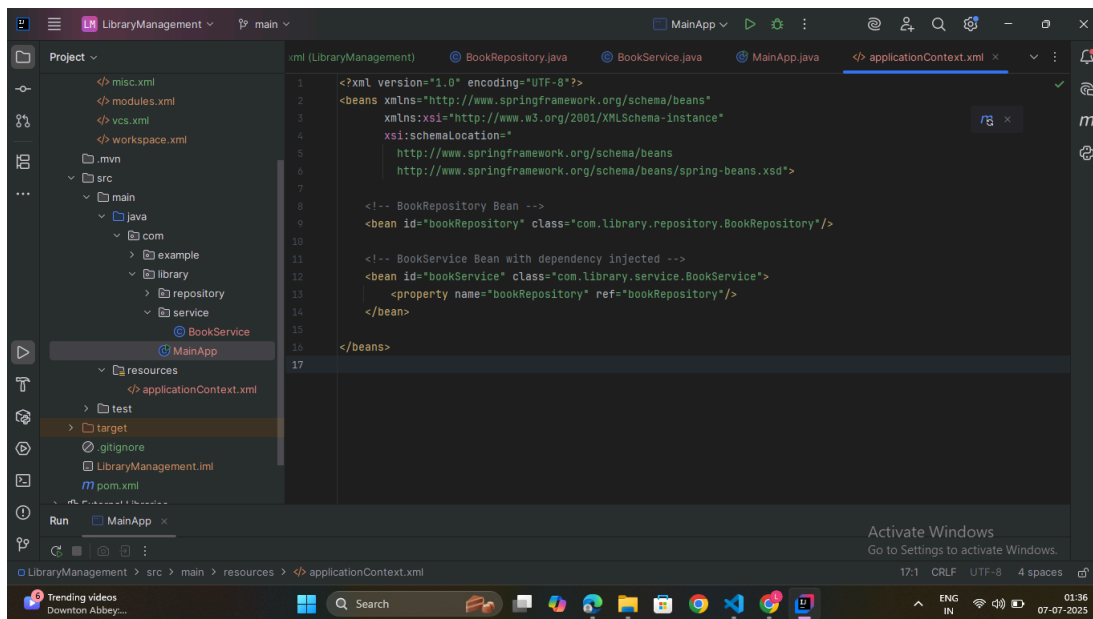
### Scenario:

In the library management application, you need to manage the dependencies between the `BookService` and `BookRepository` classes using Spring's IoC and DI.

### Solution:-

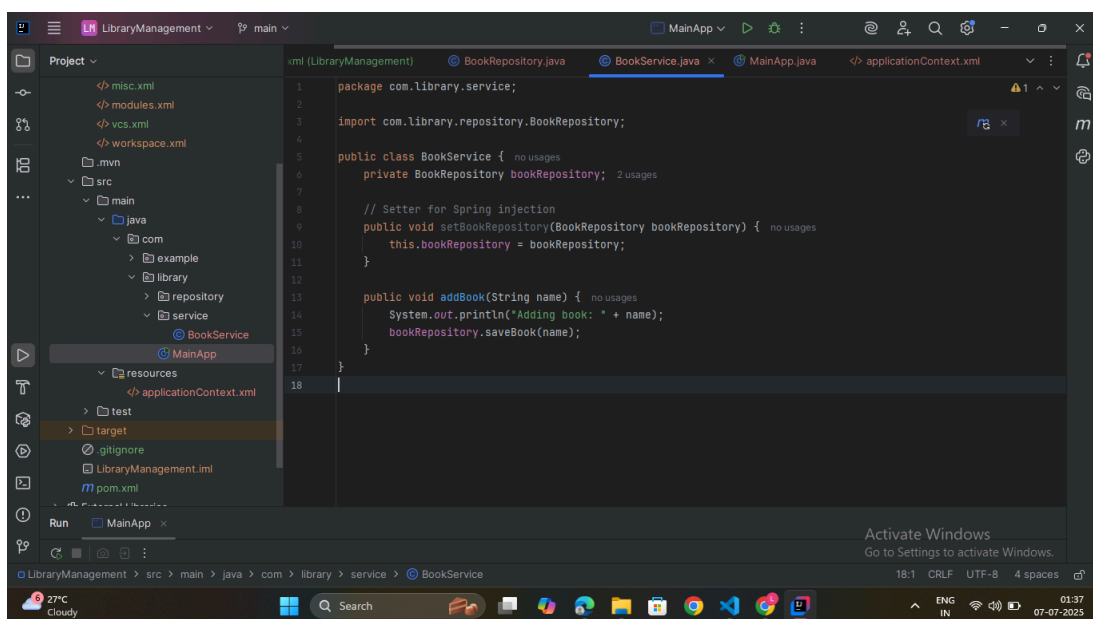
#### 1. Modify the XML Configuration:

- Update `applicationContext.xml` to wire `BookRepository` into `BookService`.



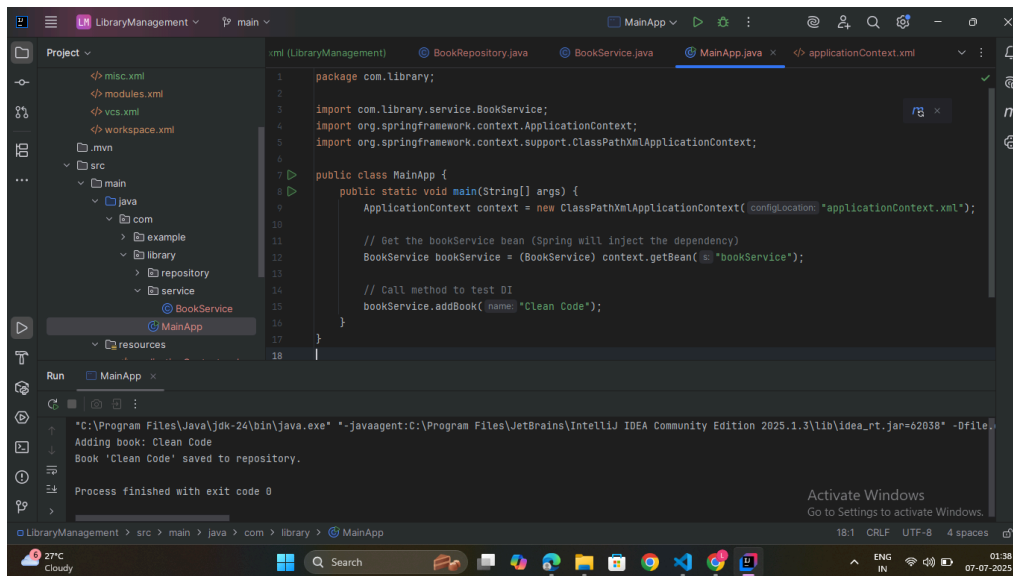
#### 2. Update the BookService Class:

- Ensure that `BookService` class has a setter method for `BookRepository`.



### 3. Test the Configuration:

- Run the **LibraryManagementApplication** main class to verify the dependency injection.



## Exercise 4: Creating and Configuring a Maven Project

### Scenario:

You need to set up a new Maven project for the library management application and add Spring dependencies.

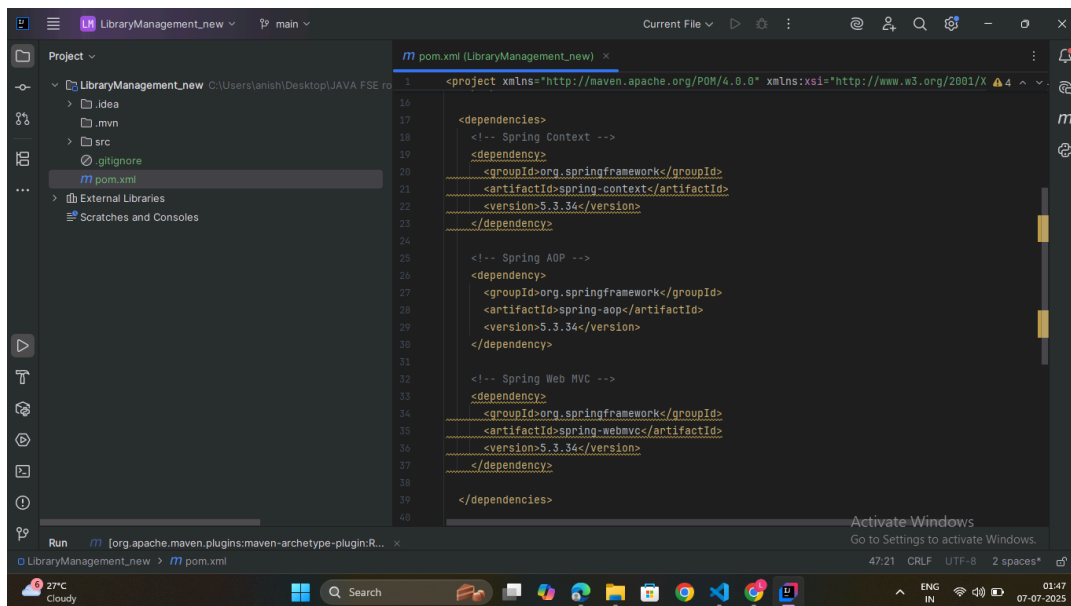
### Solution:-

#### 1. Create a New Maven Project:

- Create a new Maven project named **LibraryManagement**.

#### 2. Add Spring Dependencies in pom.xml:

- Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.



#### 3. Configure Maven Plugins:

- Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

