**Escuela Colombiana de Ingeniería Julio Garavito**

**Carrera:** INGENIERIA DE SISTEMAS.

**Materia:** CVDS

**Título: LABORATORIO 3 - TDD**

**Estudiante:** SANTIAGO ARTEAGA GUTIERREZ & CRISTIAN DAVID POLO GARRIDO

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

DEPARTAMENTO DE CIENCIAS NATURALES

CICLOS DE VIDA DE DESARROLLO DE SOFTWARE

GRUPO 02

Bogotá, Colombia

2025 – 1

# LABORATORIO 3 – TDD

#### TALLER 3

##### Testing – TDD

## PRE-RREQUISITOS

* Java OpenJDK Runtime Environment: 17.x.x
* Apache Maven: 3.9.x
* JUnit: 5.x.x
* Docker

## OBJETIVOS

1. Como hacer pruebas unitarias.
2. Utilizar anotaciones @Test del framework JUnit
3. Aplicar TDD.

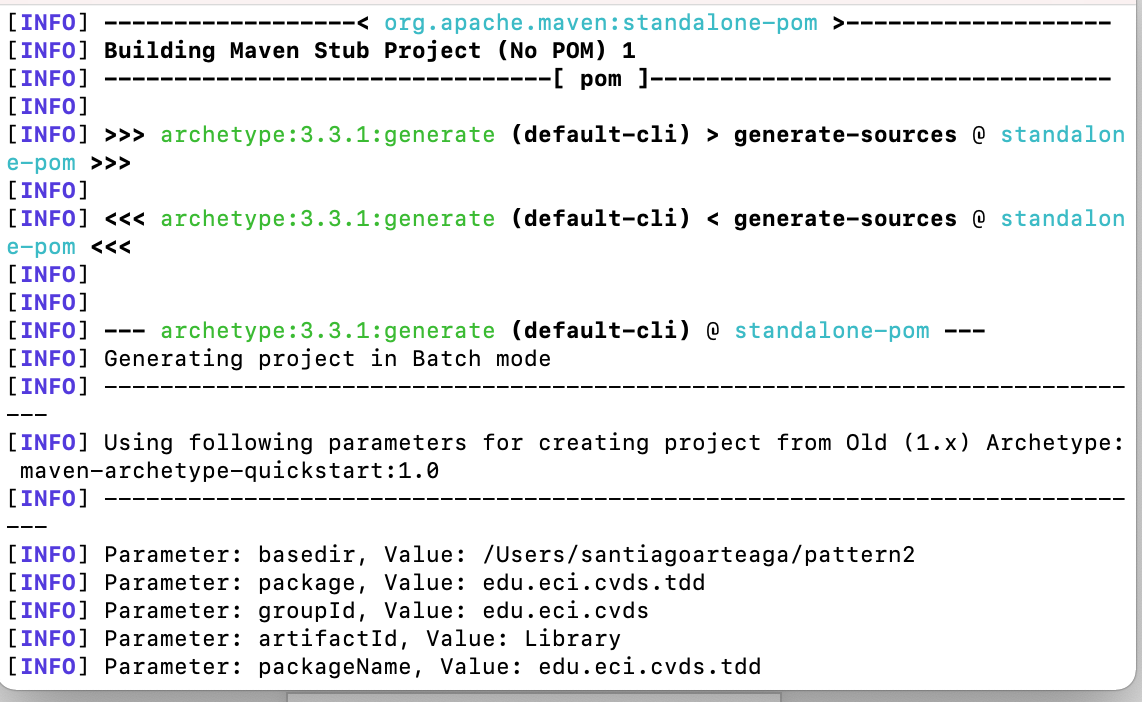
## DESCRIPCIÓN PROYECTO

El proyecto consiste en un sistema de gestión de bibliotecas, donde hay clases que representan Libro, Usuario, Prestamo, y Biblioteca. Los usuarios pueden tomar prestados libros de la biblioteca, y la Biblioteca se encarga de gestionar los préstamos, asegurarse de que los libros estén disponibles, y mantener un registro de los libros prestados.

### CREAR PROYECTO CON MAVEN

Deben crear un proyecto maven con los siguientes parámetros:

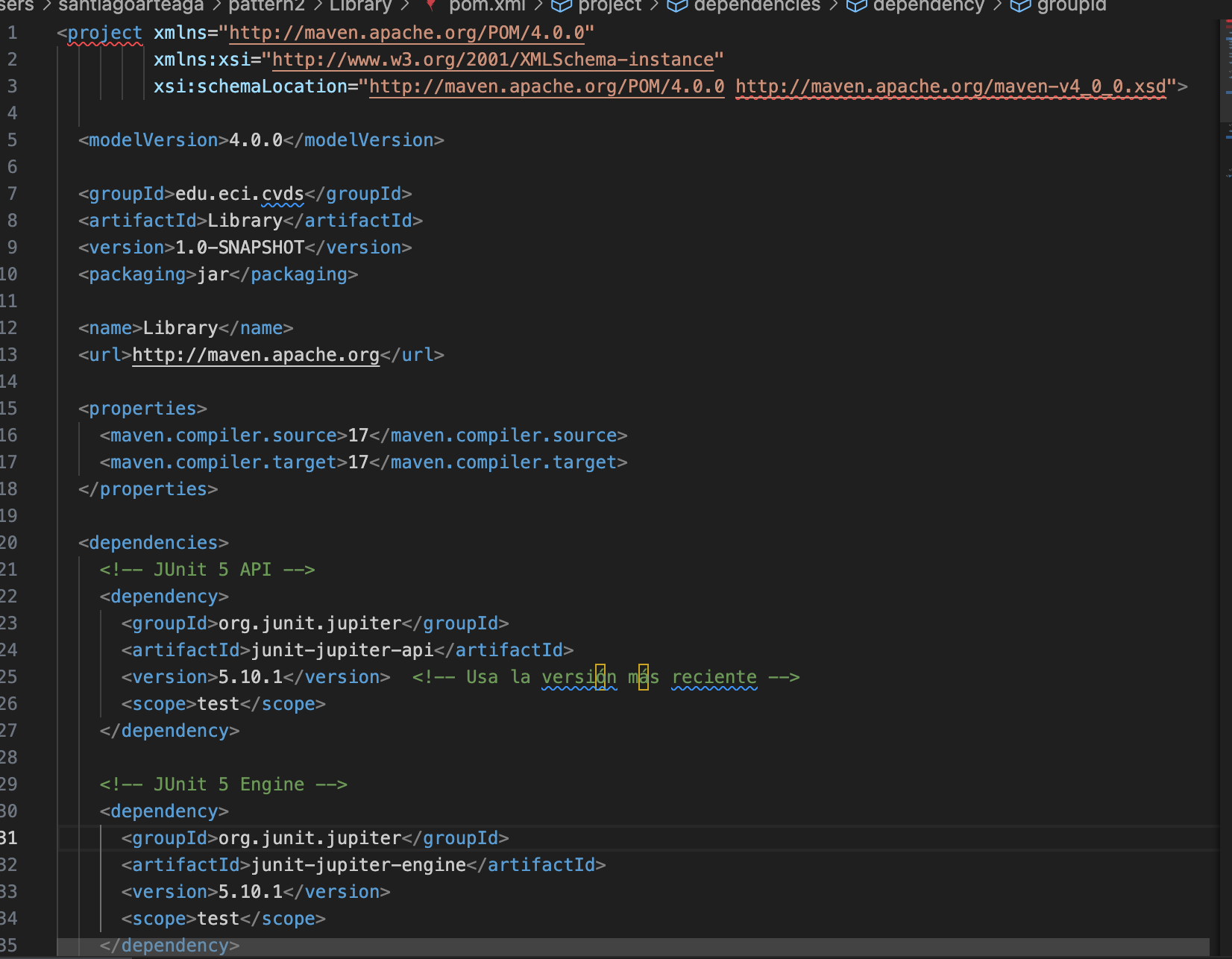


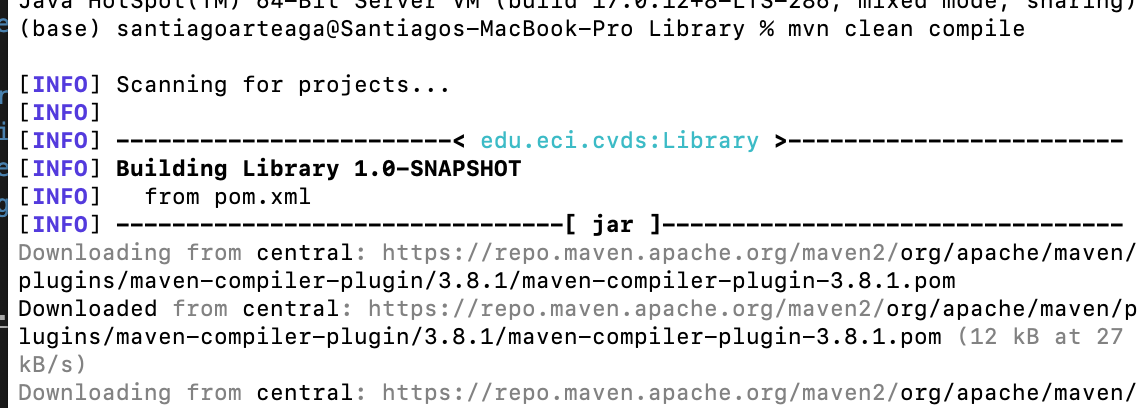


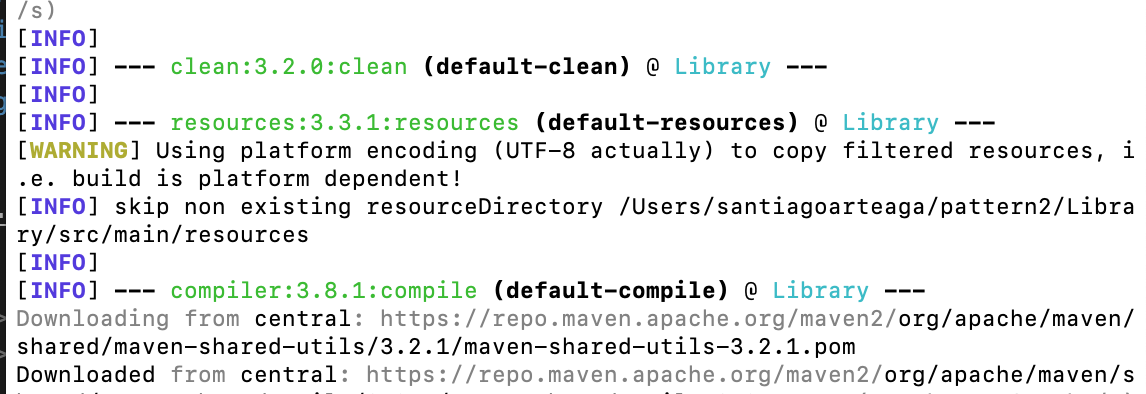
### AGREGAR DEPENDENCIA JUNIT5

* Buscar en maven central la dependencia de JUnit5 en su versión más reciente.
* Edite el archivo pom.xml del proyecto para agregar la dependencia.
* Verifique que la versión de java sea la 17

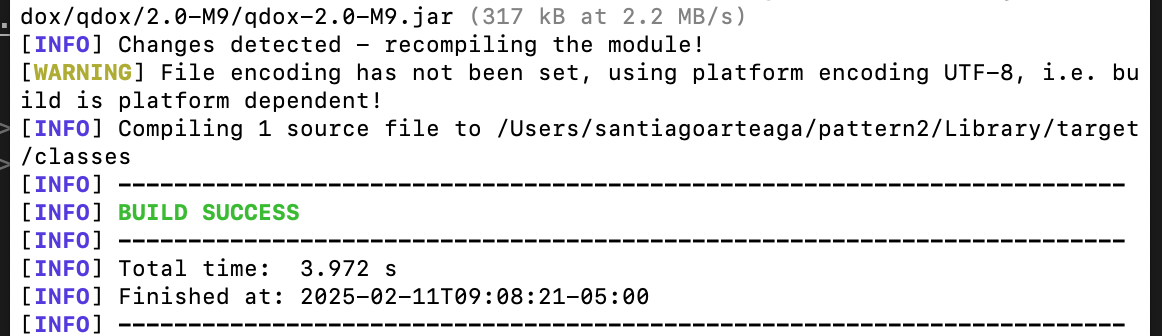








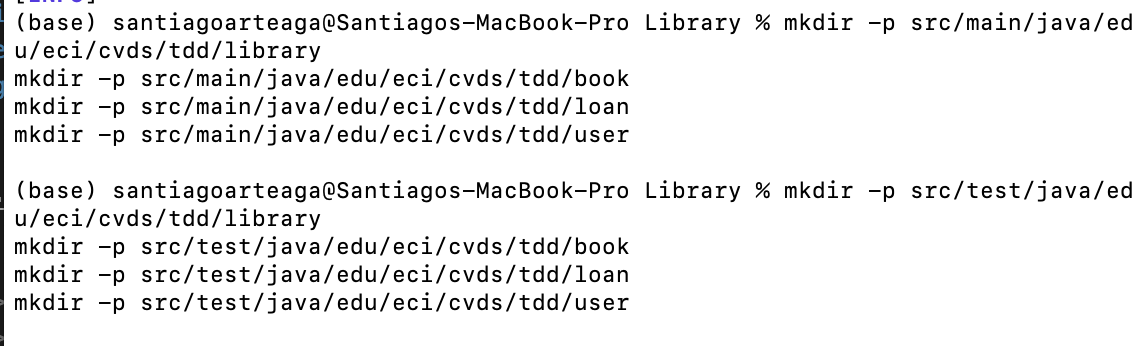
* Compile el proyecto para validar que todo este bien.



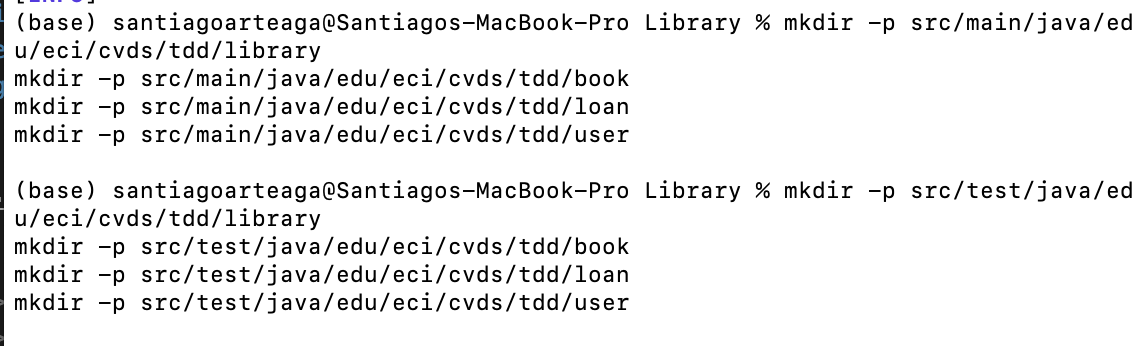
### AGREGAR ESQUELETO DEL PROYECTO

Cree los siguientes paquetes dentro de edu.eci.cvds.tdd

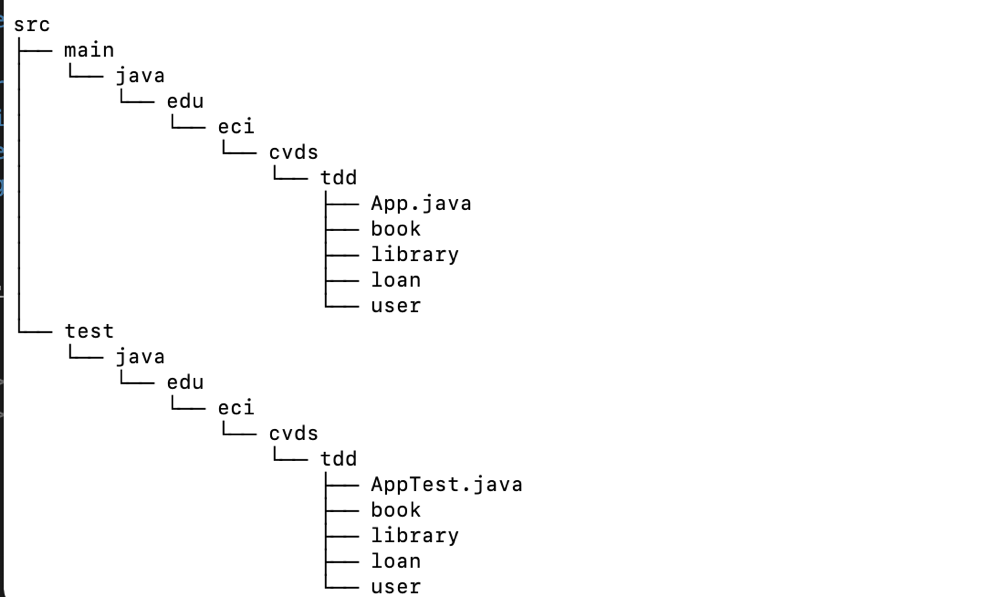
* library
  + book
  + loan
  + user



Estos paquetes también se deben crear en la carpeta de test.







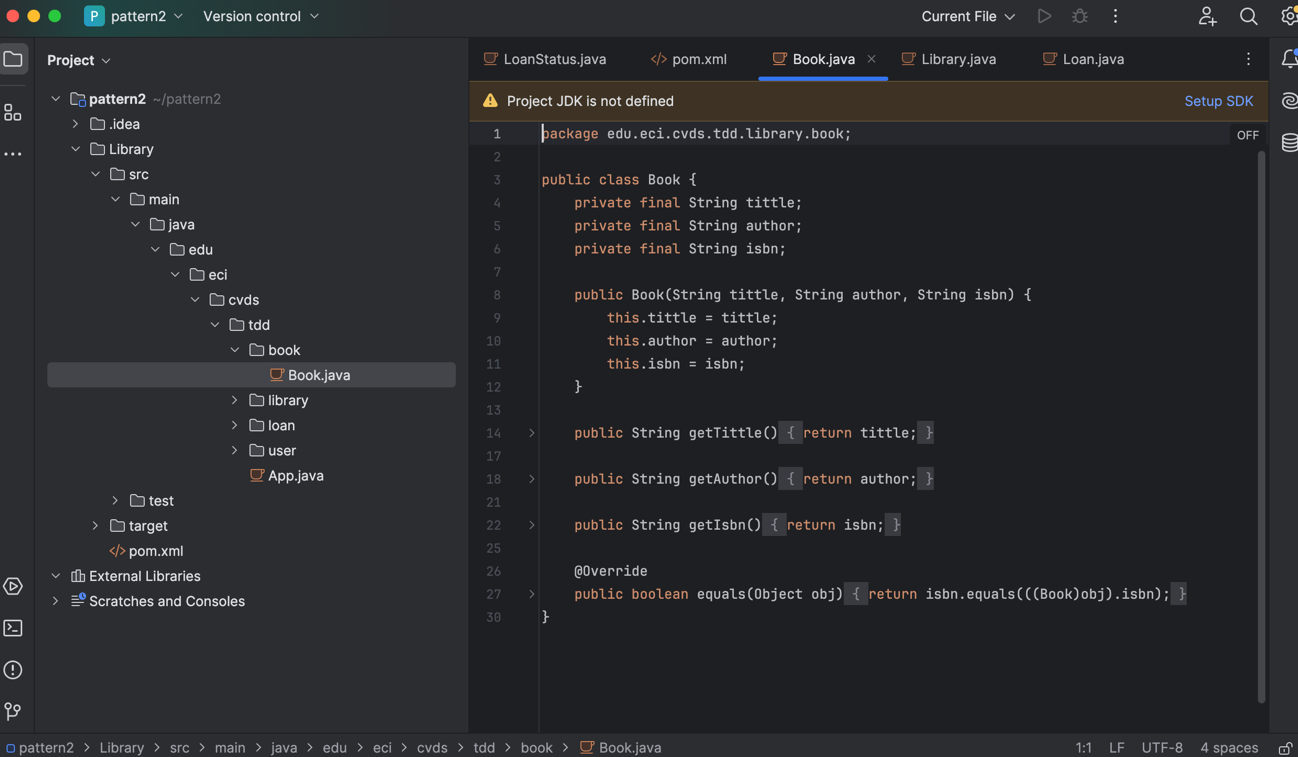
Luego verificamos que todo este compilando:



### AGREGAR CLASES

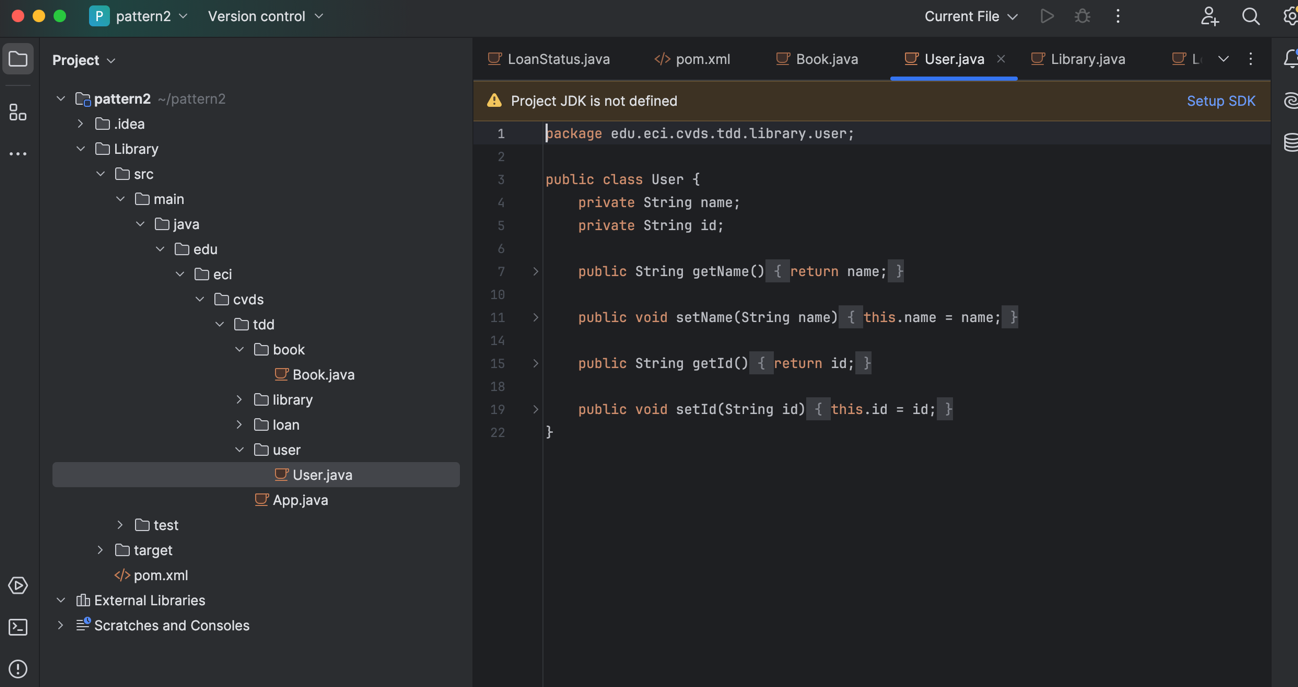
En el paquete edu.eci.cvds.tdd.library.book cree la siguiente clase:





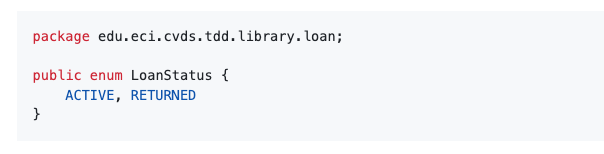
A continuación en el paquete edu.eci.cvds.tdd.library.user cree la siguiente clase:

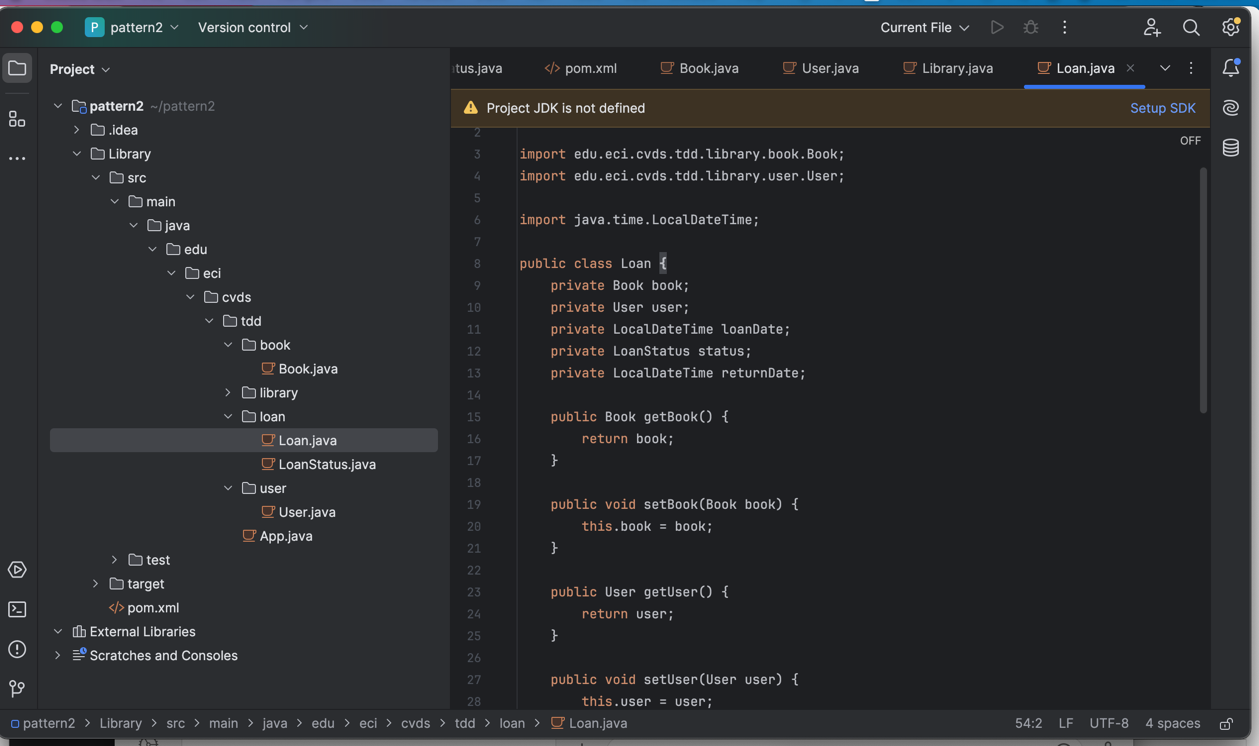


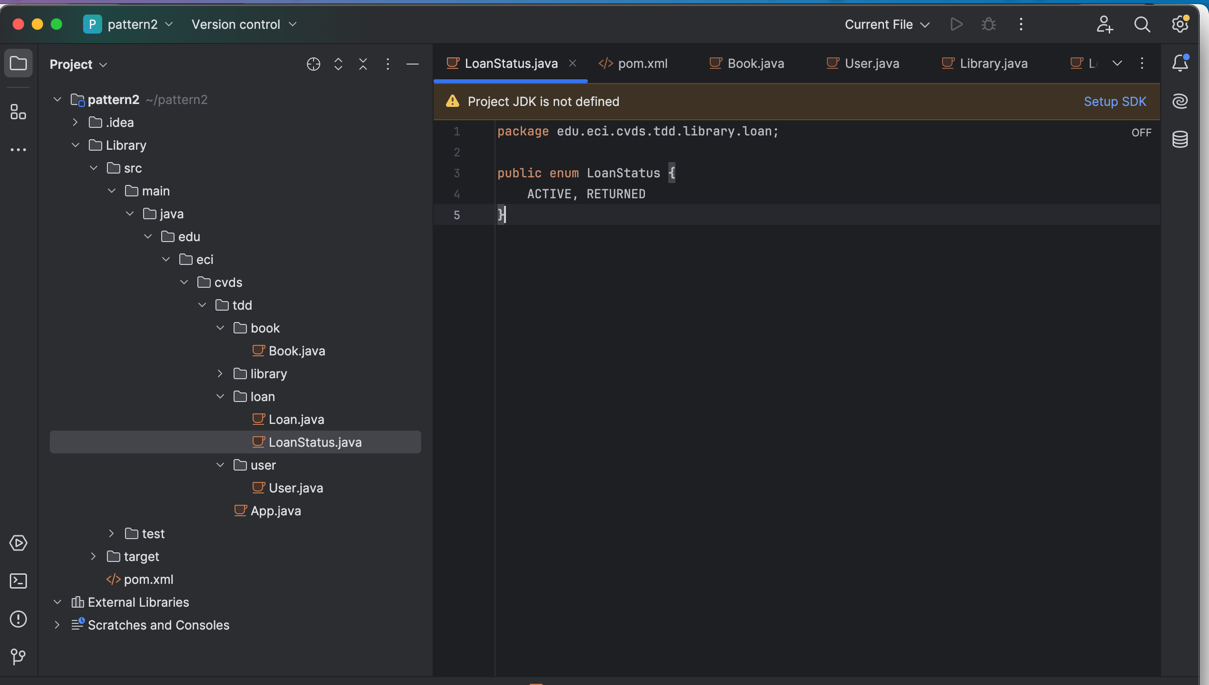


En el paquete edu.eci.cvds.tdd.library.loan se deben crear las clases Loan y el enum LoanStatus:

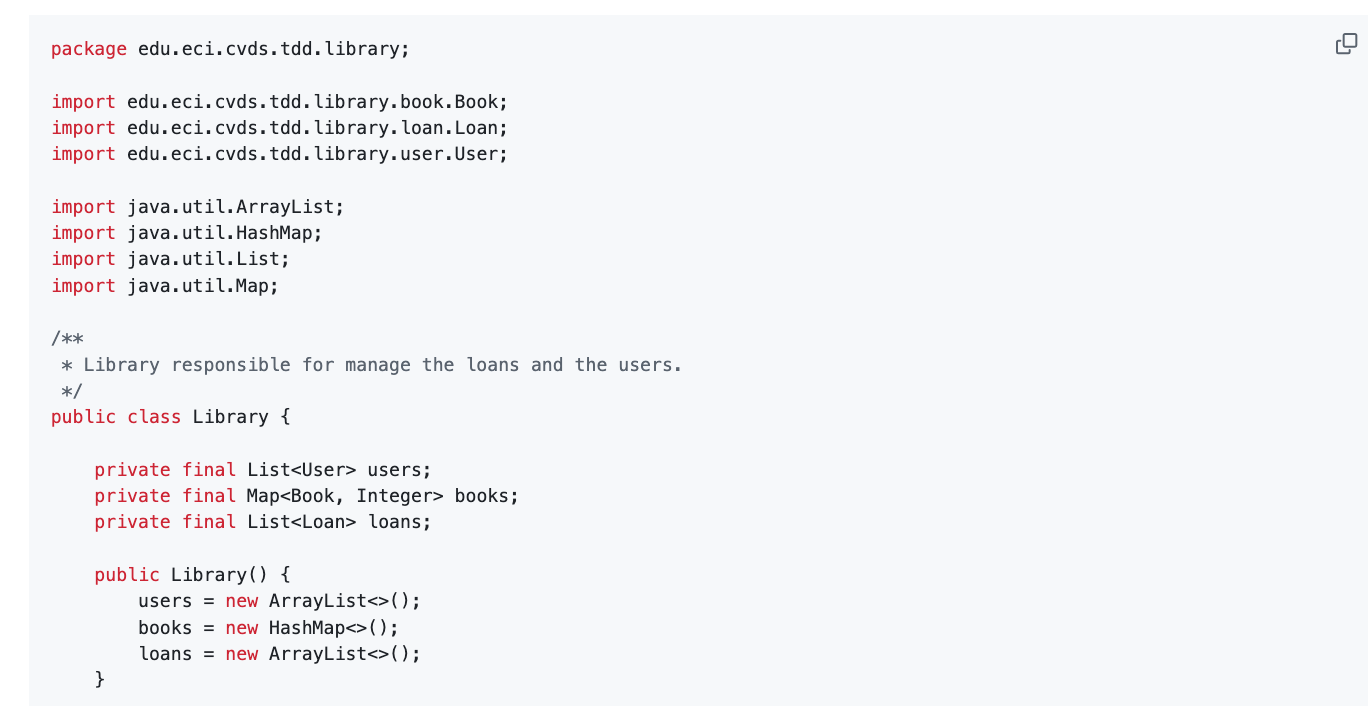


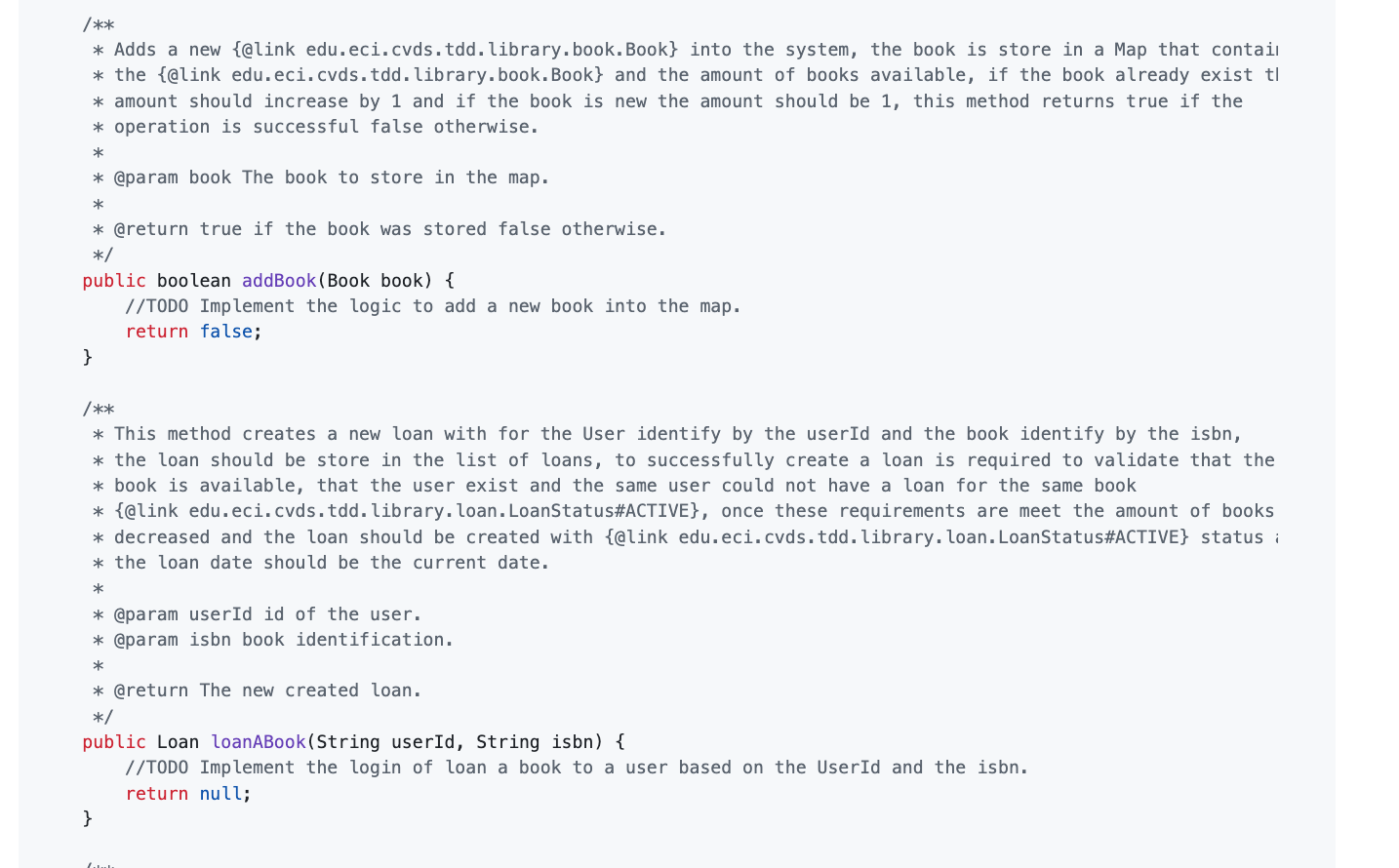


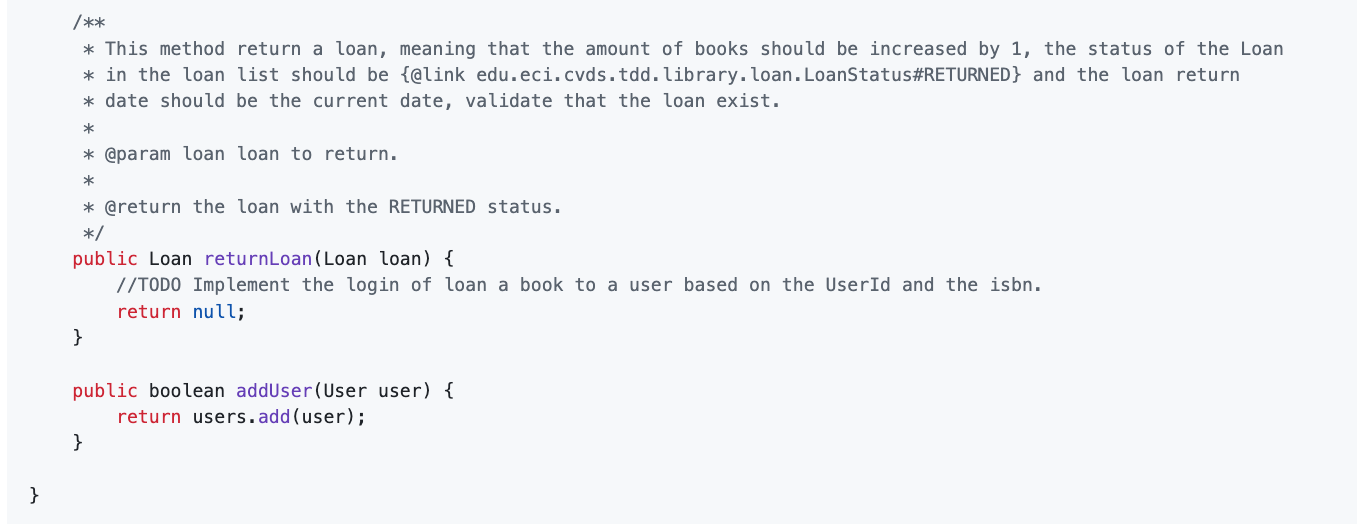


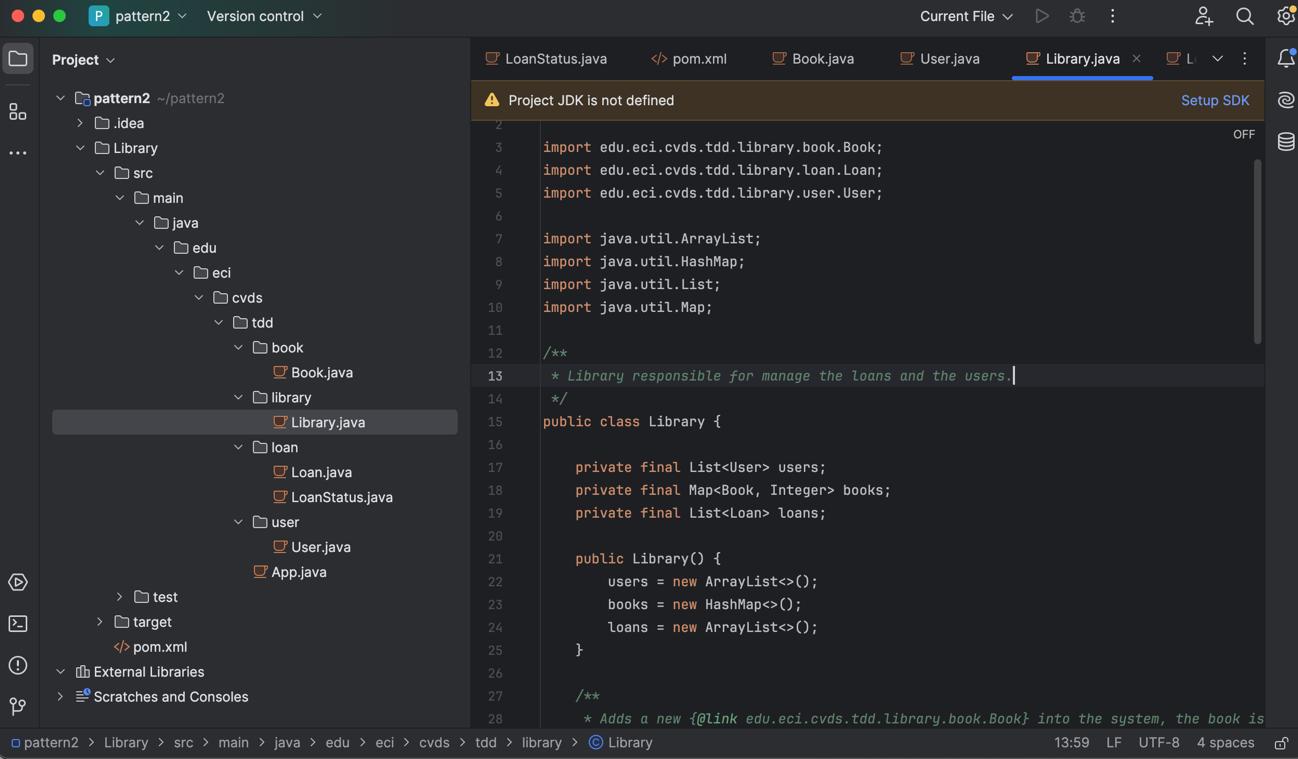


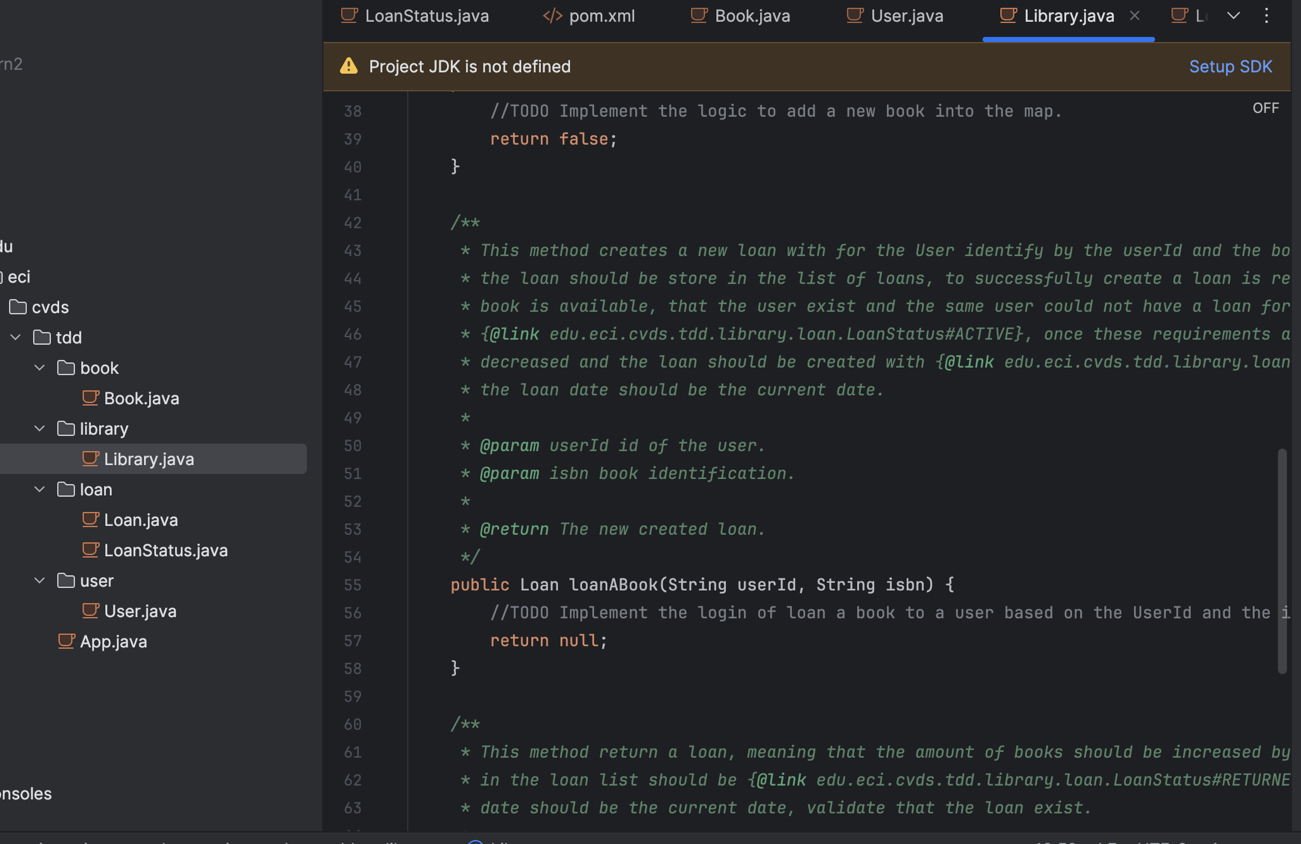
por último se debe crear la siguiente clase en el paquete edu.eci.cvds.tdd.library



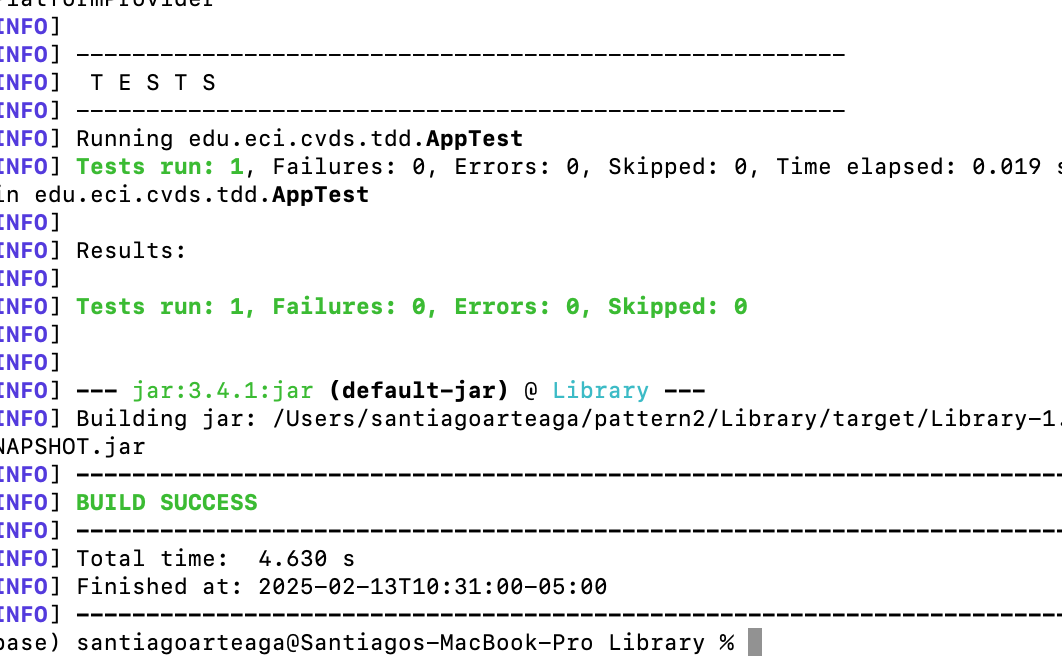








Para validar que la estructura del proyecto está bien se debe compilar usando el comando package.

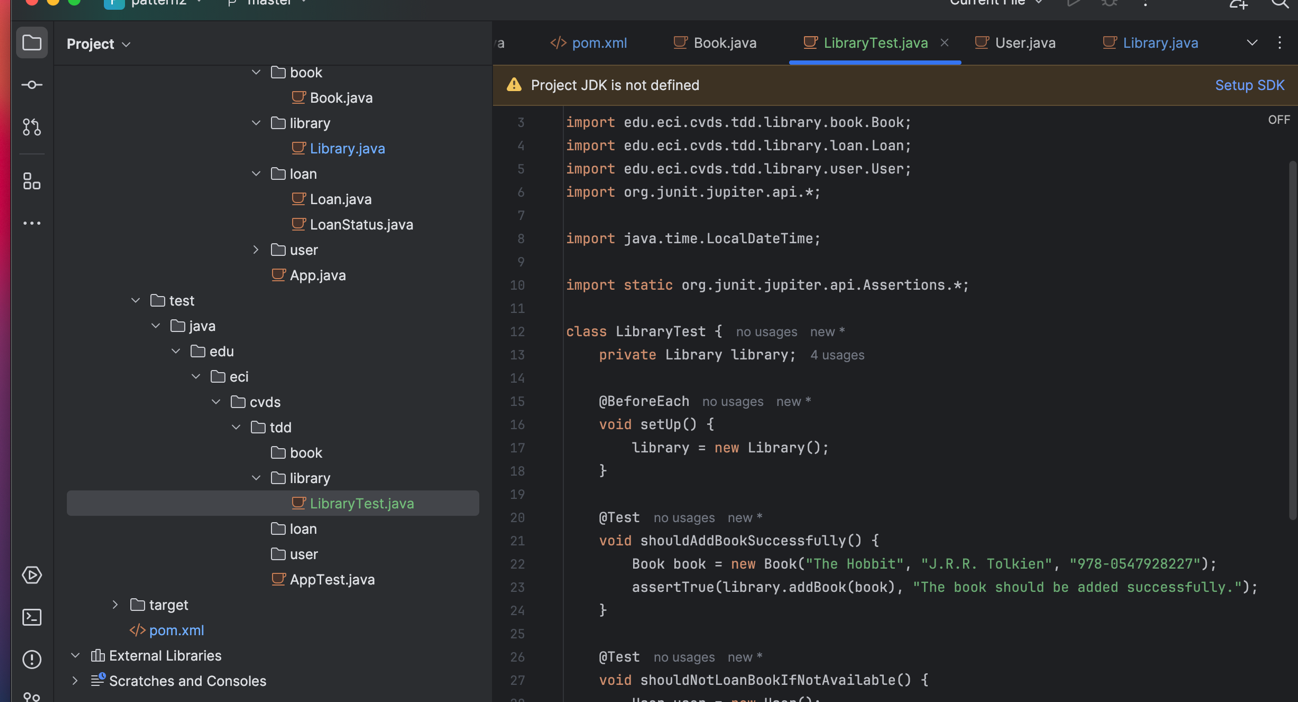


## PRUEBAS UNITARIAS Y TDD

Para poder implementar los métodos addBook, loanABook y returnLoan de la clase Library vamos a aplicar la técnica de TDD, por cada caso de prueba se debe hacer un commit, cada commit debe tener la prueba nueva y la implementación para que la prueba del commit funcione. Las pruebas anteriormente implementadas deben continuar funcionando. Como están trabajando en parejas es necesario trabajar en ramas independientes y utilizar Pull Request para mezclar los cambios.

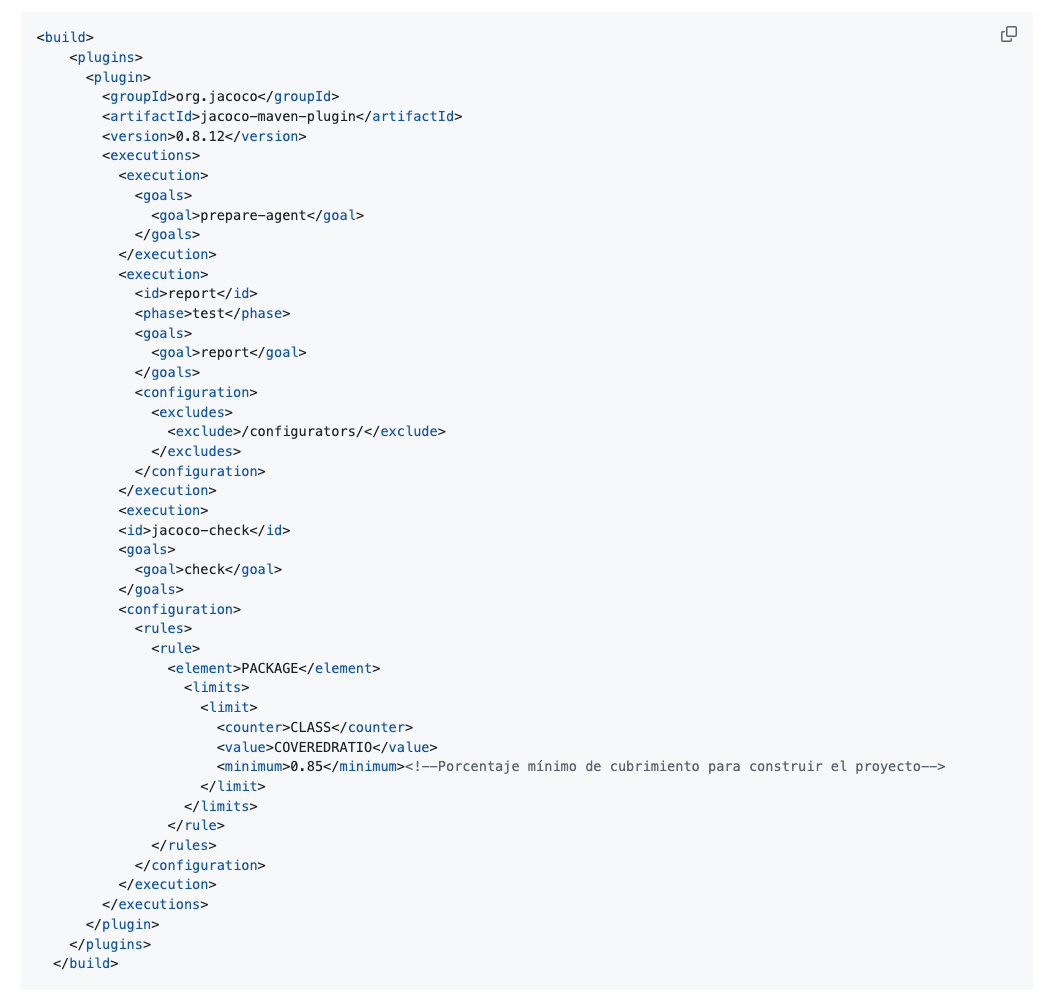
### CREAR CLASE DE PRUEBA

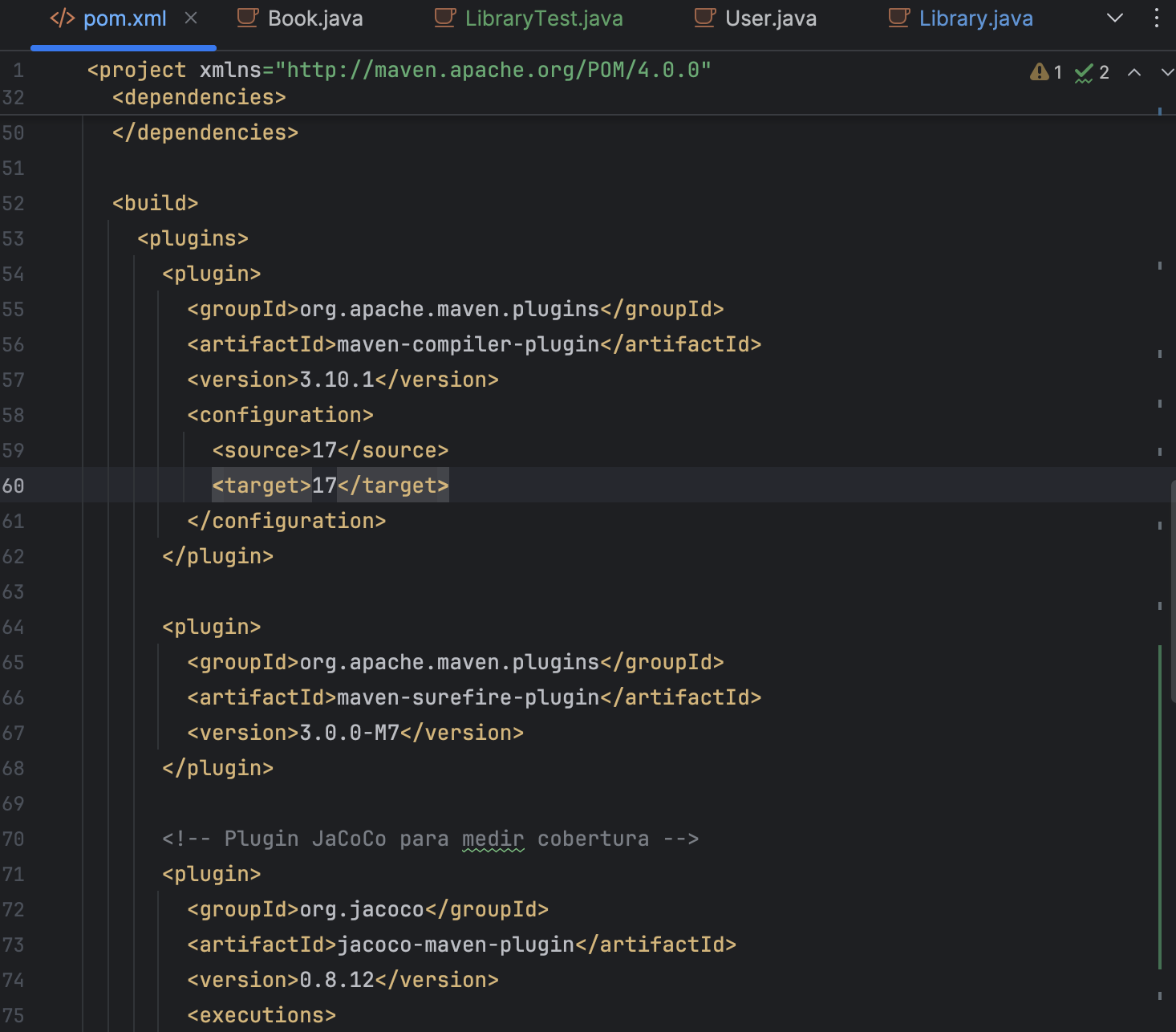
Es necesario crear la clase de prueba para edu.eci.cvds.tdd.Library, la clase debe seguir los estándares de nombres estudiados en clase.

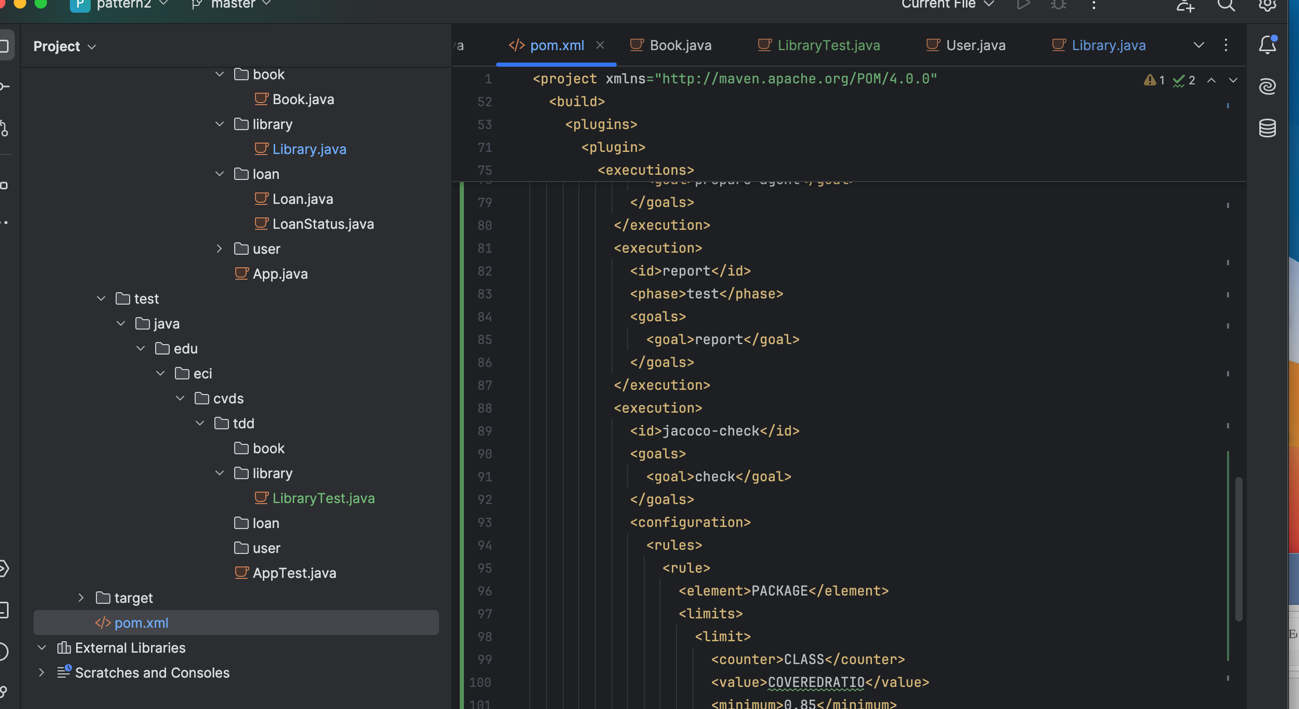
Para pensar en los casos de pruebas lean detenidamente el javadoc de los métodos para reconocer las clases de equivalencia, basados en las clases de equivalencia se debe crear una prueba la cual debe fallar y posteriormente implementar el código necesario para que funcione, este proceso se debe repetir hasta cumplir con la especificación definida en el javadoc.

### COBERTURA

* Agregar la dependencia de jacoco, utilizar la última versión disponible en maven central.
* Para usar Jacoco es necesario agregar la siguiente sección en el pom.xml

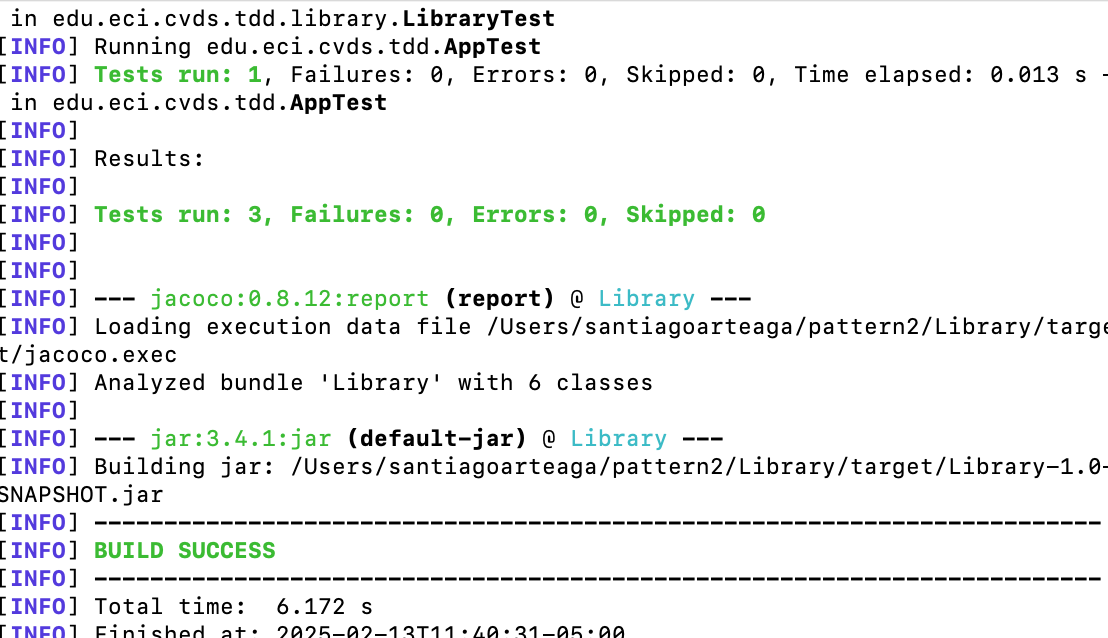






Ahora al compilar el proyecto en la carpeta target se debe crear una carpeta con el nombre site la cual tiene un index.html, al abrir dicho archivo se debe ver la cobertura total y de cada una de las clases, el objetivo es tener la cobertura superior al 80%.

Explore los links del reporte en el cual le muestra que partes del código tienen prueba y cuales no.



### SONARQUBE

Ahora es necesario hacer el análisis estático del código usando SonarQube, para lo cual necesitamos tener Docker.

* Para lo cual se debe descargar la imagen de docker con el siguiente comando docker pull sonarqube
* Ahora se debe arrancar el servicio de SonarQube con el siguiente comando docker run -d --name sonarqube -e SONAR\_ES\_BOOTSTRAP\_CHECKS\_DISABLE=true -p 9000:9000 sonarqube:latest
* Validar funcionamiento docker ps -a
* Iniciar sesión en sonar localhost:9000 cambiar la clave por defecto usuario y contraseña es admin.
* Entrar a las opciones de la cuenta.
  + Account -> settings -> generate token.
* Una vez sonar este corriendo deben generar un token
* Instale sonarLint en el IDE que este manejando.
* Añada el plugin de Sonar en el archivo pom del proyecto.



* Añada las propiedades de SonarQube y Jacoco.



* Construya el proyecto, genere el reporte de JACOCO y corrija el cubrimiento de las pruebas de unidad para que su proyecto se construya adecuadamente.
* genere la integración con sonar mvn verify sonar:sonar -D sonar.token=[TOKEN\_GENERADO]

Documente en el readme de su repositorio todas las evidencias del correcto funcionamiento de esta integración.

## ENTREGAR

* Se espera al menos que durante la sesión de laboratorio, se cree el proyecto y se agreguen las clases básicas y un par de pruebas.
* Crear un repositorio para este proyecto y agregar la url del mismo, como entrega del laboratorio.
* La entrega final se realizará en Moodle.