Dependency Injection

[redacted]

Agenda

- SOLID Principles of Object Oriented Programming
- Creating a Spring DOI Example Project
- The Spring Context
- Basics of Dependency Injection
- Dependency Injection without Spring

SOLID Principles of OOP

[redacted]

History of SOLID

- The SOLID principles date back to March 1995
- The principles originate from “Uncle Bob” Martin
- Started as writings, which ultimately were turned into the book “Agile Software Development: Principles, Patterns, and Practices”
- Michael Feathers is credited with coming up with the SOLID acronym

Why SOLID?

- Object Oriented Programming is a powerful concept
  - BUT OOP does not always lead to quality software
- The 5 principles focus on dependency management
- Poor dependency management leads to code that is brittle, fragile and difficult to change.
- Proper dependency management leads to quality code that is easy to maintain.

5 SOLID Principles

1. Single-Responsibility Principle
   - *A class should only have a single responsibility, that is, only changes to one part of the software' specification should be able to affect the specification of the class*.

Single-Responsibility Principle

- Every class should have a single responsibility
- There should never be more than one reason for a class to change
- Your classes should be small. No more than a screen full of code.
- Avoid thick “global” like classes
- Split big classes into smaller classes

5 SOLID Principles…

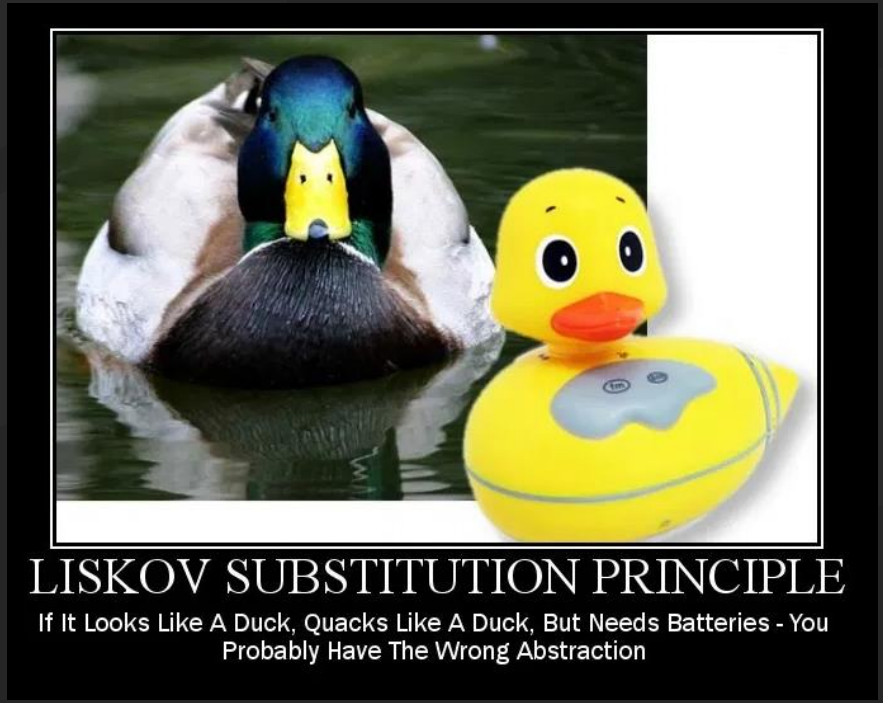
2. Open-Closed Principle
   - *Software entities, should be open for extension, but closed for modification*.

Open-Closed Principle

- Your classes should be open for extension, but closed for modification
- You should be able to extend a classes behavior without modifying it
- Use private variable with getter and setters – ONLY when you need them.
- User abstract base classes.

5 SOLID Principles…

3. Liskov Substitution Principle
   - *Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.*

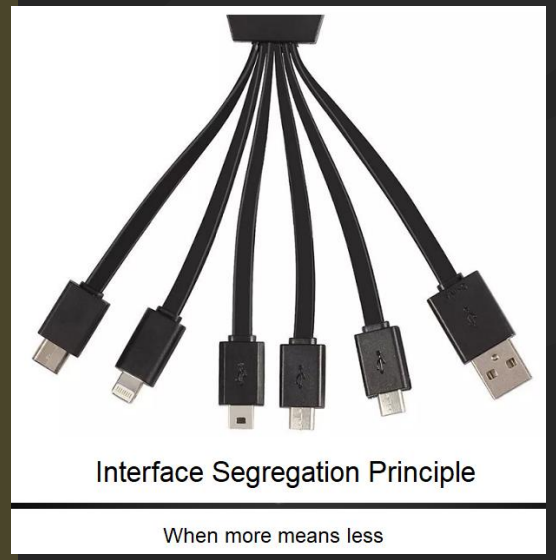


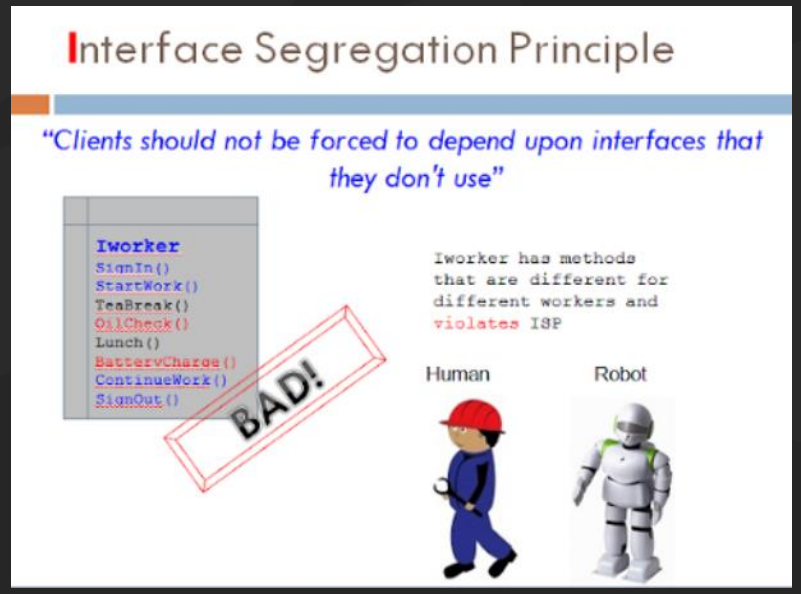
Liskov Substitution Principle

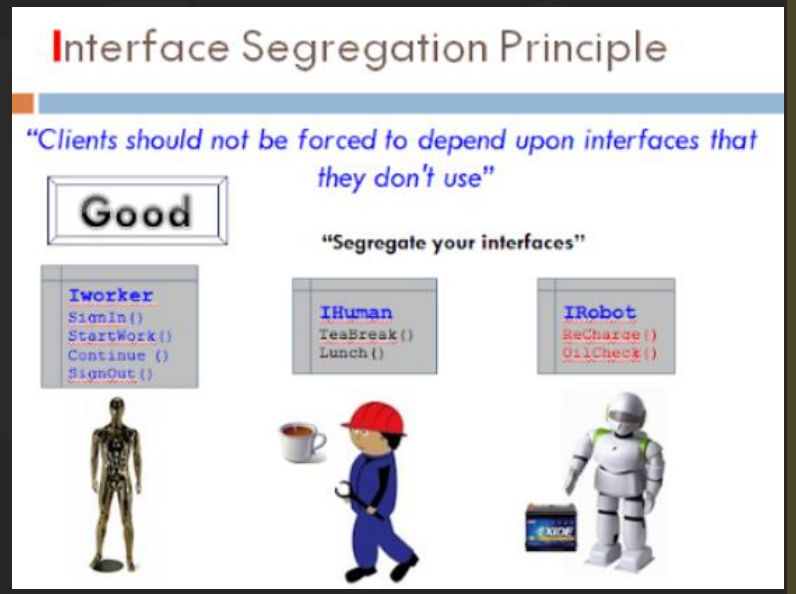
- Founded back by Barabra Liskov in 1998
- Objects in a program should be replaceable with instances of their subtypes WITHOUT altering the correctness of the program.
- Violations will often fail the “Is a ” test.
  - A Square “Is a” Rectangle
  - However, a Rectangle “Is Not” a Square

5 SOLID Principles…

4. Interface Segregation Principle
   - Many client-specific interfaces are better than one general-purpose interface.







Interface Segregation Principle

- Make fine grained interfaces that are client specific
- Many client specific interfaces are better than one “general purpose” interface
- Keep your components focused and minimize dependencies between them.
- Notice relationship to the Single Responsibility Principle?
  - i.e. → avoid big / complex interfaces.

5 SOLID Principles…

5. Dependency Inversion Principle
   - One should “depend upon abstractions, [not] concretions”.

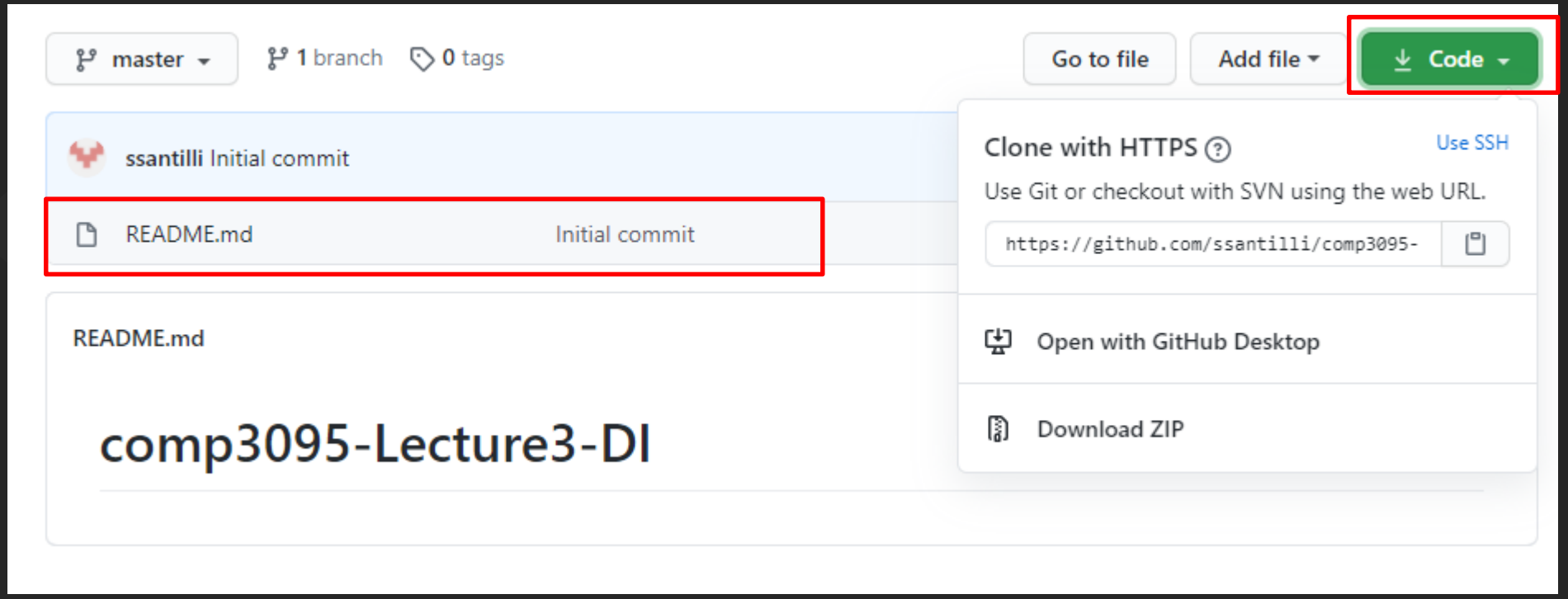
Dependency Inversion Principle

- Abstractions should not depend upon details
- Details should depend upon abstractions
- Important that higher level and lower level objects depend on the same abstract interaction.
- This is not the same as dependency injection – which is how objects obtain dependent objects.

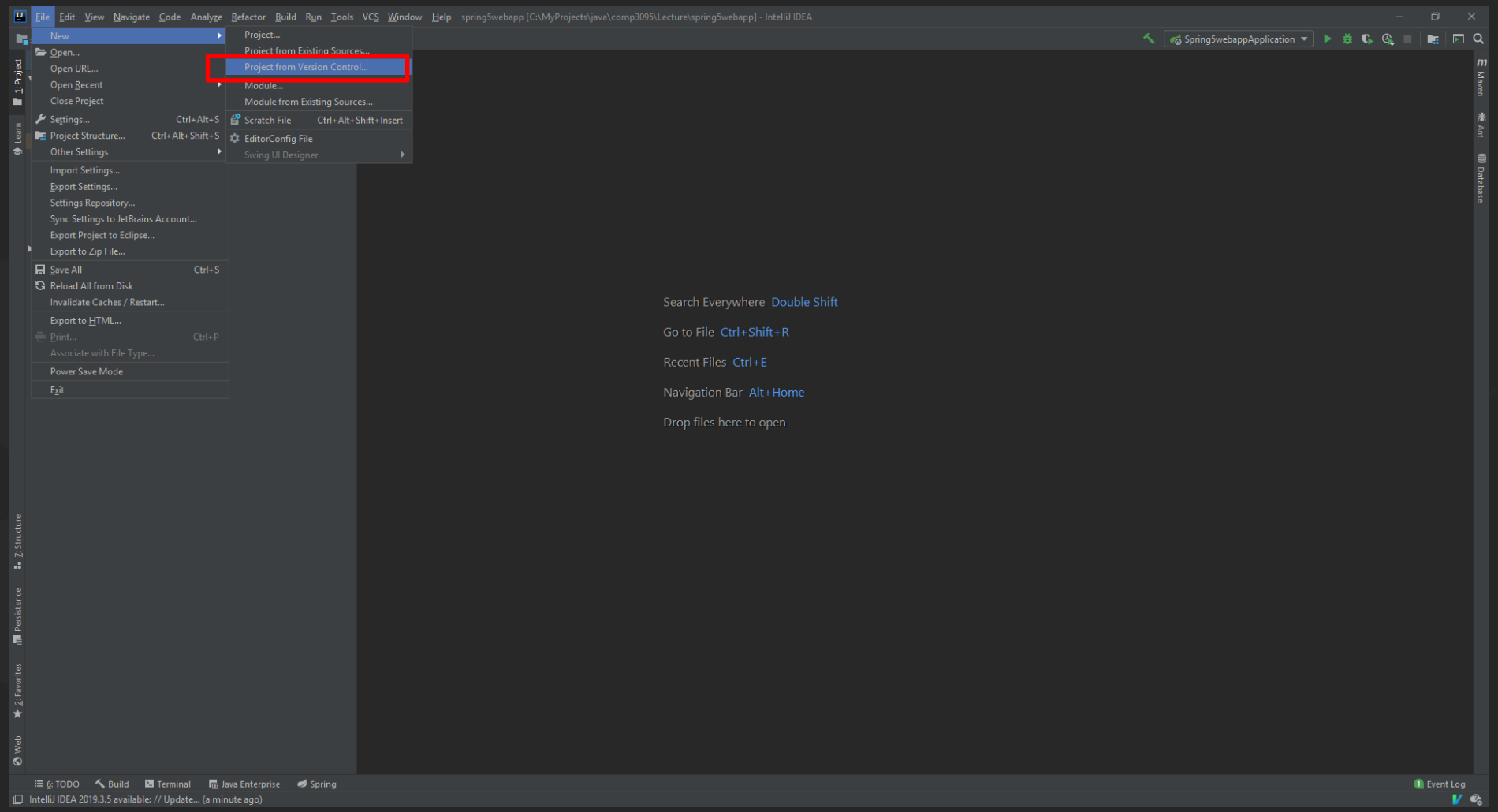
Create a Spring DI Example Project

[redacted]

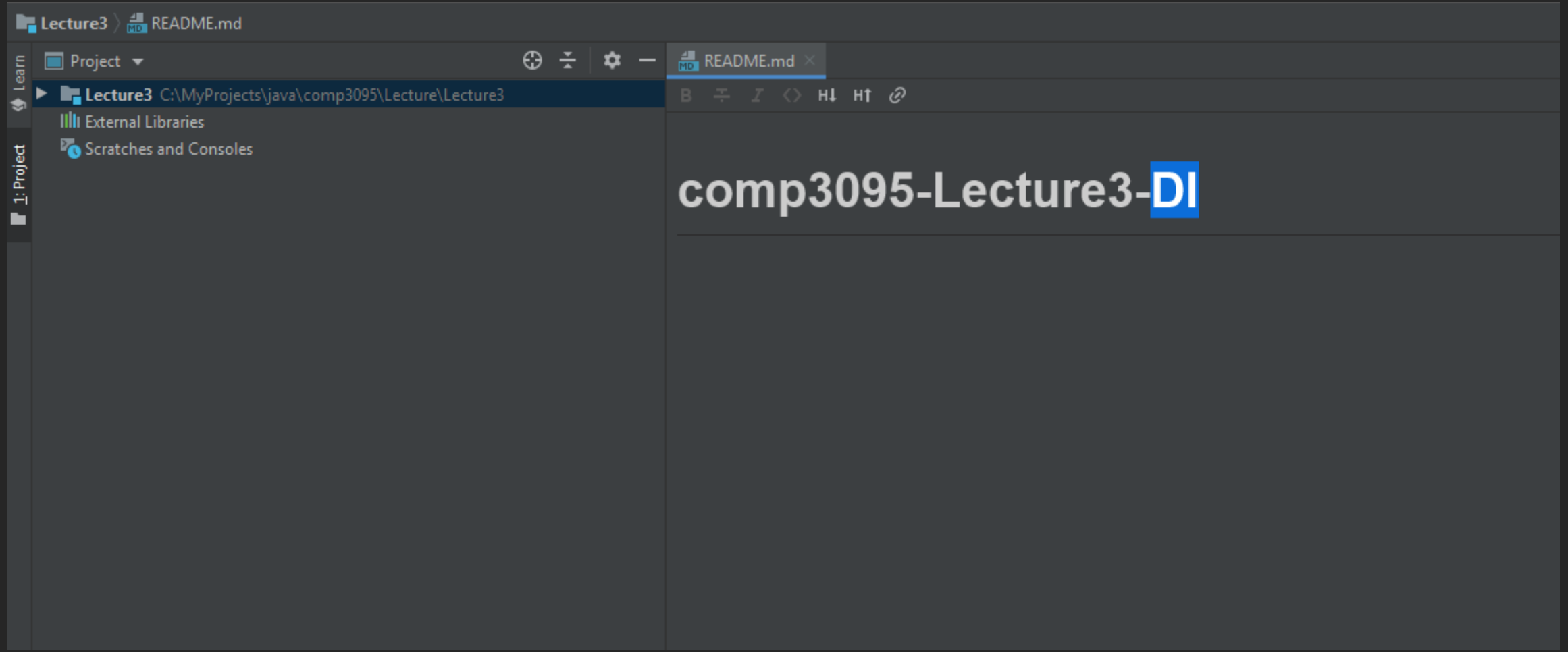
Create and Clone GIT Project



Create and Clone GIT Project…



Create and Clone GIT Project…



Create and Clone GIT Project…

- Navigate to start.spring.io and create project

No additional dependencies required for this project

spring initializr

Project
Maven Project Gradle Project

Language
Java Kotlin Groovy

Spring Boot
2.4.0 (SNAPSHOT) 2.4.0 (M2) 2.3.4 (SNAPSHOT) 2.3.3
2.2.10 (SNAPSHOT) 2.2.9 2.1.17 (SNAPSHOT) 2.1.16

Project Metadata

Group	ca.gbc.comp3095
Artifact	comp3095-Lecture3-DI
Name	comp3095-Lecture3-DI
Description	Dependency injection examples
Package name	ca.gbc.comp3095.comp3095-Lecture3-DI
Packaging	Jar War
Java	14 11 8

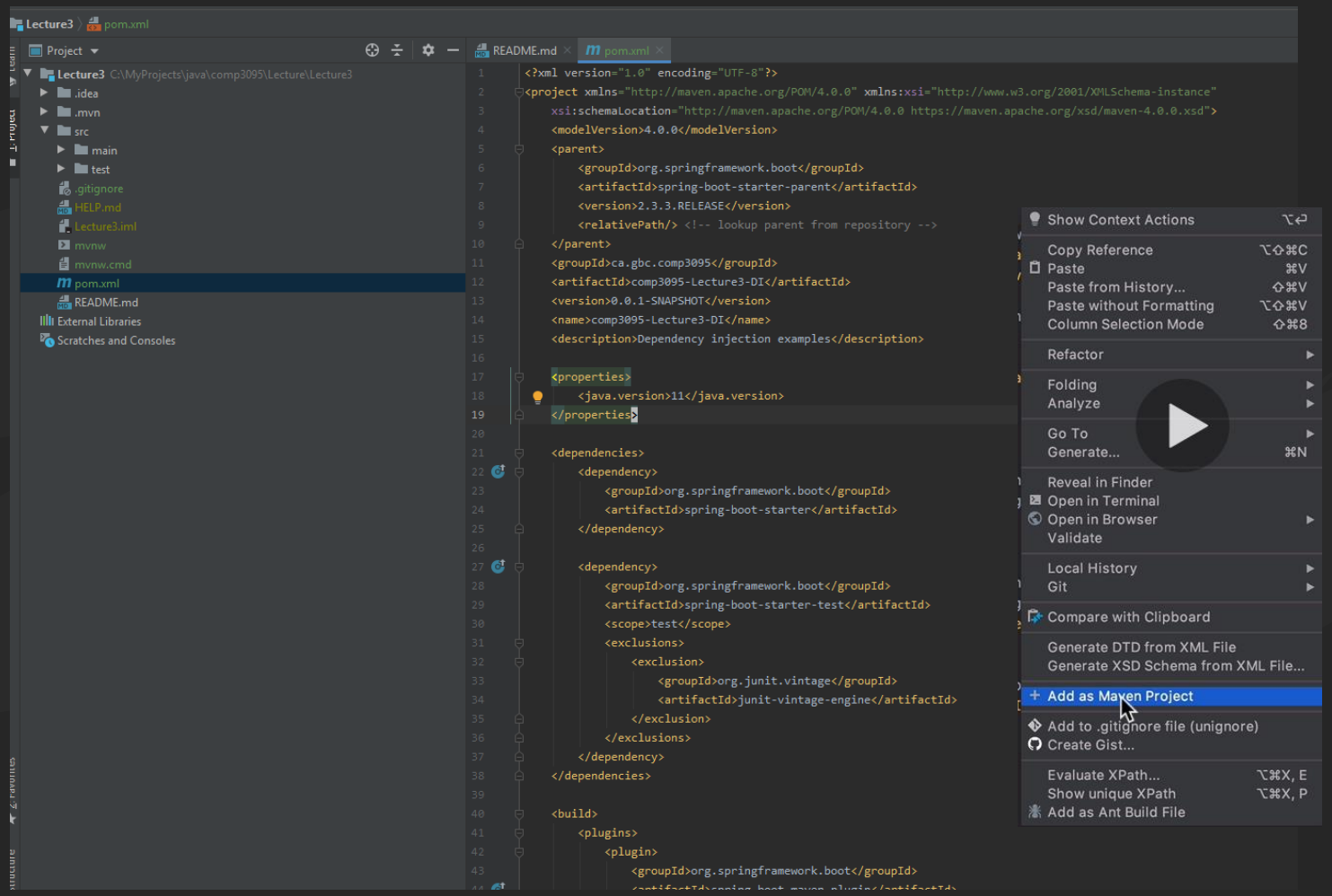
Dependencies ADD DEPENDENCIES... CTRL + B

No dependency selected

GENERATE CTRL + ⏎ EXPLORE CTRL + SPACE SHARE...

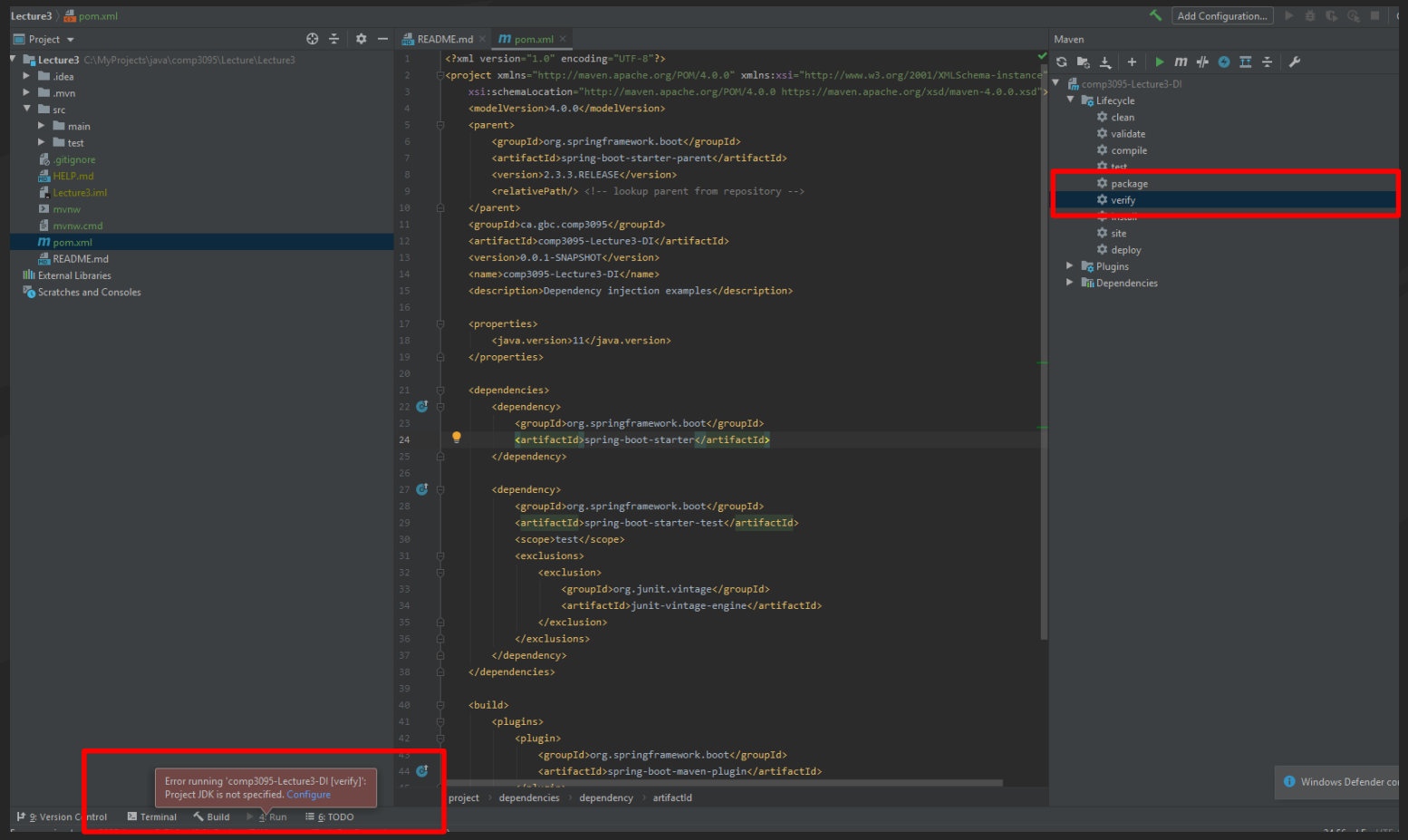
Extract Project and drag & drop into Intellij

- Click on pom.xml file, and "*Add as Maven Project*"



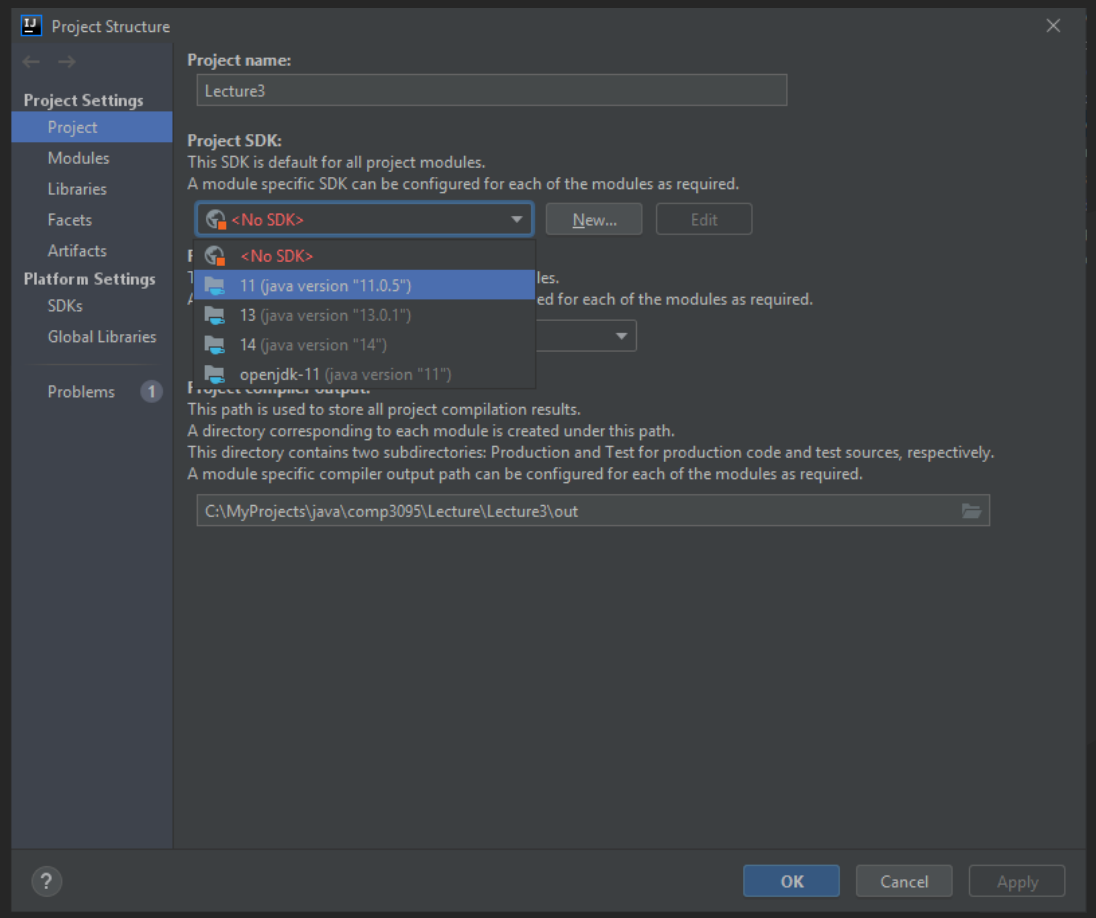
Extract Project and drag & drop into Intellij

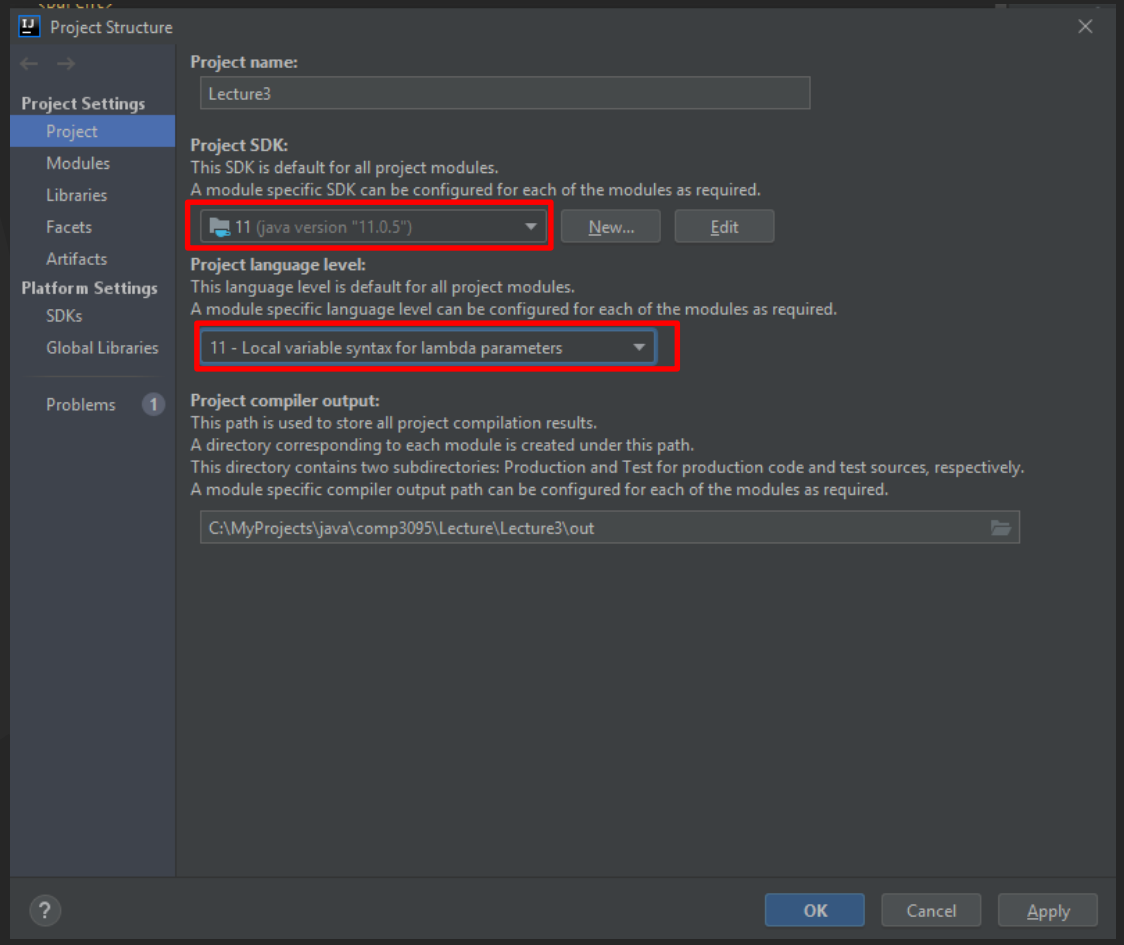
- Maven → Lifecycle → Verify
- Configure JDK (if necessary)



Extract Project and drag & drop into Intellij

- Configure JDK





Extract Project and drag & drop into Intellij

- Maven verify → Should work okay now

Run: comp3095-Lecture3-DI [verify]

```
comp3095-Lecture3-DI [verify]: at 8/17/2020 2:22 PM                    23 s 79 ms
  dependencies
    org.apiguardian:apiguardian-api:jar:1.0.0
  dependencies
    org.junit.platform:junit-platform-engine:jar:1.3.1                 730 ms
  dependencies
    org.junit.platform:junit-platform-launcher:jar:1.3.1               643 ms
  dependencies
    org.junit.platform:junit-platform-commons:jar:1.3.1                593 ms
  dependencies
    org.apache.maven.surefire:surefire-junit-platform:jar:2.22.2       578 ms
```

```
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ comp3095-Lecture3-DI ---
[INFO] Building jar: C:\MyProjects\java\comp3095\Lecture\Lecture3\target\comp3095-Lecture3-DI-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.3.3.RELEASE:repackage (repackage) @ comp3095-Lecture3-DI ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  20.052 s
[INFO] Finished at: 2020-08-17T14:22:59-04:00
[INFO] ------------------------------------------------------------------------
```

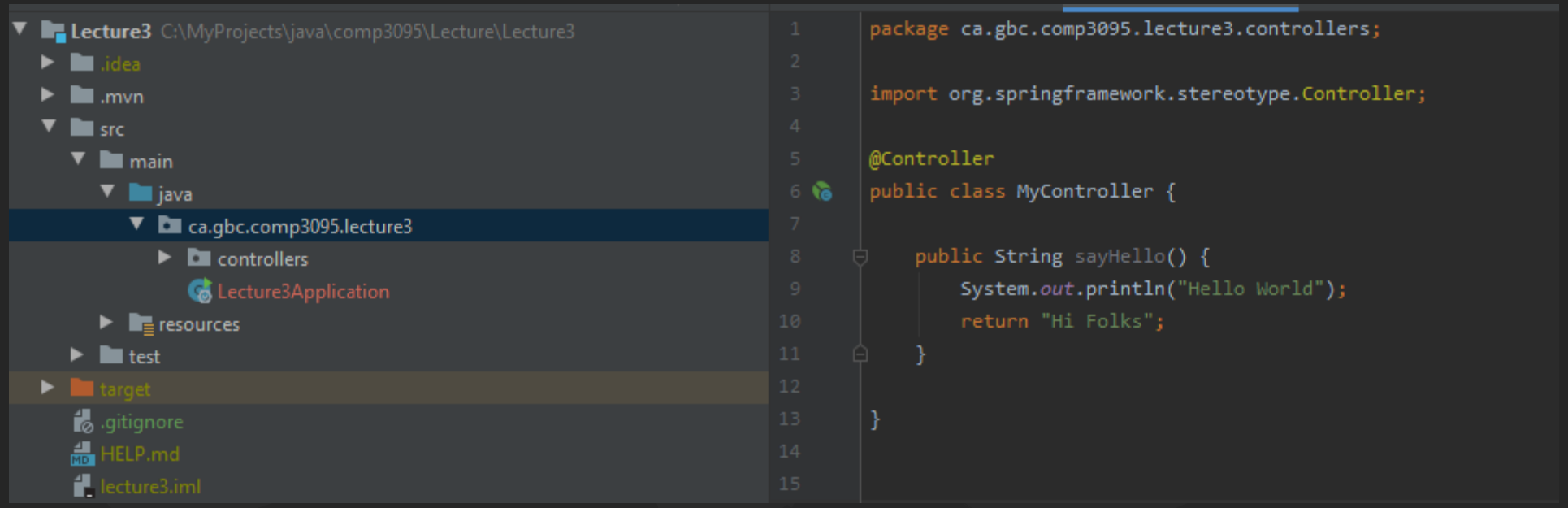
Spring Context

[redacted]

Spring Context

- Spring context are also called spring IoC (Inversion of Control) containers
- Responsible for instantiating, configuring and assembling beans by reading metadata from XML, Java annotations and/or Java Code in configuration files.

Create Controller



Obtaining a Handle on Spring Controller

```
Lecture3 C:\MyProjects\java\comp3095\Lecture\Lecture3
  .idea
  .mvn
  src
    main
      java
        ca.gbc.comp3095.lecture3
          controllers
          Lecture3Application
      resources
    test
  target
  .gitignore
  HELP.md
  lecture3.iml
  mvnw
  mvnw.cmd
  pom.xml
  README.md
External Libraries
Scratches and Consoles
```

```java
1  package ca.gbc.comp3095.lecture3;
2
3  import ...
7
8  @SpringBootApplication
9  public class Lecture3Application {
10
11     public static void main(String[] args) {
12
13         ApplicationContext ctx = SpringApplication.run(Lecture3Application.class, args);
14
15         MyController myController = (MyController) ctx.getBean( name: "myController");
16
17         String greeting = myController.sayHello();
18
19         System.out.println(greeting);
20     }
21
22 }
23
```

```
Run: Lecture3Application
Console   Endpoints
2020-08-17 17:29:59.535  INFO 14652 --- [          main] c.g.c.lecture3.Lecture3Application       : Started Lecture3Application in 1.36 seconds (JVM running for 3.735)
Hello World
Hi Folks
```

This program effectively starts and stop since there is no web context to this application (runs like a standalone application)

Notice in the source we never explicitly instantiate a new controller (using “new”) … rather the Spring Context Framework has actually done this for us already.

Spring is managing and constructing the creation of the beans / controllers.

Basics of Dependency Injection

[redacted]

Dependency Injection (DI)

- Design pattern used to implement IoC.
- It allows the creation of dependent objects of a class and provides those objects to a class through various ways.
- Where a needed dependency is injected by another object
- The class being injected has no responsibility in instantiating the object being injected.
- We move the creation and binding of the dependent objects outside of the class that depends on them.

Types of Dependency Injection

1. Property Injection
   - Least preferred
   - Can be public or private properties
   - Using private is discouraged / BAD practice

2. Setters Injection
   - Area of some debate

3. Constructor Injection
   - Most preferred

Concrete Classes vs Interfaces

- DI can be done with Concrete Classes or with Interfaces
- Generally DI with Concrete should be avoided
- DI with interfaces is highly preferred
  - Allows runtime to decide implementation to inject
  - Follows interface Segregation Principle of SOLID
  - Also makes your code more testable

Inversion of Control (IoC)

- Is a technique to allow dependencies to be injected at runtime
- Dependencies are not predetermined
- Methods defined by the user will often be called by the framework itself, rather than the users application code.
- The framework often plays the role of the main program, coordinating and sequencing the application activity.

IoC vs Dependency Injection

- IoC and Di are easily confused
- DI refers much to the composition of your classes
  - i.e. → You compose your classes with DI in mind
- IoC is the runtime environment of your code
  - i.e → Spring Frameworks IoC container
  - Spring is in control of the injection of dependencies

Best Practices with Dependency Injection

1. Favor using Constructor Injection over Setter Injection
2. Use final properties for injected components
3. Whenever practical, code to an interface

Dependency Injection without Spring

[redacted]

Adding a Service

- Add GreetingService Interface and Implementation

Project

Lecture3 C:\MyProjects\java\comp3095\Lecture\Lecture3
- .idea
- .mvn
- src
  - main
    - java
      - ca.gbc.comp3095.lecture3
        - controllers
          - MyController
        - services
          - GreetingService
          - GreetingServiceImpl
        - Lecture3Application

Lecture3Application.java | MyController.java | GreetingService.java | GreetingServiceImpl.java

```java
package ca.gbc.comp3095.lecture3.services;

public interface GreetingService {


    String sayGreeting();


}
```

Project

Lecture3 C:\MyProjects\java\comp3095\Lecture\Lecture3
- .idea
- .mvn
- src
  - main
    - java
      - ca.gbc.comp3095.lecture3
        - controllers
          - MyController
        - services
          - GreetingService
          - GreetingServiceImpl
        - Lecture3Application

Lecture3Application.java | MyController.java | GreetingService.java | GreetingServiceImpl.java

```java
package ca.gbc.comp3095.lecture3.services;

public class GreetingServiceImpl implements GreetingService {

    @Override
    public String sayGreeting() {
        return "Hello World";
    }

}
```

Property Injection

[redacted]

Property Injected Controller

- Add New Controller

Lecture3 > src > test

Project

- Lecture3 C:\MyProjects\java\comp3095\Lecture\Lecture3
  - .idea
  - .mvn
  - src
    - main
      - java
        - ca.gbc.comp3095.lecture3
          - controllers
            - MyController
            - PropertyInjectedController
          - services
            - GreetingService
            - GreetingServiceImpl
          - Lecture3Application

Lecture3Application.java | MyController.java | GreetingService.java | GreetingServiceImpl.java | PropertyInjectedController.java

```java
package ca.gbc.comp3095.lecture3.controllers;

import ca.gbc.comp3095.lecture3.services.GreetingService;

public class PropertyInjectedController {

    public GreetingService greetingService;

    public String getGreeting(){
        return greetingService.sayGreeting();
    }


}
```

Create Junit Test

- Add Junit Test

Create Test

Testing library: JUnit5

Class name: PropertyInjectedControllerTest

Superclass:

Destination package: ca.gbc.comp3095.lecture3.controllers

Generate: setUp/@Before
tearDown/@After

Generate test methods for: Show inherited methods

Member
getGreeting():String

OK Cancel

Project

Lecture3 C:\MyProjects\java\comp3095\Lecture\Lecture3
- .idea
- .mvn
- src
  - main
    - java
      - ca.gbc.comp3095.lecture3
        - controllers
          - MyController
          - PropertyInjectedController
        - services
          - GreetingService
          - GreetingServiceImpl
        - Lecture3Application
    - resources
  - test
    - java
      - ca.gbc.comp3095.lecture3
        - controllers
          - PropertyInjectedControllerTest

PropertyInjectedControllerTest.java | PropertyInjectedController.java

```java
package ca.gbc.comp3095.lecture3.controllers;

import ...

class PropertyInjectedControllerTest {

    PropertyInjectedController controller;

    //perform this before each unit test
    @BeforeEach
    void setUp(){
        controller = new PropertyInjectedController();  // mimicking what spring context would be doing for us.
        controller.greetingService = new GreetingServiceImpl();
    }

    @Test
    void getGreeting() {
        System.out.println(controller.getGreeting());
    }
}
```

Run: PropertyInjectedControllerTest.getGreeting

Tests passed: 1 of 1 test – 38 ms

Test Results	38 ms
PropertyInjectedControllerTest	38 ms
getGreeting()	38 ms

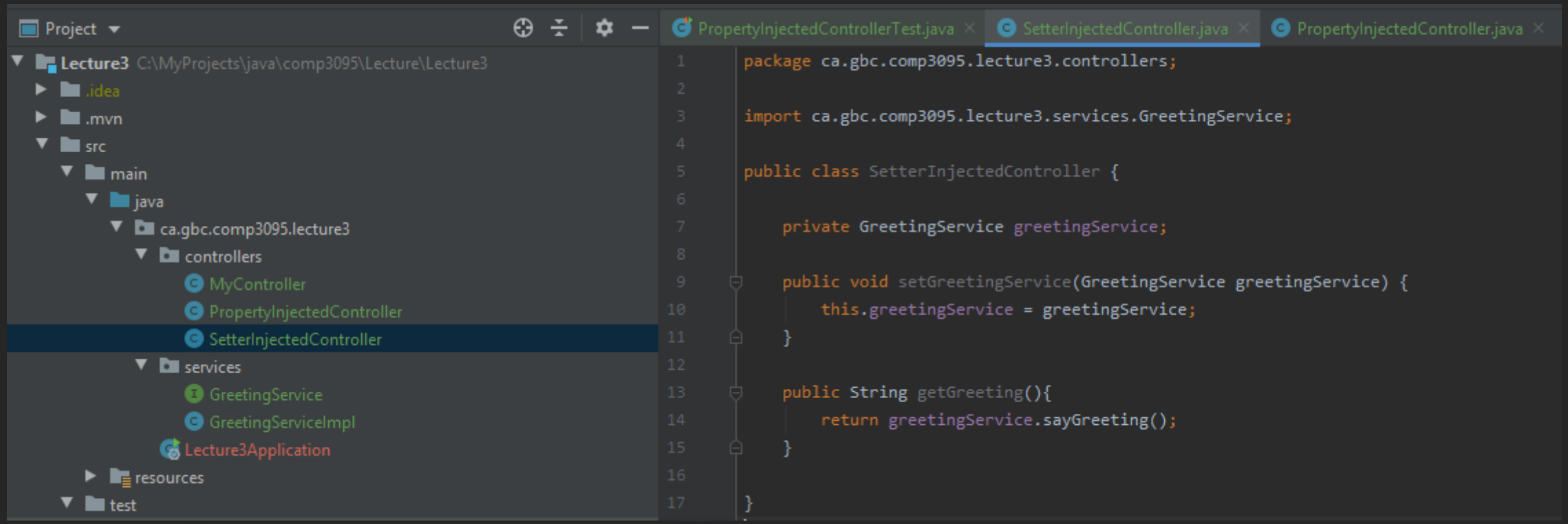
```
"C:\Program Files\Java\OpenJDK\jdk-11\bin\java.exe" ...

Hello World


Process finished with exit code 0
```

Setter Injection

[redacted]

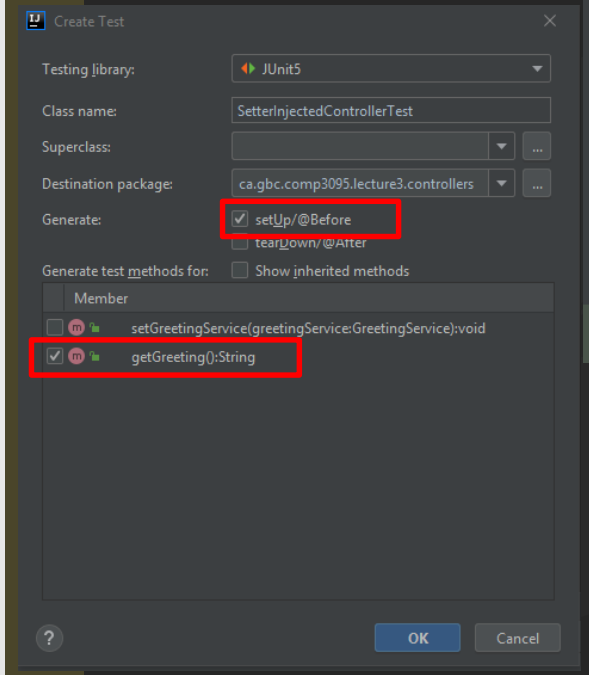
Setter Injected Controller

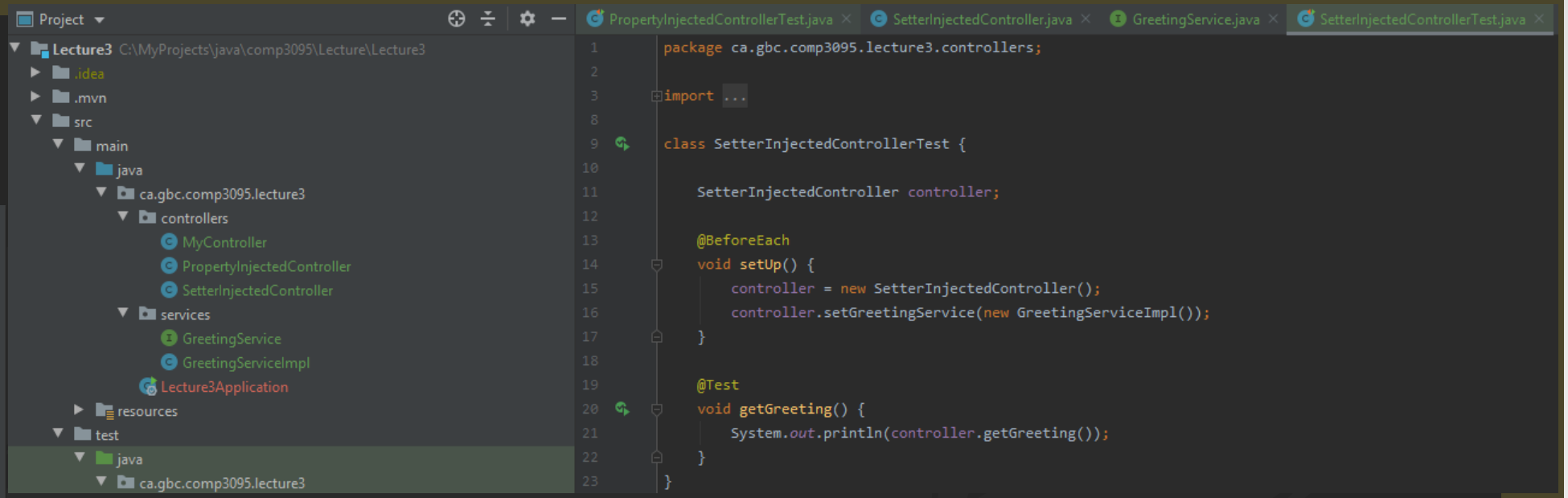
- Add New Controller

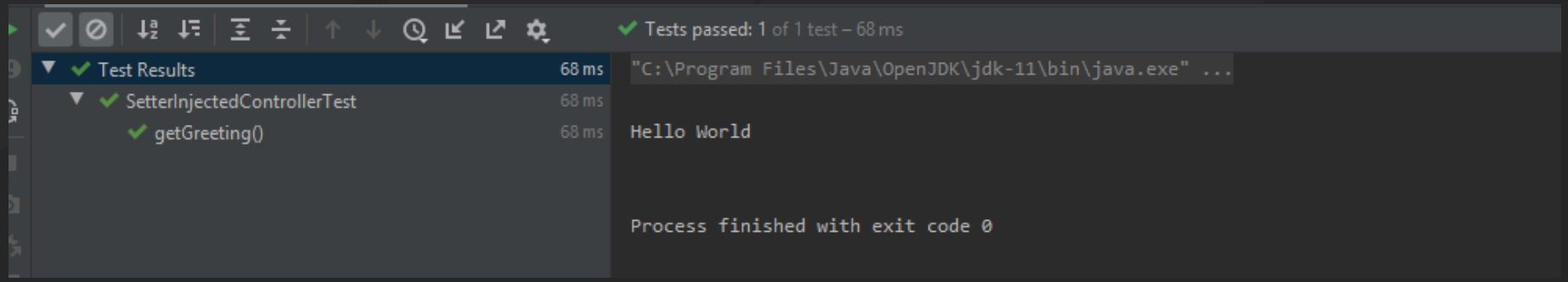


Create Junit Test

- Add Junit Test





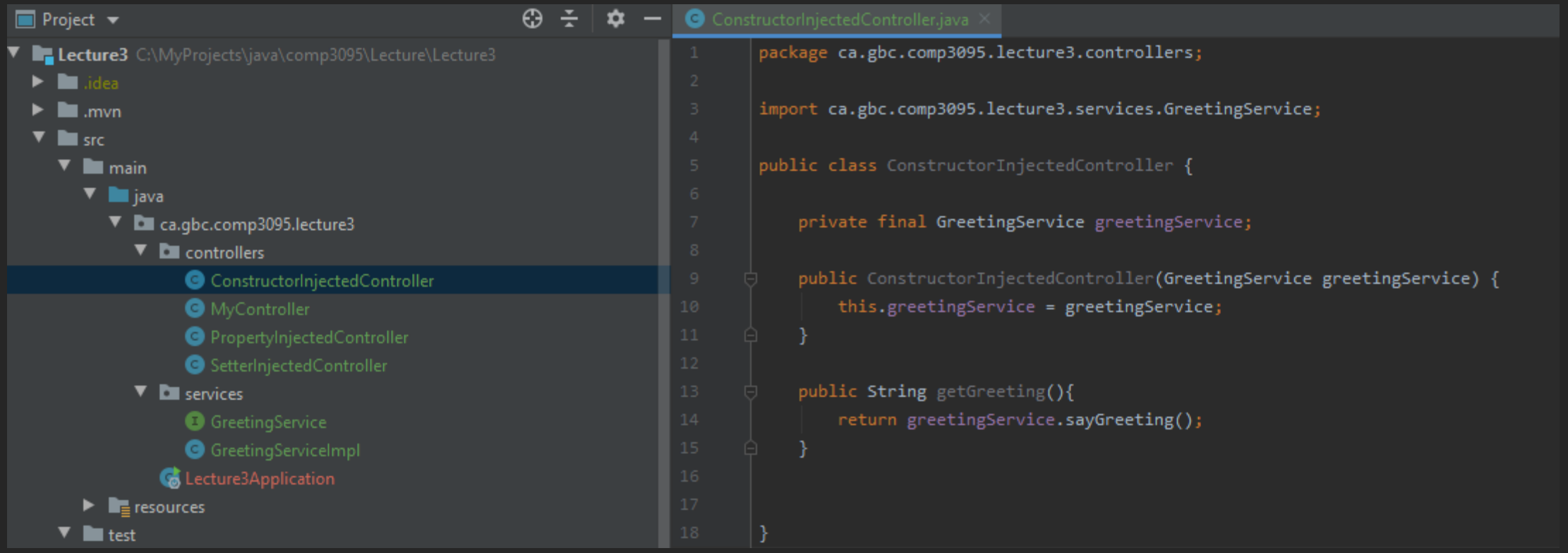


Constructor Injection

[redacted]

Constructor Injected Controller

- Add New Controller



Create Junit Test

- Add Junit Test

Create Test

Testing library: JUnit5

Class name: ConstructorInjectedControllerTest

Superclass:

Destination package: ca.gbc.comp3095.lecture3.controllers

Generate: setUp/@Before

tearDown/@After

Generate test methods for: Show inherited methods

Member

getGreeting():String

OK Cancel

Project

Lecture3 C:\MyProjects\java\comp3095\Lecture\Lecture3

.idea

.mvn

src

main

java

ca.gbc.comp3095.lecture3

controllers

ConstructorInjectedController

MyController

PropertyInjectedController

SetterInjectedController

services

GreetingService

GreetingServiceImpl

Lecture3Application

resources

test

ConstructorInjectedController.java | ConstructorInjectedControllerTest.java

```
package ca.gbc.comp3095.lecture3.controllers;

import ...

class ConstructorInjectedControllerTest {

    ConstructorInjectedController controller;

    @BeforeEach
    void setUp() {
        controller = new ConstructorInjectedController(new GreetingServiceImpl());
    }

    @Test
    void getGreeting() {
        System.out.println(controller.getGreeting());
    }
}
```

Run: ConstructorInjectedControllerTest.getGreeting

Tests passed: 1 of 1 test – 47 ms

Test Results 47 ms

ConstructorInjectedControllerTest 47 ms

getGreeting() 47 ms

```
"C:\Program Files\Java\OpenJDK\jdk-11\bin\java.exe" ...

Hello World

Process finished with exit code 0
```

Questions?

[redacted]