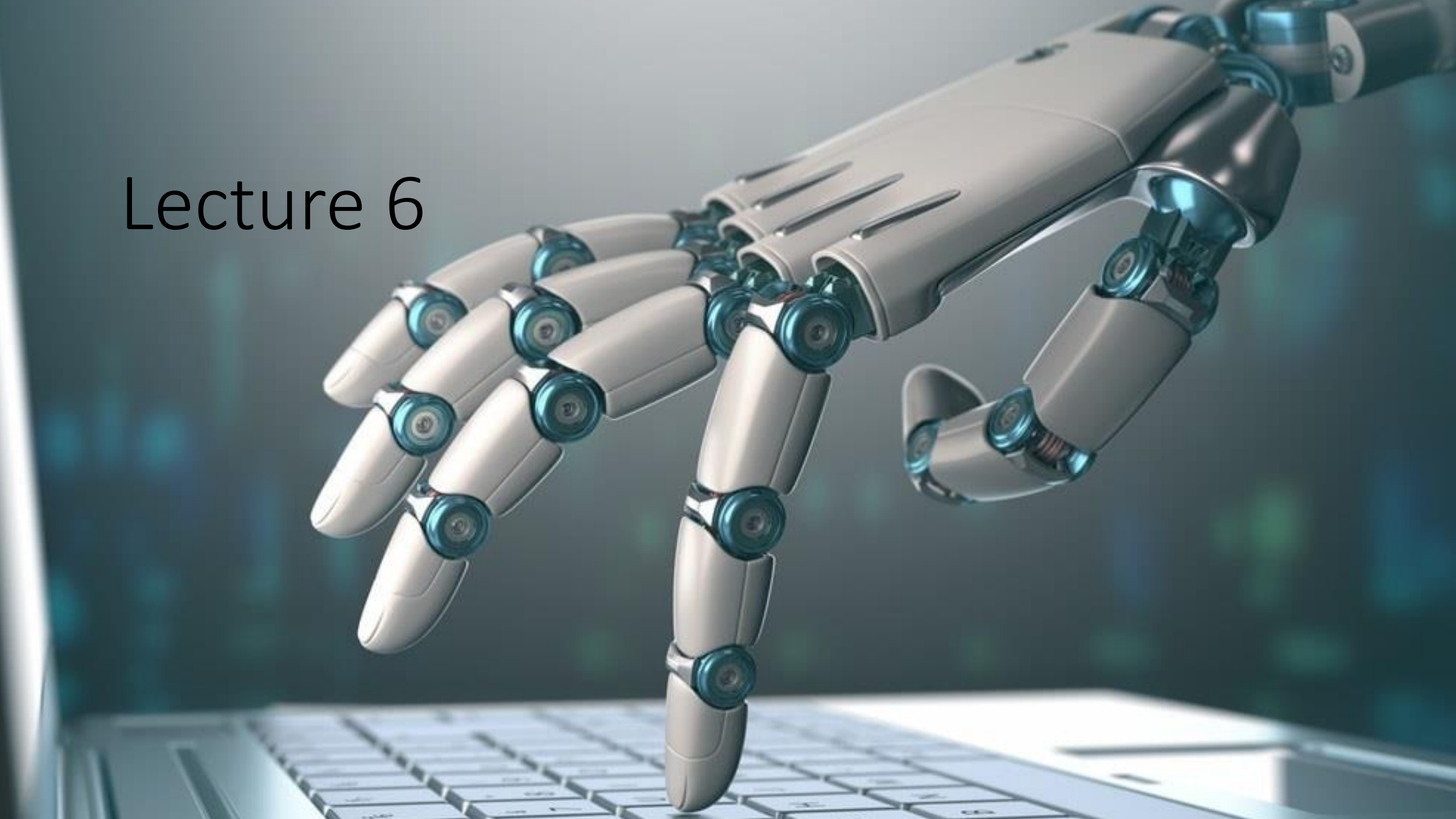
Lecture 6



Agenda

- **Maven vs Gradle**
- Annotations
- Junit
- Assertj
- Practice
- Bonus - Lombok

How to build and compile our project?

Maven

Maven

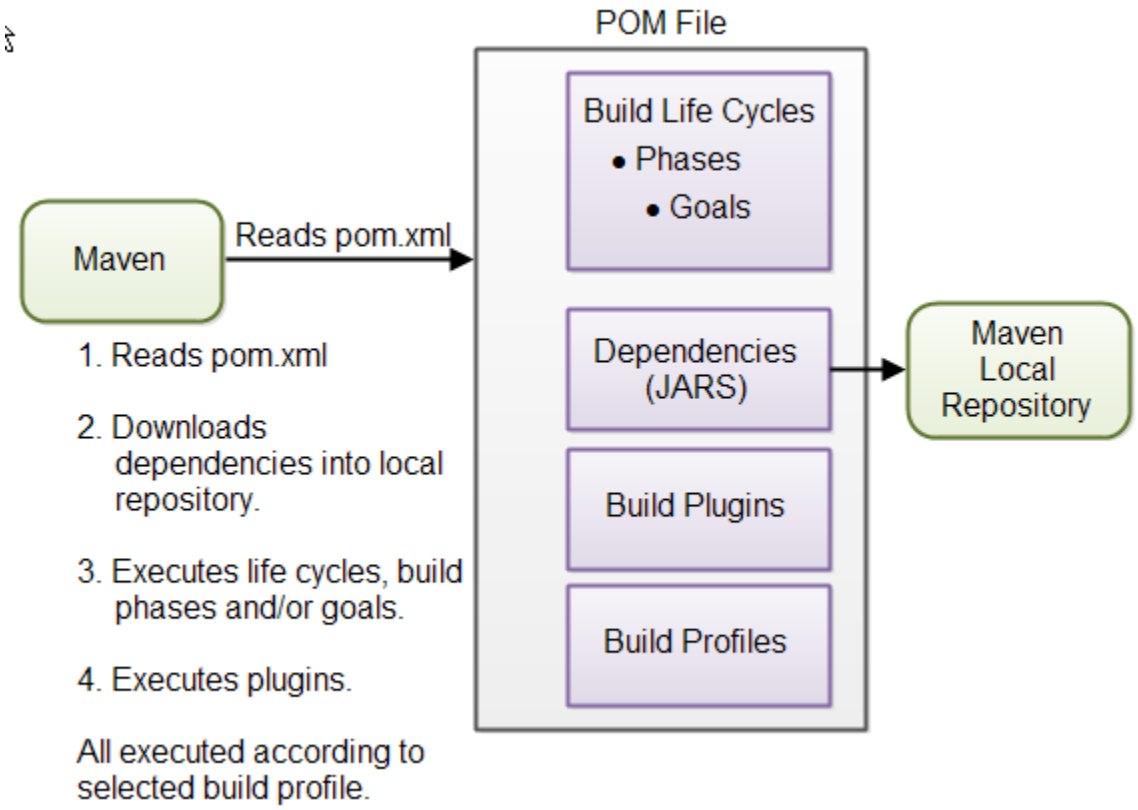
Maven is used for project build automation using Java. It helps you map out how a particular software is built, as well as its different dependencies. It uses an XML file to describe the project that you are building, the dependencies of the software with regards to third-party modules and parts, the build order, as well as the needed plugins. There are pre-defined targets for tasks such as packaging and compiling.

Maven will download libraries and plugins from the different repositories and then puts them all in a cache on your local machine. While predominantly used for Java projects, you can use it for Scala, Ruby, and C#, as well as a host of other languages.

Maven file example

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.12.1</version>
    <executions>
        <execution>
            <configuration>
                <configLocation>config/checkstyle/checkstyle.xml</configLocation>
                <consoleOutput>true</consoleOutput>
                <failsOnError>true</failsOnError>
            </configuration>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>findbugs-maven-plugin</artifactId>
    <version>2.5.4</version>
    <executions>
        <execution>
            <goals>
                <goal>check</goal>
```

Maven


POM File
Build Life Cycles
• Phases
• Goals
Maven
Reads pom.xml
Dependencies (JARS)
Maven Local Repository
Build Plugins
Build Profiles
1. Reads pom.xml
2. Downloads dependencies into local repository.
3. Executes life cycles, build phases and/or goals.
4. Executes plugins.
All executed according to selected build profile.

Gradle

Gradle

Gradle is a build automation system that is fully open source and uses the concepts you see on Apache Maven and Apache Ant. It uses domain-specific language based on the programming language Groovy, differentiating it from Apache Maven, which uses XML for its project configuration. It also determines the order of tasks run by using a directed acyclic graph.

Why Gradle

Flexibility
Full control
Chaining of targets

Dependency management

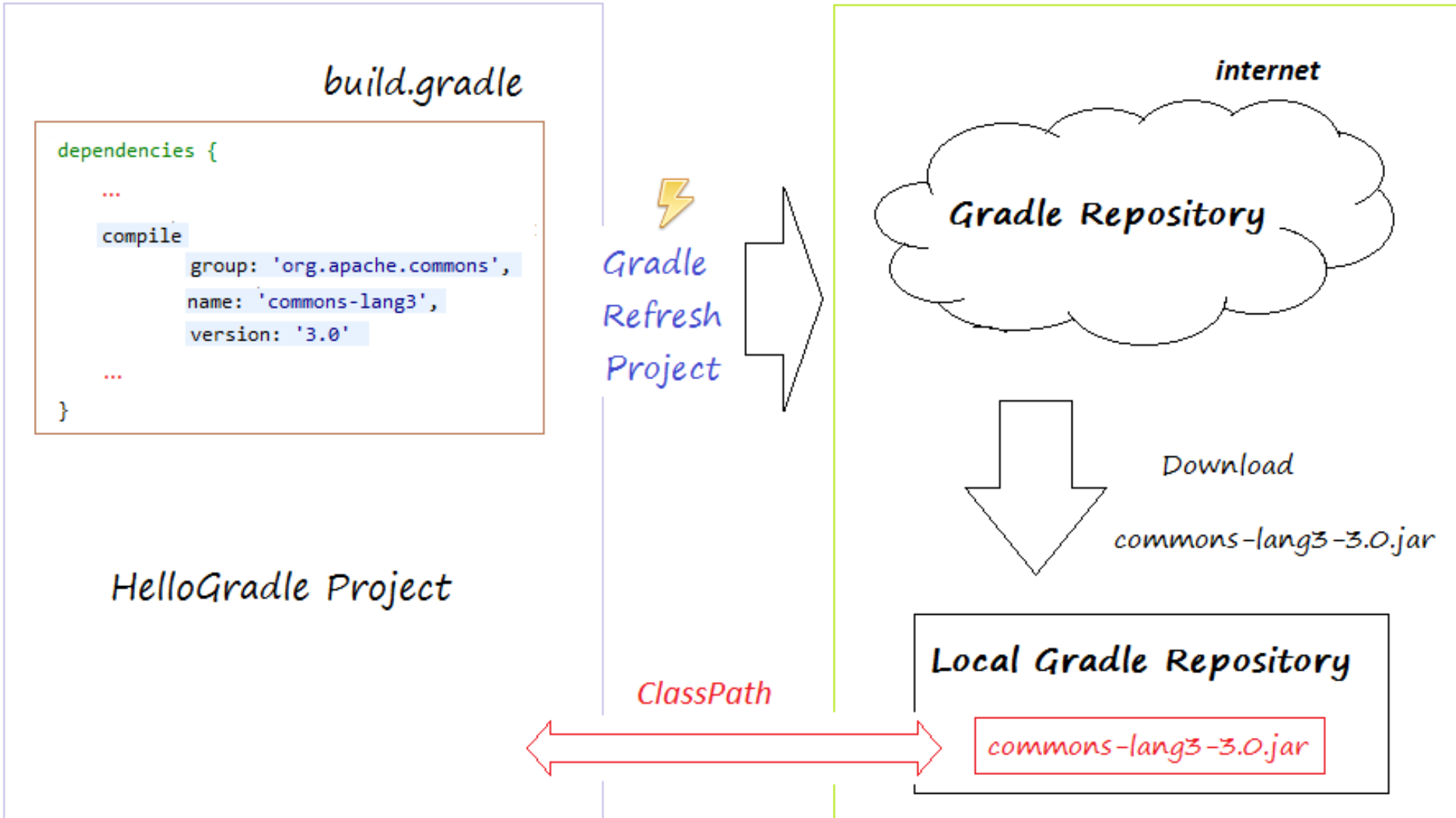
Convention over configuration
Multimodule projects
Extensibility via plugins

Groovy DSL on top of Ant

Gradle file example

```
1  apply plugin:'java'
2  apply plugin:'checkstyle'
3  apply plugin:'findbugs'
4  apply plugin:'pmd'
5
6  version ='1.0'
7
8  repositories {
9      mavenCentral()
10 }
11
12 dependencies {
13     testCompile group:'junit', name:'junit', version:'4.11'
14 }
```

How build.gradle works



Comparison Gradle vs Maven

```
complie group: 'log4j', name: 'log4j', version: '1.2.17'
```

Для сравнения аналогичный блок в maven:

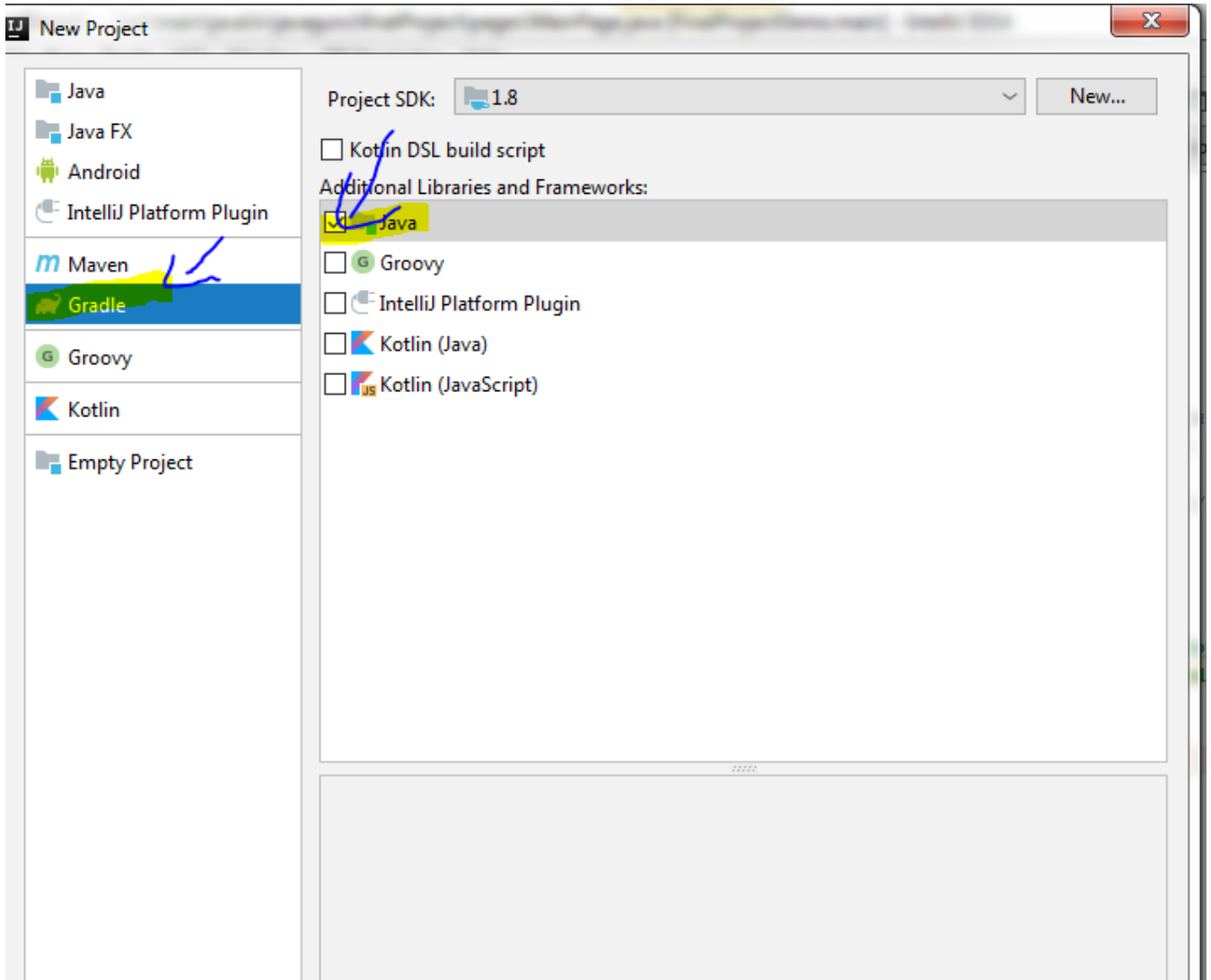
```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

Gradle Project structure

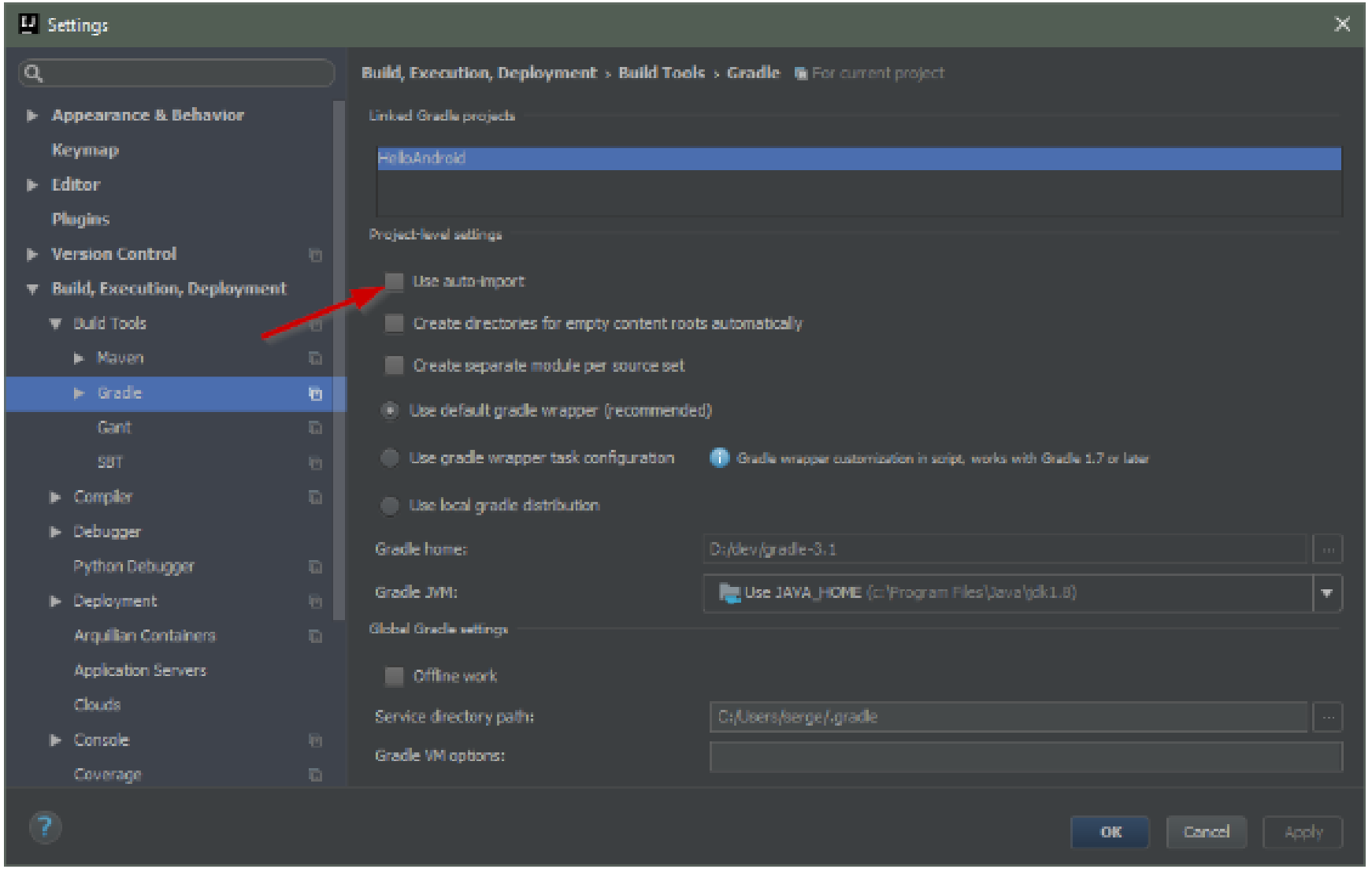
```
/project
    /src
        /main
            /java
            /resources
        /test
            /java
            /resources
```

How to Create gradle project

- File -> New



How to use Auto IMPORT



Agenda

- Maven vs Gradle
- **Annotations**
- Junit
- Assertj
- Practice
- Bonus - Lombok

Annotations

Annotations in Java

Annotations are used to provide supplement information about a program.

- Annotations start with '@'.
- Annotations do not change action of a compiled program.
- Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.
- Annotations are not pure comments as they can change the way a program is treated by compiler. See below code for example.

Обработка аннотации в Джава

На основе аннотаций компилятор может с помощью специальных обработчиков генерировать новый код и файлы конфигурации.

Обработчиками обычно выступают библиотеки и утилиты, которые можно брать у сторонних авторов (или создавать самостоятельно) и прикреплять к проекту в среде разработки. Способ подключения зависит от IDE или системы сборки. В Maven обработчики подключают с помощью модуля annotation-user или плагина maven-compiler-plugin.

Парсинг аннотаций происходит циклически. Компилятор ищет их в пользовательском коде и выбирает подходящие обработчики. Если вызванный обработчик на основе аннотации создаёт новые файлы с кодом, начинается следующий этап, где исходным материалом становится сгенерированный код. Так продолжается до тех пор, пока не будут созданы все необходимые файлы.

Examples of annotation

- @Test
- @Getter
- @Data

Example of Annotation Inside

```java
/**
 * Put on any field to make lombok build a standard setter.
 * <p>
 * Complete documentation is found at <a href="https://projectlombok.org/features/GetterSetter.html">the project lombok fea
 * <p>
 * Even though it is not listed, this annotation also has the {@code onParam} and {@code onMethod} parameter. See the full
 * <p>
 * Example:
 * <pre>
 *     private &#64;Setter int foo;
 * </pre>
 *
 * will generate:
 *
 * <pre>
 *     public void setFoo(int foo) {
 *         this.foo = foo;
 *     }
 * </pre>
 *
 * <p>
 * This annotation can also be applied to a class, in which case it'll be as if all non-static fields that don't already ha
 * a {@code Setter} annotation have the annotation.
 */
@Target({ElementType.FIELD, ElementType.TYPE})
@Retention(RetentionPolicy.SOURCE)
public @interface Setter {
    /**
     * If you want your setter to be non-public, you can specify an alternate access level here.
     */
    lombok.AccessLevel value() default lombok.AccessLevel.PUBLIC;
```

Agenda

- Maven vs Gradle
- Annotations
- **Junit**
- Assertj
- Practice
- Bonus - Lombok

What is Junit

- JUnit is a simple, open source framework to write and run repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. JUnit features include:
- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test runners for running tests
- JUnit was originally written by Erich Gamma and Kent Beck

Junit annotations

1. @BeforeClass – Run once before any of the test methods in the class, `public static void`
2. @AfterClass – Run once after all the tests in the class have been run, `public static void`
3. @Before – Run before @Test, `public void`
4. @After – Run after @Test, `public void`
5. @Test – This is the test method to run, `public void`

@Test example

```
@Test
public void componentsCorrespondToFactoryArguments() {
  String[] components = { "Test1", "Test2", "Test3" };
  MyPath myPath = MyPath.ofComponents(components);
  assertThat(myPath.getComponents(), contains(components));
}
```

Is this a test?

```java
import static org.assertj.core.api.Assertions.assertThat;

public class Test {

    public void TestingTest() {
        Crocodile crocodile = new Crocodile();
        crocodile.setColor("green");
        String crocodileColore = crocodile.getColor();
        System.out.println(crocodileColore);
        CarModel carmodel = new CarModel();
        carmodel.setColor("black");
//          carmodel.getColor();

        CarModelWithLombok carModelWithLombok = new CarModelWithLombok();

        carModelWithLombok.setColor("green");
        String someLombokColor = carModelWithLombok.getColor();
//          carModelWithLombok.setColor("black");
        assertThat(someLombokColor).isEqualTo(123);
    }
```

Is this a test?

```java
import org.junit.Test;

import static org.assertj.core.api.Assertions.assertThat;

public class MyTest {

    @Test
    public void TestingTest() {
        Crocodile crocodile = new Crocodile();
        crocodile.setColor("green");
        String crocodileColore = crocodile.getColor();
        System.out.println(crocodileColore);
        CarModel carmodel = new CarModel();
        carmodel.setColor("black");
//          carmodel.getColor();

        CarModelWithLombok carModelWithLombok = new CarModelWithLombok();

        carModelWithLombok.setColor("green");
        String someLombokColor = carModelWithLombok.getColor();
//          carModelWithLombok.setColor("black");
        assertThat(someLombokColor).isEqualTo(123);
    }
```

@Before and @After

@Before	This annotation is used if you want to execute some statement such as preconditions before each test case.
@BeforeClass	This annotation is used if you want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.
@After	This annotation can be used if you want to execute some statements after each Test Case for e.g resetting variables, deleting temporary files ,variables, etc.
@AfterClass	This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.

@Before and @After how used?

- Call **@Before setUp**
- Call **@Test** method **test1**
- Call **@After tearDown**
- Call **@Before setUp**
- Call **@Test** method **test2**
- Call **@After tearDown**

@Before and @After Example

```
public class Test extends AbstractBaseTest {

  @Before
  public void setUp() {
    System.out.println("Test.setUp");
  }

  @After
  public void tearDown() {
    System.out.println("Test.tearDown");
  }

  @Test
  public void test1() throws Exception {
    System.out.println("test1");
  }

  @Test
  public void test2() throws Exception {
    System.out.println("test2");
  }
}
```

Agenda

- Maven vs Gradle
- Annotations
- Junit
- **Assertj**
- Practice
- Bonus - Lombok

Asserts

Assertions (by way of the **assert** keyword) were added in Java 1.4. They are used to verify the correctness of an invariant in the code. They should never be triggered in production code, and are indicative of a bug or misuse of a code path. They can be activated at run-time by way of the `-ea` option on the `java` command, but are not turned on by default.

An example:

```
public Foo acquireFoo(int id) {
  Foo result = null;
  if (id > 50) {
    result = fooService.read(id);
  } else {
    result = new Foo(id);
  }
  assert result != null;

  return result;
}
```

Asserts

MOST COMMON ASSERT STATEMENTS

Assertion	Description
`Assert.assertEquals(expected, actual);` `Assert.assertNotEquals(expected, actual);`	This method is executed before each test
`Assert.assertTrue(actual);` `Assert.assertFalse(actual);`	This method is executed after each test
`Assert.assertNull(actual);` `Assert.assertNotNull(actual);`	The following static method is executed once, before the start of all tests
`Assert.assertSame(expected, actual);` `Assert.assertNotSame(expected, actual);`	The following static method is executed once after all tests have been completed

Different AssertThat Libraries

Simple Assertions

```
assertThat(a,  equalTo(b));  //Hamcrest
assertThat(a).isEqualTo(b);  //AssertJ
```

Dates Assertions

```
assertThat(tomorrow,  isAfter(today));  //Hamcrest
assertThat(tomorrow).isAfter(today);  //AssertJ
```

AssertJ

Using AssertJ improves the readability of your tests. While you may be familiar with JUnits assertions it is easy for a beginner to mix up `actual` and `expected`.

```
assertEquals(actual, expected);
assertThat(actual).isEqualTo(expected);
// this look obviously wrong
assertThat(expected).isEqualTo(actual);
```

It makes it easier to express your intent.

```
Date today = new Date();
assertTrue((birthday.getTime() > today.getTime()));
assertThat(birthday).isBefore(today);
```

More Readable Error messages

Example of a failing assertion with a description :

```
TolkienCharacter frodo = new TolkienCharacter("Frodo", 33, HOBBIT);
// failing assertion, remember to call as() before the assertion, not after !
assertThat(frodo.getAge()).as("check %s's age", frodo.getName()).isEqualTo(100);
```

The error message starts with the given description in [] :

```
[check Frodo's age] expected:<100> but was:<33>
```

Soft Assertions

```
@Test
public void host_dinner_party_where_nobody_dies() {
  Mansion mansion = new Mansion();
  mansion.hostPotentiallyMurderousDinnerParty();
  SoftAssertions softly = new SoftAssertions();
  softly.assertThat(mansion.guests()).as("Living Guests").isEqualTo(7);
  softly.assertThat(mansion.kitchen()).as("Kitchen").isEqualTo("clean");
  softly.assertThat(mansion.library()).as("Library").isEqualTo("clean");
  softly.assertThat(mansion.revolverAmmo()).as("Revolver Ammo").isEqualTo(6);
  softly.assertThat(mansion.candlestick()).as("Candlestick").isEqualTo("pristine");
  softly.assertThat(mansion.colonel()).as("Colonel").isEqualTo("well kempt");
  softly.assertThat(mansion.professor()).as("Professor").isEqualTo("well kempt");
  softly.assertAll();
}
```

Now upon running the test our JUnit exception message is far more detailed:

```
org.assertj.core.api.SoftAssertionError: The following 4 assertions failed:
1) [Living Guests] expected:<[7]> but was:<[6]>
2) [Library] expected:<'[clean]'> but was:<'[messy]'>
3) [Candlestick] expected:<'[pristine]'> but was:<'[bent]'>
4) [Professor] expected:<'[well kempt]'> but was:<'[bloodied and disheveled]'>
```

Import Library

- testCompile group: 'org.assertj', name: 'assertj-core', version: '3.14.0'

- <dependency>
- <groupId>org.assertj</groupId>
- <artifactId>assertj-core</artifactId>
- <version>3.14.0</version>
- <scope>test</scope>
- </dependency>

Agenda

- Maven vs Gradle
- Annotations
- Junit
- Assertj
- **Practice**
- Bonus - Lombok

Task 1

- Create a class with Movie
- Create 5 variables for class movie – Movie Name, Director, Genre, Year, Box Office, Budget

Task 2

- Create a class TestMovie
- Create a new Movie object
- Set movie - Rocketman, Dexter Fletcher, Musical/Drama, 2019, 195 000 000, 41 000 000

Task 3

- Edit TestMovie class method – Add annotation @Test
- E.g. AssertThat(movie.getName()).isEqualTo(“Rocketman”);
- Etc.

Task 4

- Create Testmovie2 class
- Create variable movie with type Movie
- Create @Before with public void setUp() {}
- In this method setUp() create object movie and Set movie - Rocketman, Dexter Fletcher, Musical/Drama, 2019, 195 000 000, 41 000 000
- Create @Test where you assert all parameters E.g. AssertThat(movie.getName()).isEqualTo(“Rocketman”);
- Create @After public void tearDown() { } – how we can teardown movie?

Task 5

- Create class Crocodile – with getters and setters – size, color, length, name, hungry (true or false)
- Setup
- Create @Test with @Before and @After – where you create and setup crocodile object
- Assert all parameters through softAssertions in @Test

Task 6 – Additional Stuff we can build

- CalculatorTest
- Statistics test
- Fill array

Agenda

- Maven vs Gradle
- Annotations
- Junit
- Assertj
- Practice
- **Bonus - Lombok**

What is Lombok?

Connection with Java

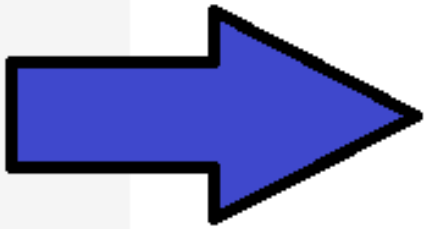
What is Lombok?

Project Lombok is a java library that automatically plugs into your editor and build tools, spicing up your java.
Never write another getter or equals method again

val
Finally! Hassle-free final local variables.
var
Mutably! Hassle-free local variables.
@NonNull
or: How I learned to stop worrying and love the NullPointerException.
@Cleanup
Automatic resource management: Call your close() methods safely with no hassle.
@Getter/@Setter
Never write public int getFoo() {return foo;} again.
@ToString
No need to start a debugger to see your fields: Just let lombok generate a toString for you!
@EqualsAndHashCode
Equality made easy: Generates hashCode and equals implementations from the fields of your object..
@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor
Constructors made to order: Generates constructors that take no arguments, one argument per final / non-nullfield, or one argument for every field.
@Data
All together now: A shortcut for @ToString, @EqualsAndHashCode, @Getter on all fields, and @Setter on all non-final fields, and @RequiredArgsConstructor!
@Value
Immutable classes made very easy.
@Builder
... and Bob's your uncle: No-hassle fancy-pants APIs for object creation!
@SneakyThrows
To boldly throw checked exceptions where no one has thrown them before!
@Synchronized
synchronized done right: Don't expose your locks.
@Getter(lazy=true)
Laziness is a virtue!
@Log
Captain's Log, stardate 24435.7: "What was that line again?"

Why to use it

```
public class Person {

  private String name;

  public String getName() {

    return name;
  }

  public void setName(String name) {

    this.name = name;
  }
}
```



```
import lombok.AccessLevel;
import org.lombok.Getter;
import org.lombok.Setter;

public class Person {

  @Getter @Setter(AccessLevel.PROTECTED)
  private String name;
}
```

How to use it in Java Project – 3 Steps

- Install Lombok Plugin
- Enable annotation process
- Add Lombok dependency to build.gradle file

Step 1 – Install Plugin

IntelliJ IDEA

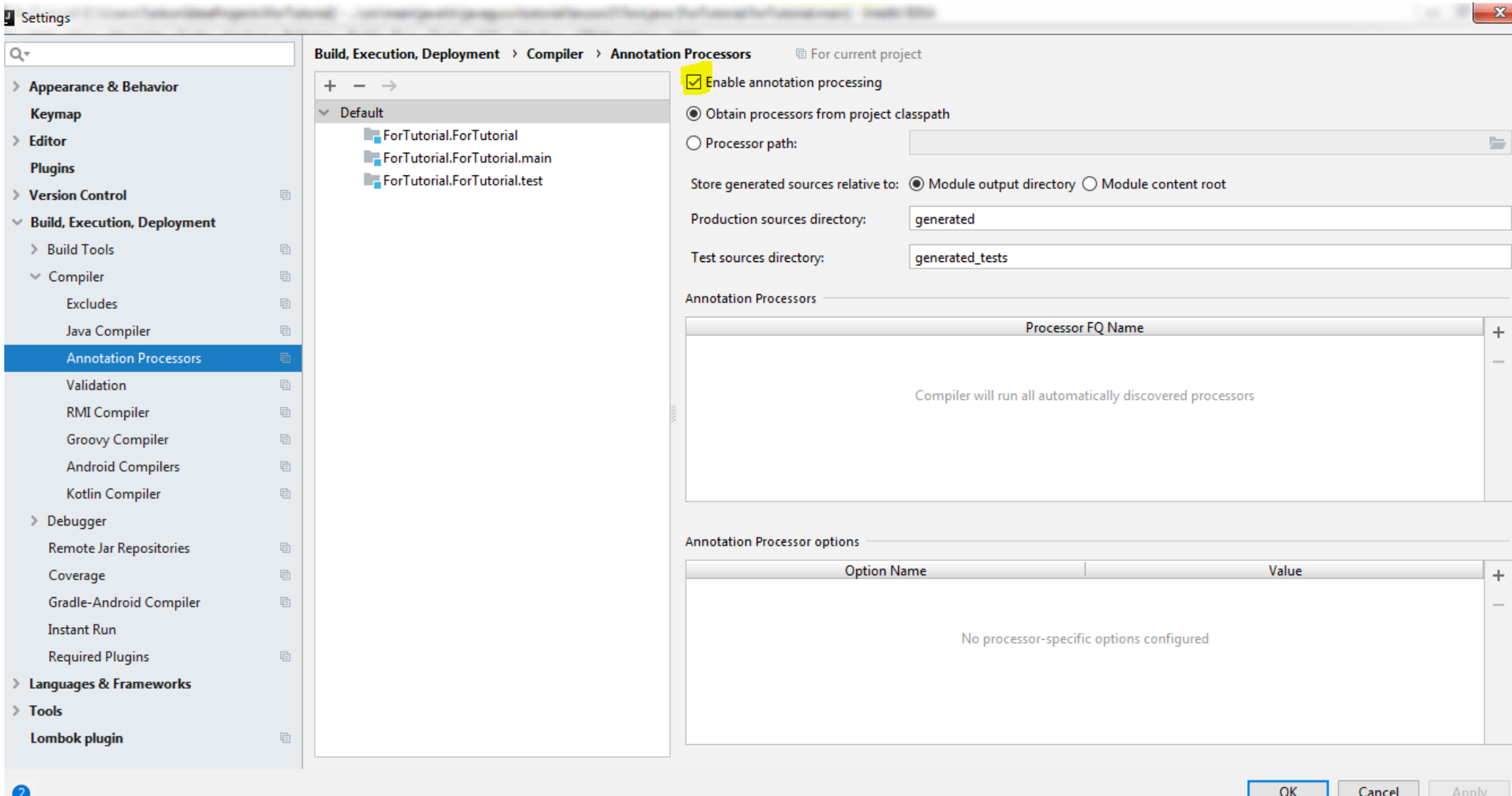
The Jetbrains IntelliJ IDEA editor is compatible with lombok.

Add the Lombok IntelliJ plugin to add lombok support for IntelliJ:

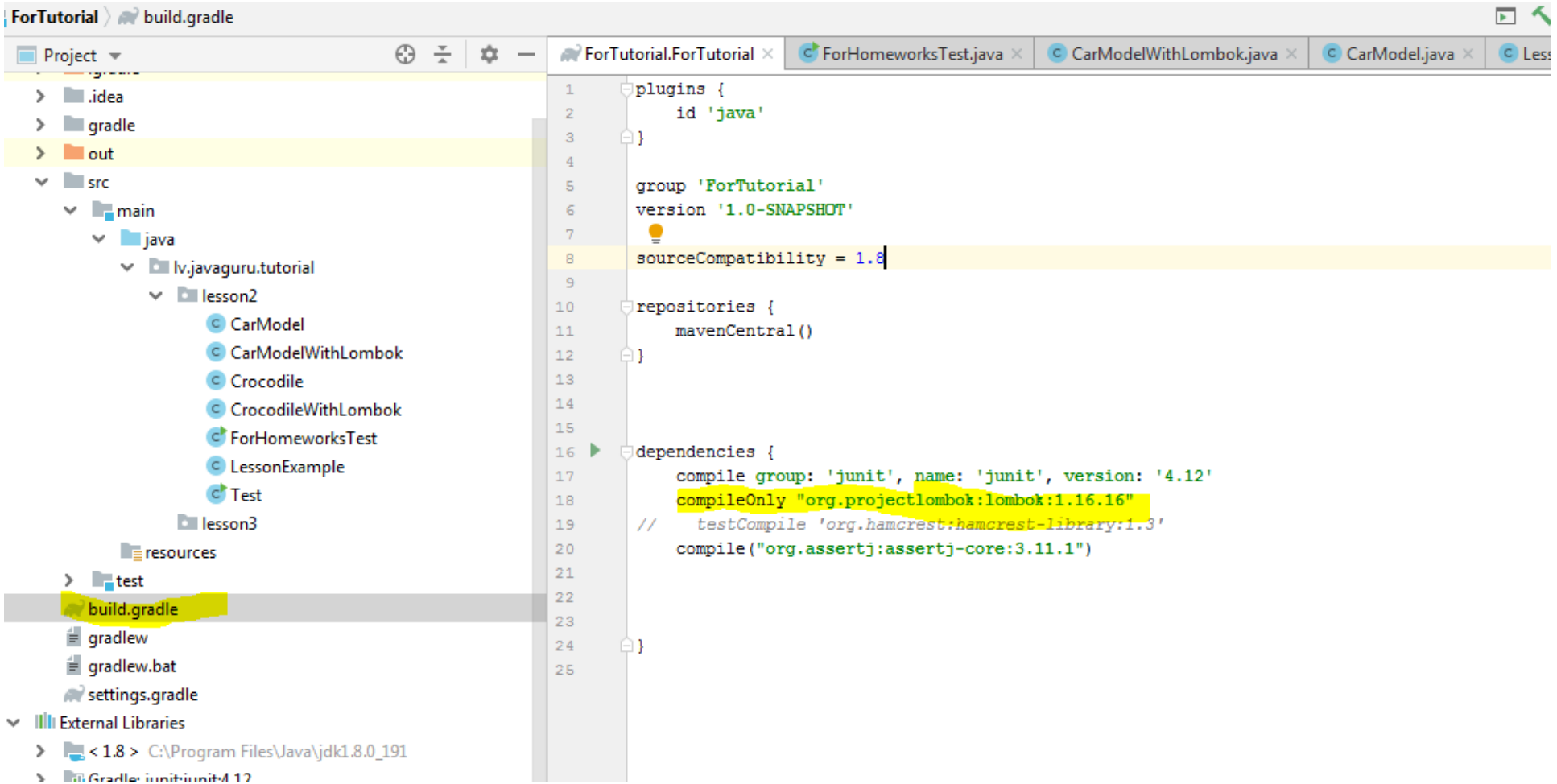
- Go to `File > Settings > Plugins`
- Click on `Browse repositories...`
- Search for `Lombok Plugin`
- Click on `Install plugin`
- Restart IntelliJ IDEA

Step 2 - Enable Annotation Processing

File->Settings -> Build, Execution, Deployment -> Compiler-> Annotation Processors->Enable annotation processing



Step 3 – Add to Build.gradle



Workaround

```
repositories {
        mavenCentral()
}

dependencies {
        compileOnly 'org.projectlombok:lombok:1.18.10'
        annotationProcessor 'org.projectlombok:lombok:1.18.10'
}
```

Bonus Lombok Task

- Copy a movie class and name it movieLombok
- Delete all getters and setters
- All variables must stay
- Before movieLombok class add annotation @Data;
- Copy Testmovie class and name it TestmovieLombok
- Change in TestmovieLombok class - movie to movieLombok
- And try to rerun test

Reference

- Gradle - https://gradle.org/
- Maven - https://maven.apache.org/
- Annotations - https://www.baeldung.com/java-custom-annotation
- JUnit -https://junit.org/junit4/
- Assertj - https://joel-costigliola.github.io/assertj/
- Lombok - https://projectlombok.org/


THANK YOU FOR YOUR ATTENTION
YouTube