

Testing

Testing

- Test (super important)

 - Questions

 - OverView

 - Unit Testing

 - Source code structure

 - Naming Conventions

 - Package Naming Convention

 - Tools

 - Steps of creating Unit Test

 - Dependency

 - Junit

 - Assertions

 - LifeCircle

 - DAO Test

 - Code Coverage

 - IDE-Intellij

 - Maven

 - Mockito

 - Quesitons

 - Mocking Frameworks

 - Mockito Steps

 - Annotations

 - Mockito Methods

 - PowerMockito

 - Integration Test

- SonarQube(Code Quality)

- Conclusion

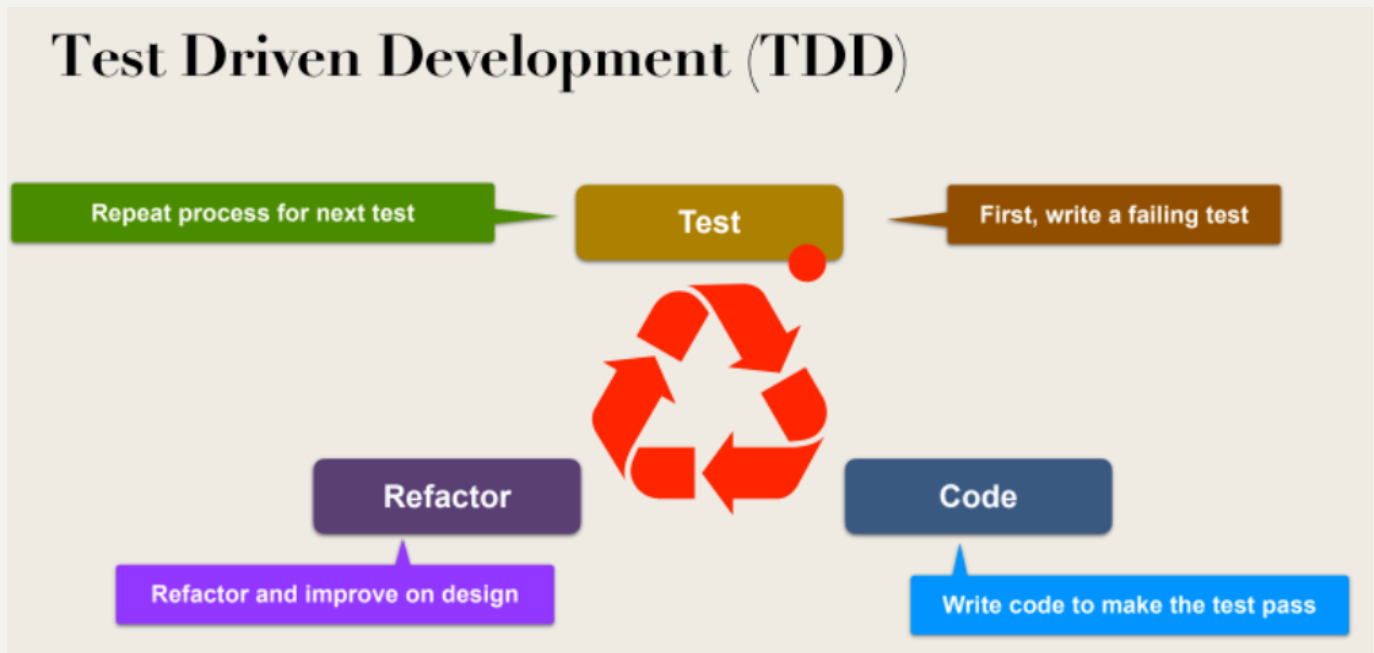
- Jira

 - Agile - Sprint

Test (super important)

Questions

1. What is the TDD?
2. What tools do you use for testing? Junit, Mockito, PowerMockito
3. What is unit-test and what is Integration Test?
4. What is automation?



OverView

General Test in industry

Environment	Test Types	Tools
Local	Peer Test	Human
	Unit Test	Junit, Mockito , PowerMock etc
Dev	Unit Test	Junit etc
	Automation Test	Selenium , TestNG, Cucumber etc
QA	Manual	Human, HP Quality Center
	Automation	Selenium , TestNG, Cucumber etc
	Regression	Automation test
Pre-Prod	Performance/Load Test	JMeter, Gatling
Prod	AB Test Alpha/Beta test (Prod-Log Retrieve)	User

看这张图时候，顺便思考开发流程和不同的环境以及数据库。

Test Suite - Automation

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
    MainControllerMockTest.class,
    UserDetailsServiceImplTest.class,
    UserDetailsServiceImplMockTest.class
})
public class CompositeApplicationTests {

    @Test
    public void contextLoads() {
    }

}
```

Automation whole application from front end

- ▶ Selenium
 - ▶ WebDriver - a driver to help grab web elements
 - ▶ Page Objects - a java class to simulate a web page
- ▶ TestNG (Data-Driven Development)
- ▶ Cucumber

Selenium: User-behavior simulation

Selenium

user-behavior simulation

```
@Test
public void testLogin(){
    driver.get("http://localhost:8080");

    WelcomePage welcomePage = new WelcomePage(driver);
    welcomePage.clickLoginLink();

    LoginPage loginPage = new LoginPage(driver);
    loginPage.login();

    UserInfoPage userInfoPage = new UserInfoPage(driver);

    Assert.assertTrue(driver.getPageSource().contains("User Info page"));
}
```

```
public class WelcomePage{

    @FindBy(id = "login")
    private WebElement loginButton;

    @FindBy(xpath = "//div[@class='col-sm-12']/h2")
    private WebElement welcomeMessage;

    public WelcomePage(WebDriver driver) {
        WebDriverWait wait = new WebDriverWait(driver, timeOutInSeconds: 5);
        wait.until(ExpectedConditions.presenceOfElementLocated(By.id("login")));
        PageFactory.initElements(driver, this);
    }

    //methods
    public void clickLoginLink(){
        loginButton.click();
    }
}
```

Cucumber: **Given** , **when** and **then**

BDD: Behavior-Driven Development

Cucumber

- ▶ TDD - Test Driven Development
 - ▶ Requirement -> Test case -> implementation -> code rephrase
- ▶ BDD - Behavior-Driven Development
 - ▶ Requirement being part of test case
 - ▶ No technical people
 - ▶ CUCUMBER
 - ▶ Gherkin language
 - ▶ FEATURE File
 - ▶ STEP definition
 - ▶ Test Case

Feature: Login functionalities

Scenario: Login with correct credentials

Given web browser loaded the welcome page

When the user click login link in welcome page

Then login page should be displayed

```
@Given("web browser loaded the welcome page")
public void webb_browser_is_open() {
    driver.get("http://localhost:8080");
}

@When("^the user click login link in welcome page$") //
public void click_login_link_in_welcome_page() {
    WelcomePage welcomePage = new WelcomePage(driver);
    welcomePage.clickLoginLink();
}

@Then("^login page should be displayed$")
public void login_page_displayed() {
    String source = driver.getPageSource();
    Assert.assertTrue(source.contains("Email:"));
}
```

Unit Testing

Unit testing involves the testing of each unit or an individual component of the software application.

The purpose is to validate that each unit of the software code performs as expected.

Unit testing is done during development of an application by the developers

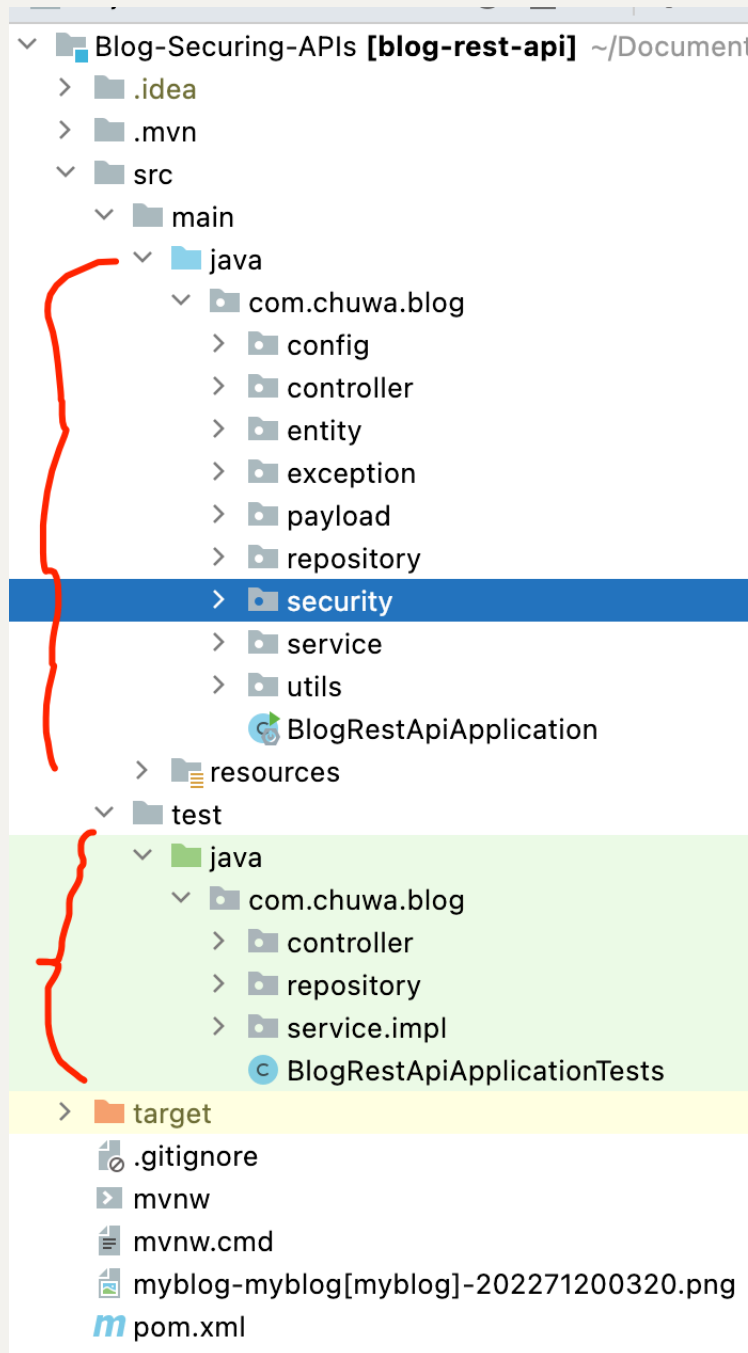
Unit may be an individual function, method, procedure, module and object.

Even though we have QA, usually the developer who developed the new code write unit test by themselves.

Source code structure

It's a good idea to keep the test classes **separate** from the main source code. So, they are developed, executed, and maintained separately from the production code.

Also, it **avoids** any possibility of running test code in the production environment.



Command + Shift + T: create or find the corresponded test

Naming Conventions

Package Naming Convention

*We should create a **similar package structure** in the src/test directory for test classes. Thus, improving the readability and maintainability of the test code. (如上图)*

好的命名能让人根据名字知道该test的意图。

进项目后，看下项目上现有的命名风格是什么。保持统一即可。

<https://methodpoet.com/unit-test-method-naming-convention/>

<https://qualitycoding.org/unit-test-naming/>

My name Convention:

- Class name: xxxxTest.java
 - UserServiceTest.java
- Method name: testMethodName.java
 - testFindUser.java
 - testFindUser_ThrowUserNotFoundException.java

Tools

1. Use Junit to assert if the result equals to expected.

```
1 // 3. assert
2 assertEquals(expected, actual);
3 Assertions.assertEquals(7, actual);
```

2. **Mock** External Services using **Mockito**, **EasyMock**, or **JMockit**.


```
1 BDDMockito.given(employeeRepository.findByEmail(employee.getEmail()))
2                               .willReturn(Optional.empty());
```

Steps of creating Unit Test

Dependency

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-test</artifactId>
4 </dependency>
```

This starter includes:

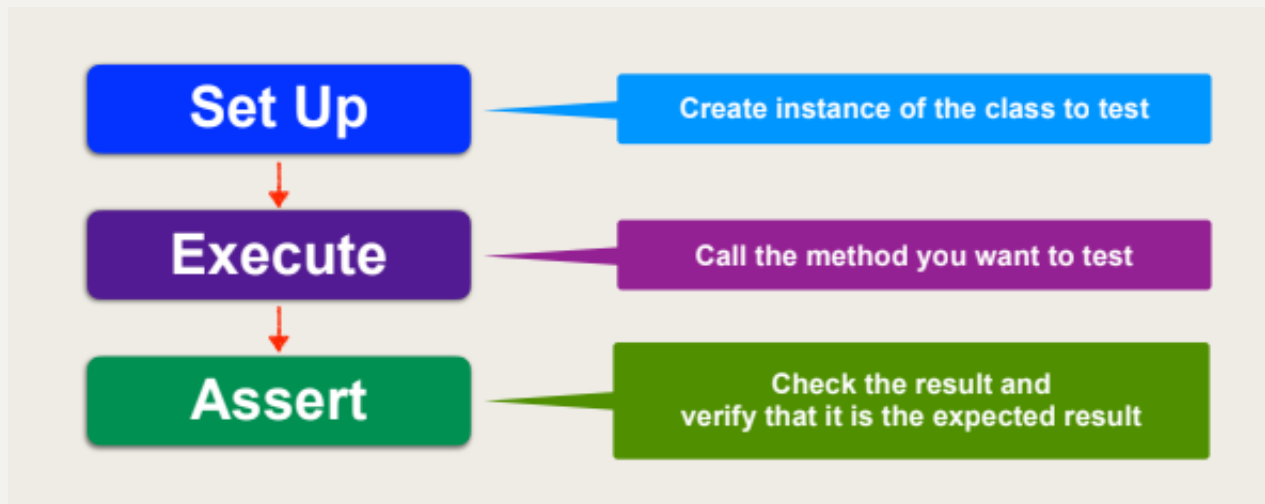
1. Spring-specific dependencies
2. Dependencies for auto-configuration
3. Set of testing libraries - JUnit, Mockito, Hamcrest, AssertJ, JSONNassert and JsonPath



```

63     <groupId>com.jayway.jsonpath</groupId>
64     <artifactId>json-path</artifactId>
65     <version>2.4.0</version>
66     <scope>compile</scope>
67 </dependency>
68 <dependency>
69     <groupId>jakarta.xml.bind</groupId>
70     <artifactId>jakarta.xml.bind-api</artifactId>
71     <version>2.3.3</version>
72     <scope>compile</scope>
73 </dependency>
74 <dependency>
75     <groupId>org.assertj</groupId>
76     <artifactId>assertj-core</artifactId>
77     <version>3.18.1</version>
78     <scope>compile</scope>
79 </dependency>
80 <dependency>
81     <groupId>org.hamcrest</groupId>
82     <artifactId>hamcrest</artifactId>
83     <version>2.2</version>
84     <scope>compile</scope>
85 </dependency>
86 <dependency>
87     <groupId>org.junit.jupiter</groupId>
88     <artifactId>junit-jupiter</artifactId>
89     <version>5.7.0</version>
90     <scope>compile</scope>
91 </dependency>
92 <dependency>

```



```
1 public class DemoUtilsTest extends TestCase {
2
3     @Test
4     public void testEqualsAndNotEquals() {
5         // 1. set up
6         DemoUtils demoUtils = new DemoUtils();
7         int expected = 6;
8
9         // 2. execute
10        int actual = demoUtils.add(2, 4);
11
12        // 3. assert
13        assertEquals(expected, actual);
14    }
15 }
```

JUnit

<https://junit.org/junit5/docs/current/user-guide/#writing-tests>

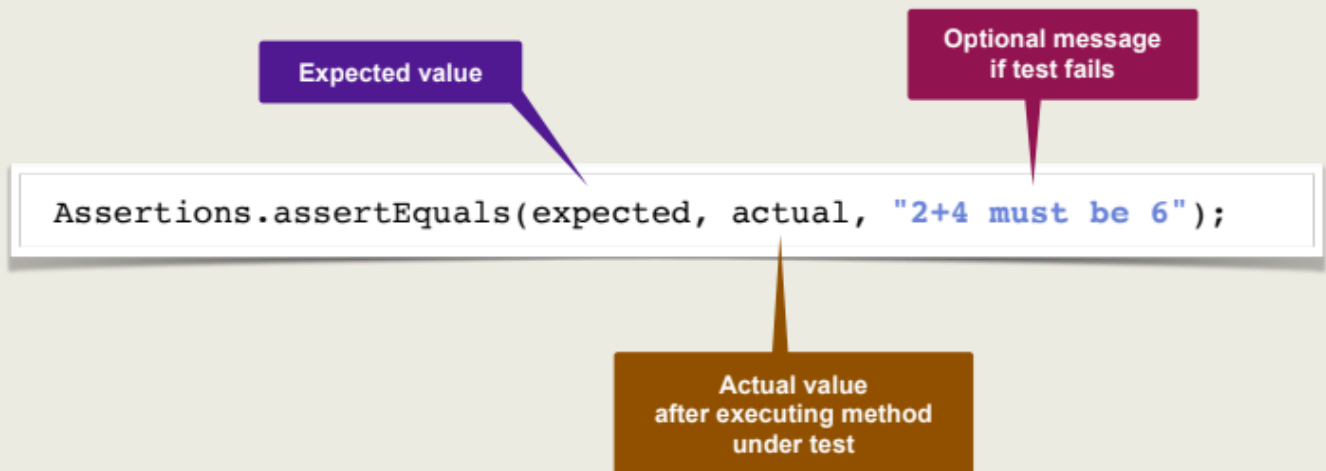
Assertions

JUnit has a collection of assertions

Defined in class: `org.junit.jupiter.api.Assertions`

Method name	Description
<code>assertEquals</code>	Assert that the values are equal
<code>assertNotEquals</code>	Assert that the values are not equal
<code>assertNull</code>	Assert that the value is null
<code>assertNotNull</code>	Assert that the value is not null
...	...

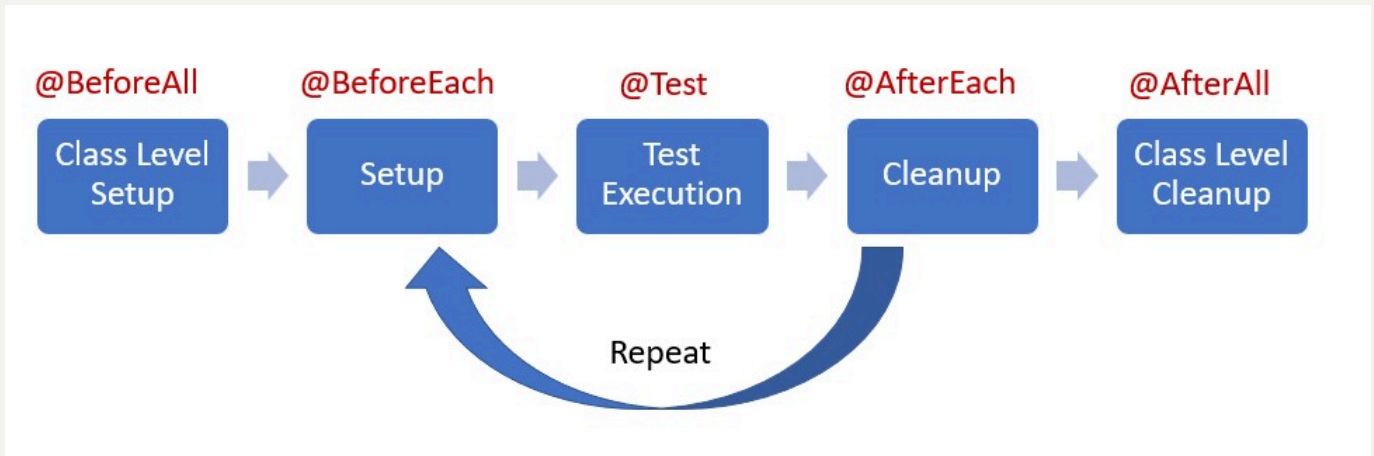
assertEquals



LifeCircle

<https://howtodoinjava.com/junit5/junit-5-test-lifecycle/>

<https://dzone.com/refcardz/junit-and-easymock#:~:text=methods%20and%20attributes.,JUnit%20Lifecycle,teardown%2C%20all%20executed%20in%20sequence.>



`@BeforeAll` 修饰的method是 **static method**，因为他是class level的。

Questions

- Why do we need `@BeforeAll` and `@BeforeEach`?
- If we don't use them, what else ways we can do?

ANNOTATION	DESCRIPTION
<code>@BeforeAll</code>	Method is executed only once, before all test methods. <i>Useful for getting database connections, connecting to servers ...</i>
<code>@AfterAll</code>	Method is executed only once, before all test methods. <i>Useful for getting database connections, connecting to servers ...</i>

DAO Test

- `@DataJpaTest`
 - Spring Boot provides the `@DataJpaTest` annotation to test the persistence layer components that will auto configure in-memory embedded database for testing purposes

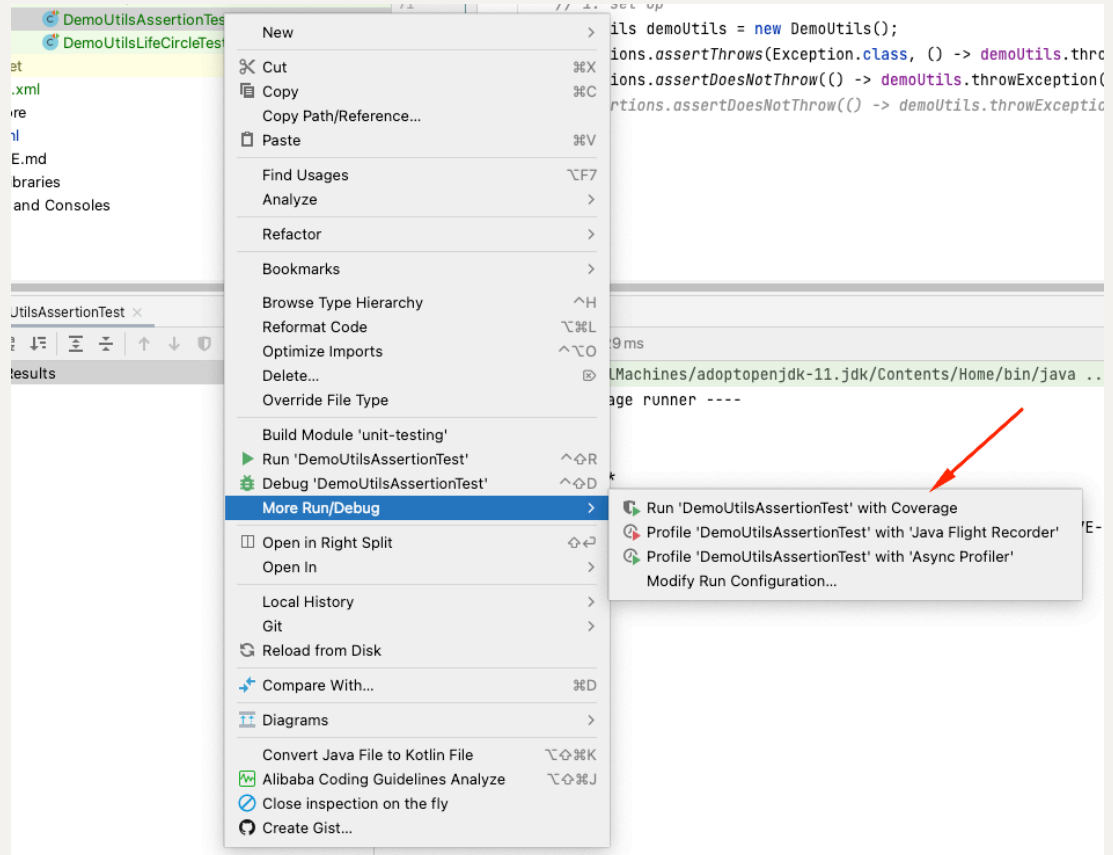
```
1 import org.junit.jupiter.api.Test;
```

Code Coverage

IDE-IntelliJ

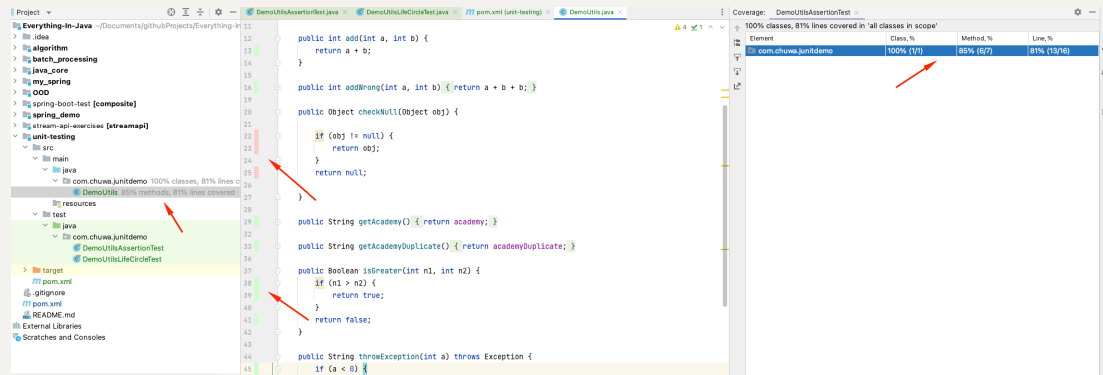
1. how to run: Run xxxTest.java with Coverage

a.



2. Report

a.



Maven

plugin in the pom.xml

report will be produced in test phase, because we set `<phase>test</phase>`

Command line: **mvn clean test**

```
1      <build>
2          <plugins>
3              <plugin>
4                  <groupId>org.jacoco</groupId>
5                  <artifactId>jacoco-maven-plugin</artifactId>
6                  <version>0.8.7</version>
7                  <executions>
8                      <execution>
9                          <goals>
10                             <goal>prepare-agent</goal>
11                         </goals>
12                     </execution>
13                     <execution>
14                         <id>report</id>
15                         <phase>test</phase>
16                         <goals>
17                             <goal>report</goal>
18                         </goals>
19                     </execution>
20                 </executions>
21             </plugin>
22         </plugins>
23     </build>
```

report is in the target/site/index.html

my_spring

OOD

spring-boot-test [composite]

spring_demo

stream-api-exercises [streamapi]

unit-testing

src

main

java

com.chuwa.junitdemo 100% classes, 81% lines covered

DemoUtils 85% methods, 81% lines covered

resources

test

java

com.chuwa.junitdemo

DemoUtilsAssertionTest

DemoUtilsLifeCircleTest

target

classes

generated-sources

generated-test-sources

maven-status

site

jacoco

com.chuwa.junitdemo

jacoco-resources

index.html

jacoco.csv

jacoco.xml

jacoco-sessions.html

surefire-reports

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.8.7</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

<execution>

<id>report</id>

<phase>test</phase>

<goals>

<goal>report</goal>

</goals>

</execution>

</exec>

</plugin>

</plugins>

</build>

</project>

Complex type: **PluginExecution**

Description : The identifier of this execution for labelling the goals during the build, and for matching executions to merge during inheritance.

Version : 4.0.0

unit-testing

unit-testing

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
com.chuwa.junitdemo	0 of 50	100%	0 of 6	100%	0 11	0 16	0 8	0 1
Total	0 of 50	100%	0 of 6	100%	0 11	0 16	0 8	0 1

Mockito

Questions

1. what is unit test?
2. why do we need to use Mockito?

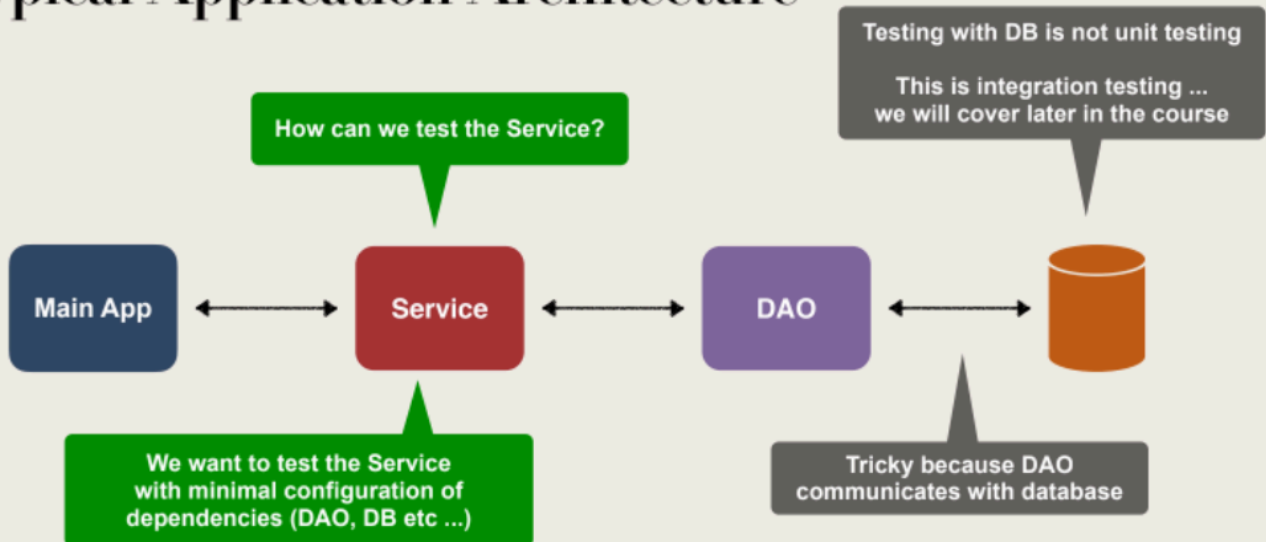
Mocking Frameworks



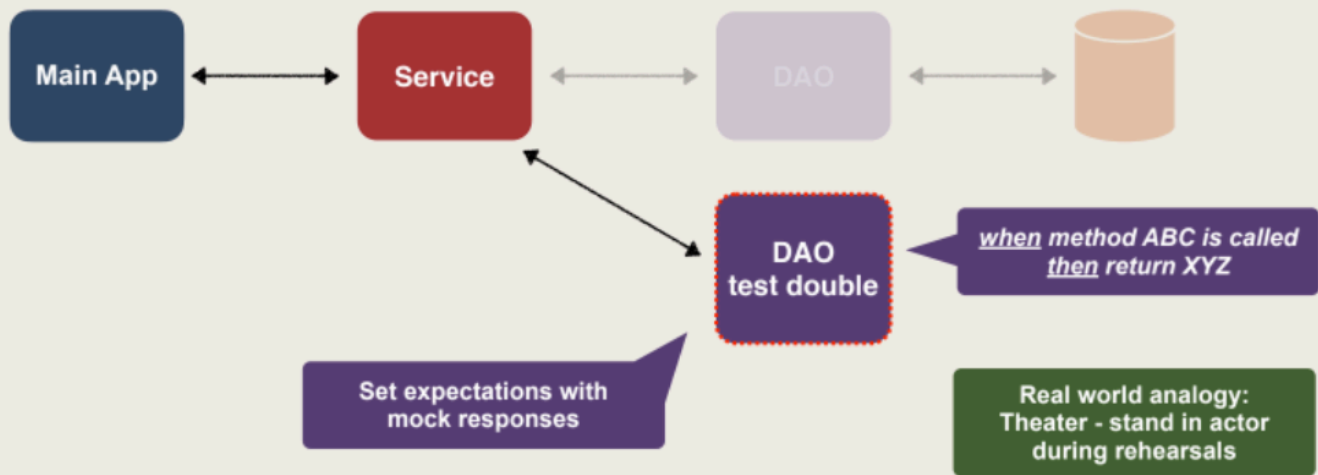
Name	Website
Mockito	<code>site.mockito.org</code>
EasyMock	<code>www.easymock.org</code>
JMockit	<code>jmockit.github.io</code>
...	

**We will use Mockito since
Spring Boot has built-in support for Mockito**

Typical Application Architecture

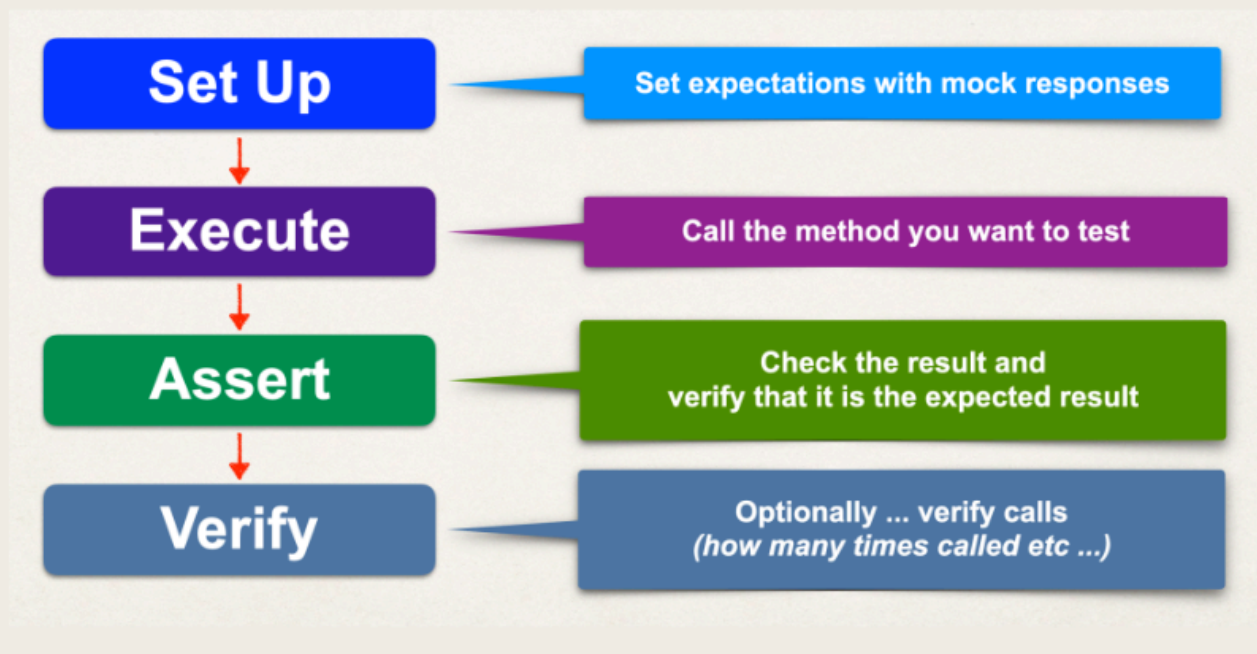


Using a Test Double



Mockito Steps

Set up expectations



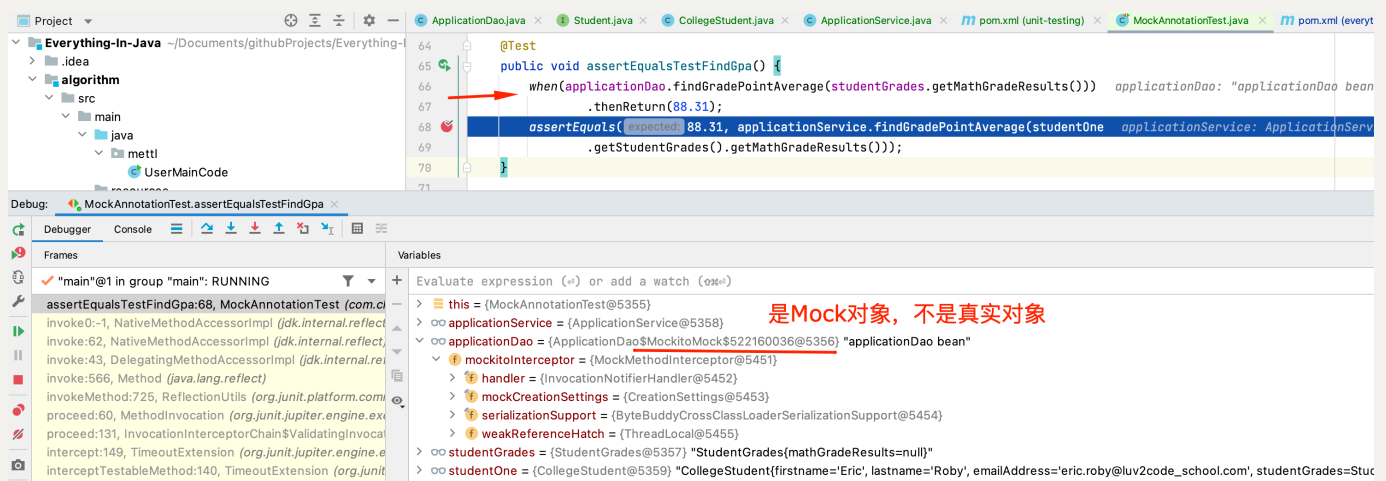
Annotations

- @SpringBootTest
- @Mock
- @Spy
- @InjectMocks
 - 被它修饰的variable service依赖一些其它方法，这些方法被@mock注释，把该类变量注入到service中
 - @InjectMocks FindUserService. this service use findUserBasicInfoDao and findUserExtraInfoDao
 - @Mock findUserBasicInfoDao, @Mock findUserExtraInfoDao
 - 这俩变量会被注入到@InjectMocks FindUserService

Mockito Methods

<https://www.jianshu.com/p/7b67806b74f6>

Need to debug to show we don't dive in the real DAO method.



- Mockito.when(mockObject.targetMethod(args)).thenReturn(desiredReturnValue);
 - args could be :
 - anyString

- anyInt
- anyLong
- anyObject
- any(clazz)
- anyCollection
- anyCollectionOf(clazz)
- anyList(Map, set)
- anyListOf(clazz)
- thenThrow()

```
1 Mockito.when(postRepositoryMock.findById(ArgumentMatchers.anyLong())).thenThrow(new RuntimeException());
```

- **doNothing()** : 指定void方法什么都不做
- **doCallRealMethod()**: 指定方法调用内部的真实逻辑
- **Mockito.doAnswer(desiredAnswer).when(mockObject).targetMethod(args);**

Throw Exception: Consecutive calls

MockAnnotationTest.java

```
@DisplayName("Multiple Stubbing")
@Test
public void stubbingConsecutiveCalls() {
    CollegeStudent nullStudent = (CollegeStudent) context.getBean("collegeStudent");

    when(applicationDao.checkNotNull(nullStudent))
        .thenThrow(new RuntimeException())
        .thenReturn("Do not throw exception second time");

    assertThrows(RuntimeException.class, () -> {
        applicationService.checkNotNull(nullStudent);
    });

    assertEquals("Do not throw exception second time", applicationService.checkNotNull(nullStudent));
}
```

First call

First call ... throw exception

Consecutive calls ... do NOT throw exception
Just return a string

Second call

PowerMockito

PowerMockito is a PowerMock's extension API to support Mockito. It provides capabilities to work with the Java **Reflection** API in a simple way to overcome the problems of Mockito, such as the lack of ability to mock **final**, **static** or **private** methods.

Repo: <https://github.com/TAIsRich/spring-boot-testing>

Branch: <https://github.com/TAIsRich/spring-boot-testing/tree/powerMockito>

```
1  @RunWith(PowerMockRunner.class)
2  @PrepareForTest({EmployeeUtil.class })
3  class CommonServiceImplTest {
4      @Test
5      void getEmailEmployees() {
6          // 已经创建好 mock 对象。
7          PowerMockito.mockStatic(EmployeeUtil.class);
8          // non-void method, just define the behavior for that
method
9
10         PowerMockito.when(EmployeeUtil.initEmployees(ArgumentMatchers.any
Int())).thenReturn(null);
11         // void method, we need set doNothign for that class, then
specify do nothing for which method
12         PowerMockito.doNothing().when(EmployeeUtil.class);
13         EmployeeUtil.deleteEmployees();
14
15         PowerMockito.doNothing().when(EmployeeUtil.class);
16         EmployeeUtil.sth();
17
18         CommonServiceImpl commonService = new CommonServiceImpl();
19         List<Employee> emailEmployees =
commonService.getEmailEmployees("@gmail.com");
20         Assertions.assertThat(emailEmployees).isNull();
21         // Whitebox.setInternalState(EmployeeUtil.class,
"employees", BDDMockito.mock(List.class));
```

```
21  
22     }  
23 }
```

<https://zhuanlan.zhihu.com/p/356231884>

<https://www.baeldung.com/intro-to-powermock>

Integration Test

No Mockito, call real methods.

SonarQube(Code Quality)

<https://next.sonarqube.com/sonarqube/projects>

Conclusion

Unit Test -> Whitebox Test: Need to use **Mockito** to mock dependency method behaviors.
For the static/final/private methods, we need to use PowerMockito.

```

1  Method with return:
2
   Mockito.When(aa.method(ArgumentMatchers.anyString()).thenReturn(new AA());
3
   Mockito.when(bb.method()).thenThrow(Mockito.mock(CCExcetion.class));
4  void method: Mockito.doNothing().when(cc).method();
5
6  @RunWith(PowerMockRunner.class)
7  @PrepareForTest({Math.class})
8  PowerMockito.mockStatic(Math.class)
9  PowerMockito.When(Math.abs(ArgumentMatchers.anyInt())).thenReturn(
10     2)
11  PowerMockito.doNothing().when(EmployeeUtil.class);
11  EmployeeUtil.deleteEmployees();

```

Integration Test -> Blackbox Test: run the real method. **No Mockito**

Life circle: @BeforeAll, @BeforeEach, @Test, @AfterEach, @AfterAll.

@Autowired and @Resource跟testing无关

Mockito.mock(AA.class);

@Mock, @Spy -> @InjectMocks

@Mockbean -> @MockMvc

```
1      // 已经创建好 mock 对象。
2      EmployeeServiceImpl mock =
Mockito.mock(EmployeeServiceImpl.class); // new
EmployeeServiceImpl();
3      PowerMockito.mockStatic(EmployeeUtil.class);
4
5      // non-void method, just define the behavior for that
method
6
7      Mockito.when(mock.getEmployeeById(ArgumentMatchers.anyLong())).th
enReturn(null);
8
9      PowerMockito.when(EmployeeUtil.initEmployees(ArgumentMatchers.any
Int())).thenReturn(null);
10
11     // void method, we need set doNothing for that class, then
specify do nothing for which method
12     Mockito.doNothing().when(mock).getAllEmployees();
13
14     PowerMockito.doNothing().when(EmployeeUtil.class);
15     EmployeeUtil.deleteEmployees();
```

Jira

Agile - Sprint

Sprint Planning -> Grooming -> Daily Standup -> Retro/Sprint Review

We use Jira to assign tickets in every sprint.

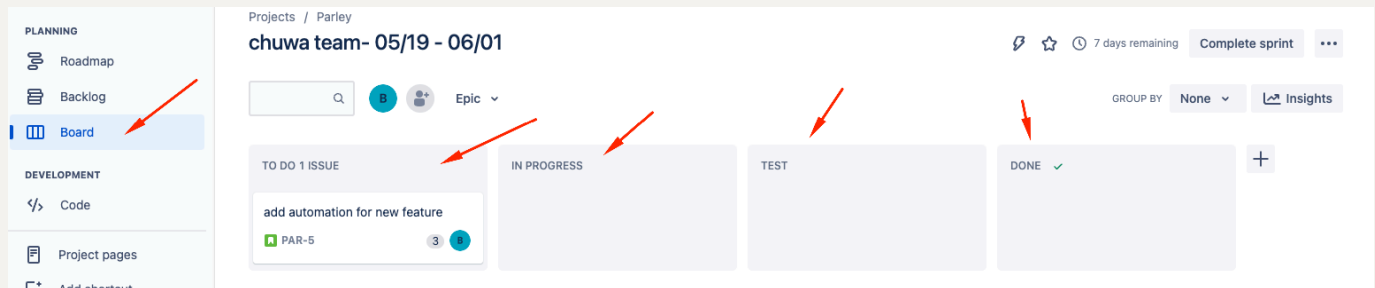
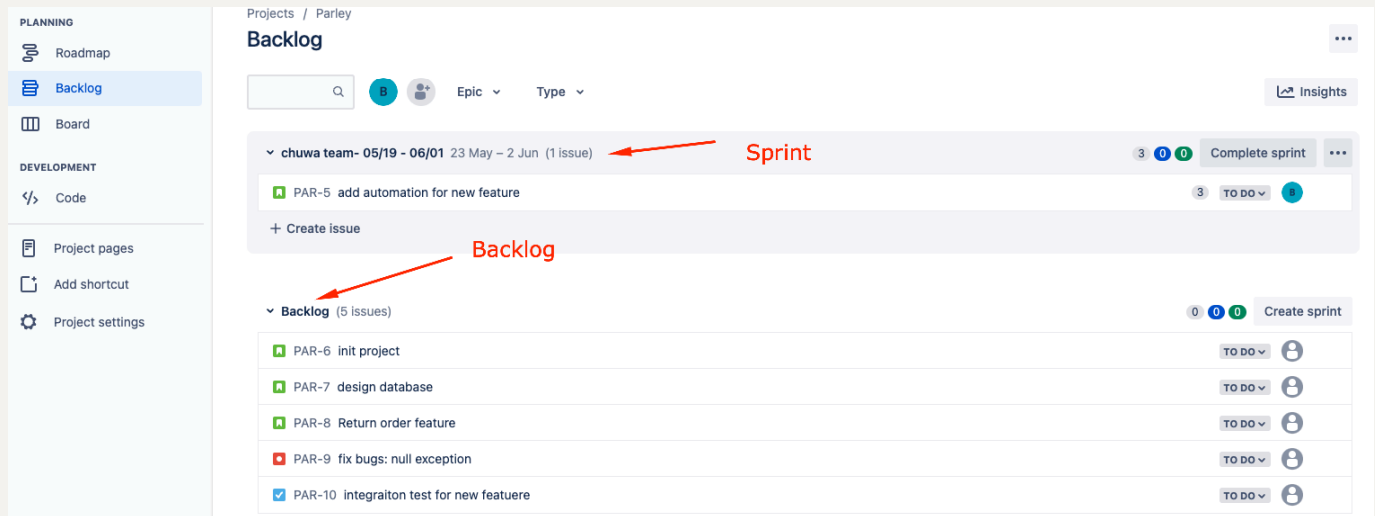
Tickets: new Story/bug fixing, points-based

Backlog & Sprint & Board

Backlog is used to store all the stories/tasks

Sprint is usually 2 weeks. Pick up some stories from backlog to sprint to do.

1 point usually means need 1 day to do.



该task/story/ticket做到哪个阶段，就移动到哪个阶段.

Sprint review 之前需要移动到Done

<https://next.sonarqube.com/sonarqube/projects>