더 자바, “애플리케이션을 테스트하는 다양한 방법”

테스트 코드를 작성하는데 사용할 수 있는 여러 방법과 도구를 설명합니다. 먼저, 테스트 작성하는 자바 개발자 90%가 넘게 사용하는 JUnit 최신 버전을 학습하여 자바로 테스트를 작성하고 실행하는 방법을 소개합니다. 다음으로 Mockito를 사용하여 테스트 하려는 코드의 의존성을 가짜로 만들어 테스트 하는 방법을 학습합니다. 그리고 도커(Docker)를 사용하는 테스트에서 유용하게 사용할 수 있는 Testcontainers를 학습합니다. 다음으로는 관점을 조금 바꿔서 JMeter를 사용해서 성능 테스트하는 방법을 살펴보고 카오스 멍키 (Chaos Monkey)를 사용해서 운영 이슈를 로컬에서 재현하는 방법을 살펴보고, 마지막으로 ArchUnit으로 애플리케이션의 아키텍처를 테스트하는 방법에 대해 학습합니다.

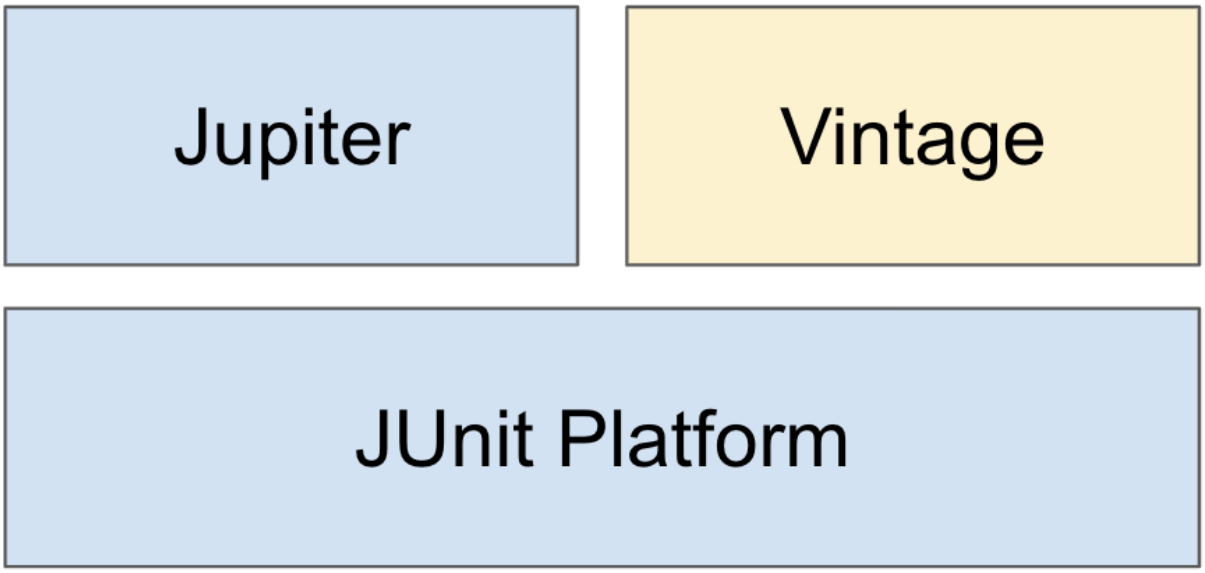
이 강좌를 학습하고 나면 여러분은 자바 애플리케이션을 테스트 하는 다양한 방법을 습득할 수 있습니다. 실제 여러분의 업무와 프로젝트에 필요한 테스트를 작성할 수 있을 겁니다.

1부. JUnit

1. JUnit 5: 소개

자바 개발자가 가장 많이 사용하는 테스팅 프레임워크.

- https://www.jetbrains.com/lp/devecosystem-2019/java/
  - “단위 테스트를 작성하는 자바 개발자 93% JUnit을 사용함.”
- 자바 8 이상을 필요로 함.
- 대체제: TestNG, Spock, ...



Platform: 테스트를 실행해주는 런처 제공. TestEngine API 제공.
Jupiter: TestEngine API 구현체로 JUnit 5를 제공.
Vintage: JUnit 4와 3을 지원하는 TestEngine 구현체.

참고:

- https://junit.org/junit5/docs/current/user-guide/

2. JUnit 5: 시작하기

스프링 부트 프로젝트 만들기

- 2.2+ 버전의 스프링 부트 프로젝트를 만든다면 기본으로 JUnit 5 의존성 추가 됨.

스프링 부트 프로젝트 사용하지 않는다면?

```
<dependency>
   <groupId>org.junit.jupiter</groupId>
   <artifactId>junit-jupiter-engine</artifactId>
   <version>5.5.2</version>
   <scope>test</scope>
</dependency>
```

기본 애노테이션

- @Test
- @BeforeAll / @AfterAll
- @BeforeEach / @AfterEach
- @Disabled

3. JUnit 5: 테스트 이름 표시하기

@DisplayNameGeneration

- Method와 Class 레퍼런스를 사용해서 테스트 이름을 표기하는 방법 설정.
- 기본 구현체로 ReplaceUnderscores 제공

@DisplayName

- 어떤 테스트인지 테스트 이름을 보다 쉽게 표현할 수 있는 방법을 제공하는 애노테이션.
- @DisplayNameGeneration 보다 우선 순위가 높다.

참고:

- https://junit.org/junit5/docs/current/user-guide/#writing-tests-display-names

4. JUnit 5: Assertion

org.junit.jupiter.api.Assertions.*

실제 값이 기대한 값과 같은지 확인	assertEqulas(expected, actual)
값이 null이 아닌지 확인	assertNotNull(actual)
다음 조건이 참(true)인지 확인	assertTrue(boolean)
모든 확인 구문 확인	assertAll(executables...)
예외 발생 확인	assertThrows(expectedType, executable)
특정 시간 안에 실행이 완료되는지 확인	assertTimeout(duration, executable)

마지막 매개변수로 Supplier<String> 타입의 인스턴스를 람다 형태로 제공할 수 있다.

- 복잡한 메시지 생성해야 하는 경우 사용하면 실패한 경우에만 해당 메시지를 만들게 할 수 있다.

AssertJ, Hemcrest, Truth 등의 라이브러리를 사용할 수도 있다.

5. JUnit 5: 조건에 따라 테스트 실행하기

특정한 조건을 만족하는 경우에 테스트를 실행하는 방법.

org.junit.jupiter.api.Assumptions.*

- assumeTrue(조건)
- assumingThat(조건, 테스트)

@Enabled___ 와 @Disabled___

- OnOS
- OnJre
- IfSystemProperty
- IfEnvironmentVariable
- If

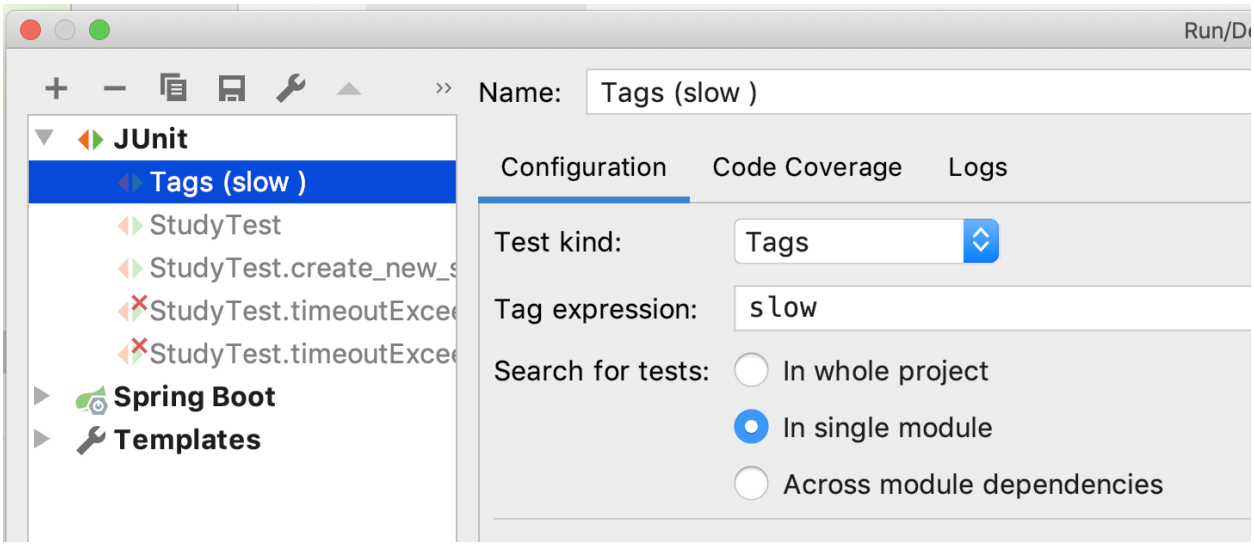
6. JUnit 5: 태깅과 필터링

테스트 그룹을 만들고 원하는 테스트 그룹만 테스트를 실행할 수 있는 기능.

@Tag

- 테스트 메소드에 태그를 추가할 수 있다.
- 하나의 테스트 메소드에 여러 태그를 사용할 수 있다.

인텔리J에서 특정 태그로 테스트 필터링 하는 방법



메이븐에서 테스트 필터링 하는 방법

```
<plugin>
   <artifactId>maven-surefire-plugin</artifactId>
   <configuration>
      <groups>fast | slow</groups>
   </configuration>
</plugin>
```

참고

- https://maven.apache.org/guides/introduction/introduction-to-profiles.html
- https://junit.org/junit5/docs/current/user-guide/#running-tests-tag-expressions

7. JUnit 5: 커스텀 태그

JUnit 5 애노테이션을 조합하여 커스텀 태그를 만들 수 있다.

FastTest.java

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Tag("fast")
@Test
public @interface FastTest {
}
```

```
@FastTest
@DisplayName("스터디 만들기 fast")
void create_new_study() {

@SlowTest
@DisplayName("스터디 만들기 slow")
void create_new_study_again() {
```

8. JUnit 5: 테스트 반복하기 1부

@RepeatedTest

- 반복 횟수와 반복 테스트 이름을 설정할 수 있다.
    - {displayName}
    - {currentRepetition}
    - {totalRepetitions}
- RepetitionInfo 타입의 인자를 받을 수 있다.

@ParameterizedTest

- 테스트에 여러 다른 매개변수를 대입해가며 반복 실행한다.
    - {displayName}
    - {index}
    - {arguments}
    - {0}, {1}, ...

9. JUnit 5: 테스트 반복하기 2부

인자 값들의 소스

- @ValueSource
- @NullSource, @EmptySource, @NullAndEmptySource
- @EnumSource
- @MethodSource
- @CsvSource
- @CvsFileSource
- @ArgumentSource

인자 값 타입 변환

- 암묵적인 타입 변환
  - 레퍼런스 참고
- 명시적인 타입 변환
  - SimpleArgumentConverter 상속 받은 구현체 제공
  - @ConvertWith

인자 값 조합

- ArgumentsAccessor
- 커스텀 Accessor
  - ArgumentsAggregator 인터페이스 구현
  - @AggregateWith

참고

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

10. JUnit 5: 테스트 인스턴스

JUnit은 테스트 메소드 마다 테스트 인스턴스를 새로 만든다.

- 이것이 기본 전략.
- 테스트 메소드를 독립적으로 실행하여 예상치 못한 부작용을 방지하기 위함이다.
- 이 전략을 JUnit 5에서 변경할 수 있다.

@TestInstance(Lifecycle.PER_CLASS)

- 테스트 클래스당 인스턴스를 하나만 만들어 사용한다.
- 경우에 따라, 테스트 간에 공유하는 모든 상태를 @BeforeEach 또는 @AfterEach에서 초기화 할 필요가 있다.
- @BeforeAll과 @AfterAll을 인스턴스 메소드 또는 인터페이스에 정의한 default 메소드로 정의할 수도 있다.

11. JUnit 5: 테스트 순서

실행할 테스트 메소드 특정한 순서에 의해 실행되지만 어떻게 그 순서를 정하는지는 의도적으로 분명히 하지 않는다. (테스트 인스턴스를 테스트 마다 새로 만드는 것과 같은 이유)

경우에 따라, 특정 순서대로 테스트를 실행하고 싶을 때도 있다. 그 경우에는 테스트 메소드를 원하는 순서에 따라 실행하도록 @TestInstance(Lifecycle.PER_CLASS)와 함께 @TestMethodOrder를 사용할 수 있다.

- MethodOrderer 구현체를 설정한다.
- 기본 구현체
    - Alphanumeric
    - OrderAnnoation
    - Random

12. JUnit 5: junit-platform.properties

JUnit 설정 파일로, 클래스패스 루트 (src/test/resources/)에 넣어두면 적용된다.

테스트 인스턴스 라이프사이클 설정
junit.jupiter.testinstance.lifecycle.default = per_class

확장팩 자동 감지 기능
junit.jupiter.extensions.autodetection.enabled = true

@Disabled 무시하고 실행하기
junit.jupiter.conditions.deactivate = org.junit.*DisabledCondition

테스트 이름 표기 전략 설정
junit.jupiter.displayname.generator.default = \
  org.junit.jupiter.api.DisplayNameGenerator$ReplaceUnderscores

13. JUnit 5: 확장 모델

JUnit 4의 확장 모델은 @RunWith(Runner), TestRule, MethodRule.
JUnit 5의 확장 모델은 단 하나, Extension.

확장팩 등록 방법

- 선언적인 등록 @ExtendWith
- 프로그래밍 등록 @RegisterExtension
- 자동 등록 자바 ServiceLoader 이용

확장팩 만드는 방법

- 테스트 실행 조건
- 테스트 인스턴스 팩토리
- 테스트 인스턴스 후-처리기
- 테스트 매개변수 리졸버
- 테스트 라이프사이클 콜백
- 예외 처리
- ...

참고

- https://junit.org/junit5/docs/current/user-guide/#extensions

14. JUnit 5: JUnit 4 마이그레이션

junit-vintage-engine을 의존성으로 추가하면, JUnit 5의 junit-platform으로 JUnit 3과 4로 작성된 테스트를 실행할 수 있다.

- @Rule은 기본적으로 지원하지 않지만, junit-jupiter-migrationsupport 모듈이 제공하는 @EnableRuleMigrationSupport를 사용하면 다음 타입의 Rule을 지원한다.
    - ExternalResource
    - Verifier
    - ExpectedException

JUnit 4	JUnit 5
@Category(Class)	@Tag(String)
@RunWith, @Rule, @ClassRule	@ExtendWith, @RegisterExtension
@Ignore	@Disabled
@Before, @After, @BeforeClass, @AfterClass	@BeforeEach, @AfterEach, @BeforeAll, @AfterAll

15. JUnit 5: 연습 문제

1. 테스트 이름을 표기하는 방법으로 공백, 특수 문자 등을 자유롭게 쓸 수 있는 애노테이션은?

2. JUnit 5, jupiter는 크게 세가지 모듈로 나눌 수 있습니다. 다음 중에서 테스트를 실행하는 런처와 테스트 엔진의 API를 제공하는 모듈은 무엇일까요?
① junit jupiter ② junit vintage ③ junit platform

3. JUnit 5에서 테스트 그룹을 만들고 필터링 하여 실행하는데 사용하는 애노테이션은?

4. 다음 코드는 여러 Assertion을 모두 실행하려는 테스트 코드입니다. 빈칸에 적절한 코드는 무엇인가요?

```
@Test
@DisplayName("스터디 만들기")
void create_new_study() {
    Study actual = new Study(1, "테스트 스터디");
    ________(
        () -> assertEquals(1, actual.getLimit()),
        () -> assertEquals("테스트 스터디", actual.getName()),
        () -> assertEquals(StudyStatus.DRAFT, actual.getStatus())
    );
}
```

5. 다음은 JUnit 5가 제공하는 애노테이션으로 컴포짓 애노테이션을 만드는 코드입니다. 이 애노테이션에 적절한 Rention 전략은 무엇인가요?

```
@Target(ElementType.METHOD)
@Retention(_______________)
@Test
@Tag("fast")
public @interface FastTest {
}
```

6. 다음 중 JUnit 5가 제공하는 확장팩 등록 방법이 아닌것은?
① @ExtendWith
② @Rule
③ @RegisterExtension
④ ServiceLoader

7. 다음 코드는 유즈케이스 테스트를 작성한 것입니다. 다음 빈 칸에 적절한 코드는?

```
@TestInstance(TestInstance.Lifecycle._________)
@TestMethodOrder(MethodOrderer.____________.class)
public class StudyCreateUsecaseTest {

    private Study study;

    @Order(1)
    @Test
    @DisplayName("스터디 만들기")
    public void create_study() {
        study = new Study(10, "자바");
        assertEquals(StudyStatus.DRAFT, study.getStatus());
    }

    @Order(2)
    @Test
    @DisplayName("스터디 공개")
    public void publish_study() {
        study.publish();
        assertEquals(StudyStatus.OPENED, study.getStatus());
        assertNotNull(study.getOpenedDateTime());
    }

}
```

8. 다음은 여러 매개변수를 바꿔가며 동일한 테스트를 실행하는 코드입니다. 빈칸에 적잘한 코드는?

```
@Order(4)
@DisplayName("스터디 만들기")
@_________________(name = "{index} {displayName} message={0}")
@CsvSource({"10, '자바 스터디'", "20, 스프링"})
void parameterizedTest(@____________(StudyAggregator.class) Study study) {
    System.out.println(study);
}

static class StudyAggregator implements ArgumentsAggregator {
    @Override
    public Object aggregateArguments(ArgumentsAccessor accessor,
ParameterContext context) throws ArgumentsAggregationException {
        return new Study(accessor.getInteger(0), accessor.getString(1));
    }
}
```

2부. Mockito

16. Mockito 소개

Mock: 진짜 객체와 비슷하게 동작하지만 프로그래머가 직접 그 객체의 행동을 관리하는 객체.
Mockito: Mock 객체를 쉽게 만들고 관리하고 검증할 수 있는 방법을 제공한다.

테스트를 작성하는 자바 개발자 50%+ 사용하는 Mock 프레임워크.

- https://www.jetbrains.com/lp/devecosystem-2019/java/

현재 최신 버전 3.1.0

단위 테스트에 고찰

- https://martinfowler.com/bliki/UnitTest.html

대체제: EasyMock, JMock

17. Mockito 시작하기

스프링 부트 2.2+ 프로젝트 생성시 spring-boot-starter-test에서 자동으로 Mockito 추가해 줌.

스프링 부트 쓰지 않는다면, 의존성 직접 추가.

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.1.0</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>3.1.0</version>
    <scope>test</scope>
</dependency>
```

다음 세 가지만 알면 Mock을 활용한 테스트를 쉽게 작성할 수 있다.

- Mock을 만드는 방법
- Mock이 어떻게 동작해야 하는지 관리하는 방법
- Mock의 행동을 검증하는 방법

Mockito 레퍼런스

- https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html

18. Mock 객체 만들기

Mockito.mock() 메소드로 만드는 방법

```
MemberService memberService = mock(MemberService.class);
StudyRepository studyRepository = mock(StudyRepository.class);
```

@Mock 애노테이션으로 만드는 방법

- JUnit 5 extension으로 MockitoExtension을 사용해야 한다.
- 필드
- 메소드 매개변수

```
@ExtendWith(MockitoExtension.class)
class StudyServiceTest {

    @Mock MemberService memberService;

    @Mock StudyRepository studyRepository;
```

```
@ExtendWith(MockitoExtension.class)
class StudyServiceTest {

    @Test
    void createStudyService(@Mock MemberService memberService,
                            @Mock StudyRepository studyRepository) {
        StudyService studyService = new StudyService(memberService,
studyRepository);
        assertNotNull(studyService);
    }

}
```

19. Mock 객체 Stubbing

모든 Mock 객체의 행동

- Null을 리턴한다. (Optional 타입은 Optional.empty 리턴)
- Primitive 타입은 기본 Primitive 값.
- 콜렉션은 비어있는 콜렉션.
- Void 메소드는 예외를 던지지 않고 아무런 일도 발생하지 않는다.

Mock 객체를 조작해서

- 특정한 매개변수를 받은 경우 특정한 값을 리턴하거나 예뢰를 던지도록 만들 수 있다.
    - How about some stubbing?
    - Argument matchers
- Void 메소드 특정 매개변수를 받거나 호출된 경우 예외를 발생 시킬 수 있다.
    - Subbing void methods with exceptions
- 메소드가 동일한 매개변수로 여러번 호출될 때 각기 다르게 행동호도록 조작할 수도 있다.
    - Stubbing consecutive calls

20. Mock 객체 Stubbing 연습 문제

다음 코드의 // TODO에 해당하는 작업을 코딩으로 채워 넣으세요.

```
Study study = new Study(10, "테스트");

// TODO memberService 객체에 findById 메소드를 1L 값으로 호출하면
Optional.of(member) 객체를 리턴하도록 Stubbing
// TODO studyRepository 객체에 save 메소드를 study 객체로 호출하면 study 객체
그대로 리턴하도록 Stubbing

studyService.createNewStudy(1L, study);

assertNotNull(study.getOwner());
assertEquals(member, study.getOwner());
```

git checkout 216112f5706fef76f56c735faed200f311b8d919

21. Mock 객체 확인

Mock 객체가 어떻게 사용이 됐는지 확인할 수 있다.

- 특정 메소드가 특정 매개변수로 몇번 호출 되었는지, 최소 한번은 호출 됐는지, 전혀 호출되지 않았는지
    - Verifying exact number of invocations
- 어떤 순서대로 호출했는지
    - Verification in order
- 특정 시간 이내에 호출됐는지
    - Verification with timeout
- 특정 시점 이후에 아무 일도 벌어지지 않았는지
    - Finding redundant invocations

22. Mockito BDD 스타일 API

BDD: 애플리케이션이 어떻게 “행동”해야 하는지에 대한 공통된 이해를 구성하는 방법으로, TDD에서 창안했다.

행동에 대한 스팩

- Title
- Narrative
    - As a / I want / so that
- Acceptance criteria
    - Given / When / Then

Mockito는 BddMockito라는 클래스를 통해 BDD 스타일의 API를 제공한다.

When -> Given

```
given(memberService.findById(1L)).willReturn(Optional.of(member));
given(studyRepository.save(study)).willReturn(study);
```

Verify -> Then

```
then(memberService).should(times(1)).notify(study);
then(memberService).shouldHaveNoMoreInteractions();
```

참고

- https://javadoc.io/static/org.mockito/mockito-core/3.2.0/org/mockito/BDDMockito.html
- https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html#BDD_behavior_verification

23. Mockito 연습 문제

다음 StudyService 코드에 대한 테스트를 Mockito를 사용해서 Mock 객체를 만들고 Stubbing과 Verifying을 사용해서 테스트를 작성하세요.

StudyService.java

```
public Study openStudy(Study study) {
    study.open();
    Study openedStudy = repository.save(study);
    memberService.notify(openedStudy);
    return openedStudy;
}
```

StudyServiceTest.java

```
@DisplayName("다른 사용자가 볼 수 있도록 스터디를 공개한다.")
@Test
void openStudy() {
    // Given
    StudyService studyService = new StudyService(memberService,
studyRepository);
    Study study = new Study(10, "더 자바, 테스트");
    // TODO studyRepository Mock 객체의 save 메소드를호출 시 study를
리턴하도록 만들기.

    // When
    studyService.openStudy(study);

    // Then
    // TODO study의 status가 OPENED로 변경됐는지 확인
    // TODO study의 openedDataTime이 null이 아닌지 확인
    // TODO memberService의 notify(study)가 호출 됐는지 확인.
}
```

git checkout d52c6af44e82ca1c474bd45290e4a44340ffd483

3부. 도커와 테스트

24. Testcontainers 소개

테스트에서 도커 컨테이너를 실행할 수 있는 라이브러리.

- https://www.testcontainers.org/
- 테스트 실행시 DB를 설정하거나 별도의 프로그램 또는 스크립트를 실행할 필요 없다.
- 보다 Production에 가까운 테스트를 만들 수 있다.
- 테스트가 느려진다.

25. Testcontainers 설치

Testcontainers JUnit 5 지원 모듈 설치

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>1.15.1</version>
    <scope>test</scope>
</dependency>
```

https://www.testcontainers.org/test_framework_integration/junit_5/

@Testcontainers

- JUnit 5 확장팩으로 테스트 클래스에 @Container를 사용한 필드를 찾아서 컨테이너 라이프사이클 관련 메소드를 실행해준다.

@Container

- 인스턴스 필드에 사용하면 모든 테스트 마다 컨테이너를 재시작 하고, 스태틱 필드에 사용하면 클래스 내부 모든 테스트에서 동일한 컨테이너를 재사용한다.

여러 모듈을 제공하는데, 각 모듈은 별도로 설치해야 한다.

- Postgresql 모듈 설치
- https://www.testcontainers.org/modules/databases/
- https://www.testcontainers.org/modules/databases/postgres/

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>postgresql</artifactId>
    <version>1.15.1</version>
    <scope>test</scope>
</dependency>
```

application.properties

```
spring.datasource.url=jdbc:tc:postgresql:///studytest
spring.datasource.driver-class-name=org.testcontainers.jdbc.ContainerDatabaseDriver
```

26. Testcontainers, 기능 살펴보기

컨테이너 만들기

- New GenericContainer(String imageName)

네트워크

- withExposedPorts(int...)
- getMappedPort(int)

환경 변수 설정

- withEnv(key, value)

명령어 실행

- withCommand(String cmd...)

사용할 준비가 됐는지 확인하기

- waitingFor(Wait)
- Wait.forHttp(String url)
- Wait.forLogMessage(String message)

로그 살펴보기

- getLogs()
- followOutput()

27. Testcontainers, 컨테이너 정보를 스프링 테스트에서 참조하기

@ContextConfiguration

- 스프링이 제공하는 애노테이션으로, 스프링 테스트 컨텍스트가 사용할 설정 파일 또는 컨텍스트를 커스터마이징할 수 있는 방법을 제공한다.

ApplicationContextInitializer

- 스프링 ApplicationContext를 프로그래밍으로 초기화 할 때 사용할 수 있는 콜백 인터페이스로, 특정 프로파일을 활성화 하거나, 프로퍼티 소스를 추가하는 등의 작업을 할 수 있다.

TestPropertyValues

- 테스트용 프로퍼티 소스를 정의할 때 사용한다.

Environment

- 스프링 핵심 API로, 프로퍼티와 프로파일을 담당한다.

전체 흐름

1. Testcontainer를 사용해서 컨테이너 생성
2. ApplicationContextInitializer를 구현하여 생선된 컨테이너에서 정보를 축출하여 Environment에 넣어준다.
3. @ContextConfiguration을 사용해서 ApplicationContextInitializer 구현체를 등록한다.
4. 테스트 코드에서 Environment, @Value, @ConfigurationProperties 등 다양한 방법으로 해당 프로퍼티를 사용한다.

28. Testcontainers, 도커 Compose 사용하기 1

테스트에서 (서로 관련있는) 여러 컨테이너를 사용해야 한다면?

Docker Compose: https://docs.docker.com/compose/

- 여러 컨테이너를 한번에 띄우고 서로 간의 의존성 및 네트워크 등을 설정할 수 있는 방법
- docker-compose up / down

Testcontainser의 docker compose 모듈을 사용할 수 있다.

- https://www.testcontainers.org/modules/docker_compose/

대체제: https://github.com/palantir/docker-compose-rule

- 2019 가을 KSUG 발표 자료 참고
- https://bit.ly/2q8S3Qo

29. Testcontainers, 도커 Compose 사용하기 2

도커 Compose 서비스 정보 참조하기

특정 서비스 Expose

```
@Container
static DockerComposeContainer composeContainer =
        new DockerComposeContainer(new
File("src/test/resources/docker-compose.yml"))
        .withExposedService("study-db", 5432);
```

Compose 서비스 정보 참조

```
static class ContainerPropertyInitializer implements
ApplicationContextInitializer<ConfigurableApplicationContext> {

    @Override
    public void initialize(ConfigurableApplicationContext context) {
        TestPropertyValues.of("container.port=" +
composeContainer.getServicePort("study-db", 5432))
                .applyTo(context.getEnvironment());
    }
}
```

4부. 성능 테스트

30. JMeter 소개

https://jmeter.apache.org/

성능 측정 및 부하 (load) 테스트 기능을 제공하는 오픈 소스 자바 애플리케이션.

다양한 형태의 애플리케이션 테스트 지원
- 웹 - HTTP, HTTPS
- SOAP / REST 웹 서비스
- FTP
- 데이터베이스 (JDBC 사용)
- Mail (SMTP, POP3, IMAP)
- ...

CLI 지원
- CI 또는 CD 툴과 연동할 때 편리함.
- UI 사용하는 것보다 메모리 등 시스템 리소스를 적게 사용.

주요 개념
- Thread Group: 한 쓰레드 당 유저 한명
- Sampler: 어떤 유저가 해야 하는 액션
- Listener: 응답을 받았을 할 일 (리포팅, 검증, 그래프 그리기 등)
- Configuration: Sampler 또는 Listener가 사용할 설정 값 (쿠키, JDBC 커넥션 등)
- Assertion: 응답이 성공적인지 확인하는 방법 (응답 코드, 본문 내용 등)

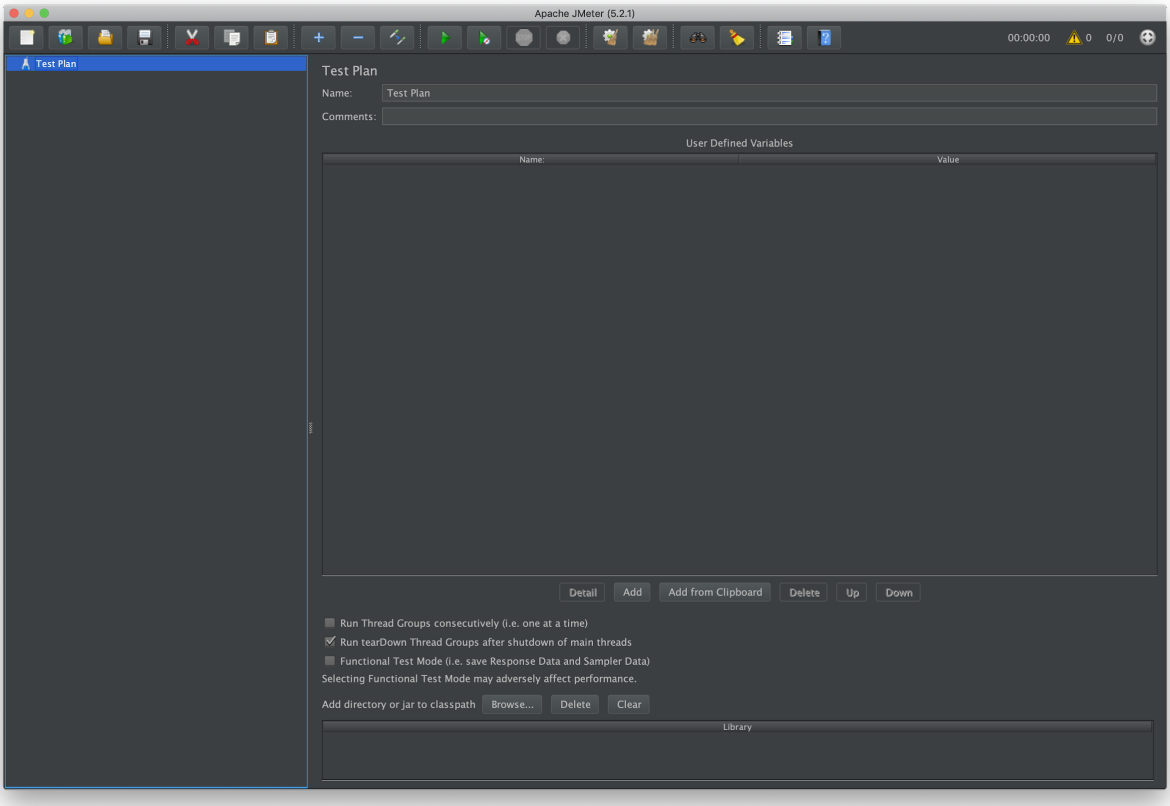
대체제:
- Gatling
- nGrinder

31. JMeter 설치

https://jmeter.apache.org/download_jmeter.cgi

압축 파일 받고 압축 파일 풀기. 원한다면 PATH에 bin 디렉토리를 추가.

bin/jmeter 실행하기



32. JMeter 사용하기

Thread Group 만들기

- Number of Threads: 쓰레드 개수
- Ramp-up period: 쓰레드 개수를 만드는데 소요할 시간
- Loop Count: infinite 체크 하면 위에서 정한 쓰레드 개수로 계속 요청 보내기. 값을 입력하면 해당 쓰레드 개수 X 루프 개수 만큼 요청 보냄.

Sampler 만들기

- 여러 종류의 샘플러가 있지만 그 중에 우리가 사용할 샘플러는 HTTP Request 샘플러.
- HTTP Sampler
    - 요청을 보낼 호스트, 포트, URI, 요청 본문 등을 설정
- 여러 샘플러를 순차적으로 등록하는 것도 가능하다.

Listener 만들기

- View Results Tree
- View Resulrts in Table
- Summary Report
- Aggregate Report
- Response Time Graph
- Graph Results
- ...

Assertion 만들기

- 응답 코드 확인
- 응답 본문 확인

CLI 사용하기

- jmeter -n -t 설정 파일 -l 리포트 파일

5부. 운영 이슈 테스트

33. Chaos Monkey 소개

카오스 엔지니어링 툴

- 프로덕션 환경, 특히 분산 시스템 환경에서 불확실성을 파악하고 해결 방안을 모색하는데 사용하는 툴

운영 환경 불확실성의 예

- 네트워크 지연
- 서버 장애
- 디스크 오작동
- 메모리 누수
- ...

카오스 멍키 스프링 부트

- 스프링 부트 애플리케이션에 카오스 멍키를 손쉽게 적용해 볼 수 있는 툴
- 즉, 스프링 부트 애플리케이션을 망가트릴 수 있는 툴

카오스 멍키 스프링 부트 주요 개념

공격 대상 (Watcher)	공격 유형 (Assaults)
• @RestController • @Controller • @Service • @Repository • @Component	• 응답 지연 (Latency Assault) • 예외 발생 (Exception Assault) • 애플리케이션 종료 (AppKiller Assault) • 메모리 누수 (Memory Assault)

34. Chaos Monkey 설치

의존성 추가

```
<dependency>
   <groupId>de.codecentric</groupId>
   <artifactId>chaos-monkey-spring-boot</artifactId>
   <version>2.1.1</version>
</dependency>

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Chaos-monkey-spring-boot

- 스프링 부트용 카오스 멍키 제공

Spring-boot-starter-actuator

- 스프링 부트 운영 툴로, 런타임 중에 카오스 멍키 설정을 변경할 수 있다.
- 그밖에도 헬스 체크, 로그 레벨 변경, 매트릭스 데이터 조회 등 다양한 운영 툴로 사용 가능.
- /actuator

카오스 멍키 활성화

- spring.profiles.active=chaos-monkey

스프링 부트 Actuator 엔드 포인트 활성화

```
management.endpoint.chaosmonkey.enabled=true
management.endpoints.web.exposure.include=health,info,chaosmonkey
```

35. CM4SB 응답 지연

응답 지연 이슈 재현 방법

1. Repository Watcher 활성화

chaos.monkey.watcher.repository=true

2. 카오스 멍키 활성화

http post localhost:8080/actuator/chaosmonkey/enable

3. 카오스 멍키 활서화 확인

http localhost:8080/actuator/chaosmonkey/status

4. 카오스 멍키 와처 확인

http localhost:8080/actuator/chaosmonkey/watchers

5. 카오스 멍키 지연 공격 설정

http POST localhost:8080/actuator/chaosmonkey/assaults level=3 latencyRangeStart=2000 latencyRangeEnd=5000 latencyActive=true

6. 테스트

JMeter 확인

A -> B1, B2

참고:

https://codecentric.github.io/chaos-monkey-spring-boot/2.1.1/#_customize_watcher

36. CM4SB 에러 발생

에러 발생 재현 방법

http POST localhost:8080/actuator/chaosmonkey/assaults level=3 latencyActive=false exceptionsActive=true exception.type=java.lang.RuntimeException

https://codecentric.github.io/chaos-monkey-spring-boot/2.1.1/#_examples

6부. 아키텍처 테스트

37. ArchUnit 소개

https://www.archunit.org/

애플리케이션의 아키텍처를 테스트 할 수 있는 오픈 소스 라이브러리로, 패키지, 클래스, 레이어, 슬라이스 간의 의존성을 확인할 수 있는 기능을 제공한다.

아키텍처 테스트 유즈 케이스

- A 라는 패키지가 B (또는 C, D) 패키지에서만 사용 되고 있는지 확인 가능.
- *Serivce라는 이름의 클래스들이 *Controller 또는 *Service라는 이름의 클래스에서만 참조하고 있는지 확인.
- *Service라는 이름의 클래스들이 ..service.. 라는 패키지에 들어있는지 확인.
- A라는 애노테이션을 선언한 메소드만 특정 패키지 또는 특정 애노테이션을 가진 클래스를 호출하고 있는지 확인.
- 특정한 스타일의 아키텍처를 따르고 있는지 확인.

참고

- https://blogs.oracle.com/javamagazine/unit-test-your-architecture-with-archunit
- https://www.archunit.org/userguide/html/000_Index.html
- Moduliths

38. ArchUnit 설치

https://www.archunit.org/userguide/html/000_Index.html#_junit_5

JUnit 5용 ArchUnit 설치

```
<dependency>
   <groupId>com.tngtech.archunit</groupId>
   <artifactId>archunit-junit5-engine</artifactId>
   <version>0.12.0</version>
   <scope>test</scope>
</dependency>
```

주요 사용법

1. 특정 패키지에 해당하는 클래스를 (바이트코드를 통해) 읽어들이고
2. 확인할 규칙을 정의하고
3. 읽어들인 클래스들이 그 규칙을 잘 따르는지 확인한다.

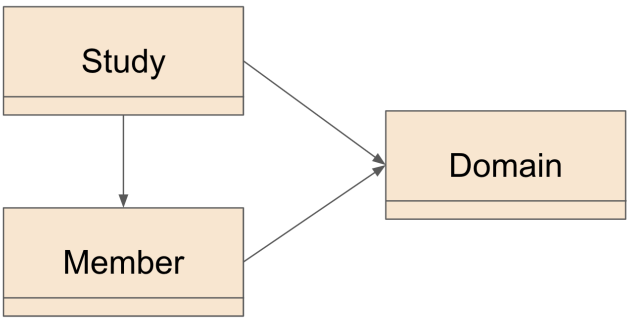
```
@Test
public void Services_should_only_be_accessed_by_Controllers() {
    JavaClasses importedClasses = new
ClassFileImporter().importPackages("com.mycompany.myapp");

    ArchRule myRule = classes()
        .that().resideInAPackage("..service..")
        .should().onlyBeAccessed().byAnyPackage("..controller..",
"..service..");

    myRule.check(importedClasses);
}
```

JUnit 5 확장팩 제공

- @AnalyzeClasses: 클래스를 읽어들여서 확인할 패키지 설정
- @ArchTest: 확인할 규칙 정의

39. ArchUnit: 패키지 의존성 확인하기

확인하려는 패키지 구조



실제로 그런지 확인하려면...

- ..domain.. 패키지에 있는 클래스는 ..study.., ..member.., ..domain에서 참조 가능.
- ..member.. 패키지에 있는 클래스는 ..study..와 ..member..에서만 참조 가능.
    - (반대로) ..domain.. 패키지는 ..member.. 패키지를 참조하지 못한다.
- ..study.. 패키지에 있는 클래스는 ..study.. 에서만 참조 가능.
- 순환 참조 없어야 한다.

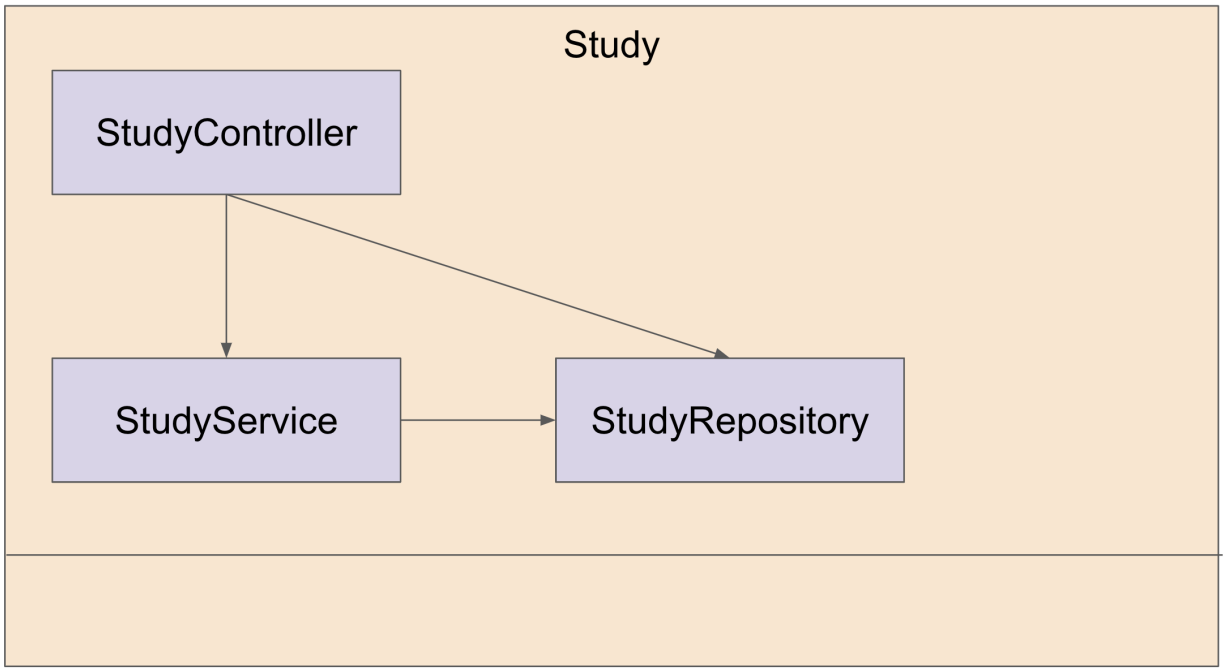
40. ArchUnit: JUnit 5 연동하기

@AnalyzeClasses: 클래스를 읽어들여서 확인할 패키지 설정
@ArchTest: 확인할 규칙 정의

```
@AnalyzeClasses(packagesOf = App.class)
public class ArchTests {

    @ArchTest
    ArchRule domainPackageRule = classes().that().resideInAPackage("..domain..")
            .should().onlyBeAccessed().byClassesThat()
            .resideInAnyPackage("..study..", "..member..", "..domain..");

    @ArchTest
    ArchRule memberPackageRule = noClasses().that().resideInAPackage("..domain..")
            .should().accessClassesThat().resideInAPackage("..member..");

    @ArchTest
    ArchRule studyPackageRule = noClasses().that().resideOutsideOfPackage("..study..")
            .should().accessClassesThat().resideInAnyPackage("..study..");

    @ArchTest
    ArchRule freeOfCycles = slices().matching("..inflearnthejavatest.(*)..")
            .should().beFreeOfCycles();

}
```

41. ArchUnit: 클래스 의존성 확인하기

확인하려는 클래스 의존성



테스트 할 내용

- StudyController는 StudyService와 StudyRepository를 사용할 수 있다.
- Study* 로 시작하는 클래스는 ..study.. 패키지에 있어야 한다.
- StudyRepository는 StudyService와 StudyController를 사용할 수 없다.

7부. 정리

42. 더 자바, 애플리케이션을 테스트하는 다양한 방법 정리

이밖에도..

Selenium WebDriver
- 웹 브라우저 기반 자동화된 테스트 작성에 사용할 수 있는 툴

DBUnit
- 데이터베이스에 데이터를 CVS, Excel 등으로 넣어주는 툴

REST Assured
- REST API 테스트 라이브러리

Cucumber
- BDD를 지원하는 테스트 라이브러리.