

세미나 1

1. 안드로이드 테스트 종류 및 설명
 2. 안드로이드 테스트를 위한 설정
 3. TDD란?
 - 아키텍처(개발영역에 따른 TDD 작성 패턴)
 - MVC, MVP, MVVP 패턴 설정
 - 예제 설명
 - 테스트 법칙
 4. 원가드어드민 LoginActivity 리팩토링
-

1. 테스트의 종류

- Unit테스트 - Local Unit Testing(안드로이드 시스템에 대한 의존성 없음)
 - 관련 툴: JUnit, Mockito, PowerMock
 - 에뮬레이터나 실제 단말로 테스트를 한다.
 - test: 유닛 테스트가 위치하는 곳으로 여러분의 로컬 머신의 JVM에서 수행되며 에뮬레이터나 실물 디바이스에서 돌릴 수 없습니다. 즉, 이 테스트로는 Context와 같은 안드로이드 클래스에 접근할 수 없습니다.
- UI테스트 - Instrumented Unit Testing
 - Espresso, UIAutomator, Robotium, Appium, Calabash, Robolectric
 - Mock 이라는 실제 오브젝트 대신 가상의 오브젝트로 대체하여 테스트를 진행하는 기법
 - 중간단계의 테스트로, UI Framework를 제외하고 테스트를 하기 때문에 UI Tests보단 가볍고 Mock이 아닌 실제 Jar파일을 사용한다.
 - androidTest : Android Instrumentation 테스트로 안드로이드 에뮬레이터나 실물 디바이스에서 테스트를 수행할 수 있습니다.
- 처음 접한 용어
 - JUnit4
 - Espresso
 - JUnit Rules, AndroidJUnitRunner
 - ActivityTestRule(Activity 테스트), ServiceTestRule (서비스 테스트)
 - Abstract/Interface 테스트 할 도구 Mockito
 - When, thenReturn 사용
 - <https://softarchitecture.tistory.com/64>

2. 안드로이드 테스트 위한 설정

```
android {  
    ...  
    testOptions {  
        unitTests.returnDefaultValues = true  
    }  
  
    defaultConfig {  
        ...  
  
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
    }  
}
```

그 외 `androidTestCompile` 할수 있는 `dependency`를 추가하여 테스트

```
dependencies {  
    // JUnit4  
    testImplementation 'junit:junit:4.12'  
  
    androidTestCompile 'com.android.support:support-annotations:23.3.0'  
  
    // Test dependency  
    androidTestCompile 'com.android.support.test:runner:0.5'  
    ...  
    // mockito  
    testImplementation 'org.mockito:mockito-core:2.22.0'  
  
    // Espresso  
    androidTestImplementation 'com.android.support.test:runner:1.0.2'  
    androidTestImplementation 'com.android.support.test:rules:1.0.2'  
    androidTestImplementation 'com.android.support.test.espresso:espresso-  
core:3.0.2'  
    androidTestImplementation 'com.android.support.test.espresso:espresso-  
intents:3.0.2'}
```

3. Annotation

- `@Before` : `@Test`를 시작하기 전 사전에 진행해야 할 사전 정의에 해당됩니다. `@Test`가 시작되기 전 항상 호출되게 됩니다.(단위 테스트 포함)
- `@After` : `@After`은 모든 테스트가 종료되면 호출되게 됩니다. 메모리에서 resource를 release 할 수 있습니다.
- `@Test` : `@Before`가 완료되면 실제 코드 테스트를 진행하게 됩니다.
- `@Rule` : 해당 Test class에서 사용하게 될 `ActivityTestRule`과 `ServiceTestRule`에 대하여 정의하게 됩니다.
- `@BeforeClass`, `@AfterClass` : public static method로 정의하여야 하며, `@Before`, `@After`와 동일하게 한 번씩만 실행되게 됩니다.

- @Test(timeout=) : @Test 룰에 대한 timeout을 지정하게 됩니다. timeout 안에 테스트가 완료되지 않으면 fail이 되며, time은 milliseconds으로만 사용할 수 있습니다. 예) @Test(timeout=500)
- @RequiresDevice : 에뮬레이터를 사용하지 않고 기기만 사용할 수 있습니다.
- @SdkSupress : minSdkVersion을 지정할 수 있습니다.
- @SmallTest, @MediumTest, @LargeTest : 테스트 성격을 구분하여 테스트할 수 있습니다.

4. Unit 기본 함수

- void assertEquals (expected, actual) - 두 프리미티브 / 객체가 같은지 확인합니다.
- void assertTrue (condition)*- 조건이 참인지 확인합니다.
- void assertFalse (condition) - 조건이 false인지 확인합니다.
- void assertNotNull (object) - 객체가 null이 아닌지 검사합니다.
- void assertNull (object) - 객체가 null인지 검사합니다.
- void assertSame (expected, actual) - assertEquals () 메소드는 두 개의 객체 참조가 같은 객체를 가리키는 지 여부를 테스트합니다.
- void assertNotSame (unexpected, actual) - assertNotSame () 메소드는 두 객체 참조가 같은 객체를 가리키지 않는지 여부를 테스트합니다.
- void assertEquals (expectedArray, actualArray) - assertEquals () 메서드는 두 배열이 서로 같은지 여부를 테스트합니다.

테스트 앱

<https://alexzh.com/2016/03/24/android-testing-unit-testing/>

안드로이드 gradle 종속성

https://developer.android.com/studio/build/dependencies?utm_source=android-studio#dependency_configurations

TDD

- Test Driven Development : 테스트 주도개발(테스트가 개발을 이끌어 간다는 이념)
- 테스트를 먼저 만들고 테스트를 통과하기 위한 코드를 짜는 것
- TDD 적용 상황
 - 어떤 부분에 대한 코딩을 여러번 해봤고 결과가 어떻게 나올지 뻔하다면 TDD를 하지 않아도 된다.
 - 처음해보는 주제, 고객의 요구 조건이 바뀔수 있는 경우, 내가 개발하고 나서 이코드를 누가 유지보수 할지 모르는 경우에 적합
- TDD를 하면 개발시간이 늘어난다? —> 아니다
- TDD를 하면 코드 복잡도가 떨어진다. 엔트로피(Entropy)가 낮아진다. 깨끗한 코드 가 나온다. 유지보수 비용이 낮아진다.

단위테스트 지원 라이브러리

개발영역에 따른 TDD 작성패턴

Mock 를 이용한 TDD

Architecture 고민

MVC, MVP, MVVM

Taestable 테스트 가능한 아키텍처의 핵심 아이디어는 응용프로그램의 부분을 분리하여 개별적으로 유지관리하고 테스트 할수 있도록 하는 것

<https://github.com/ss4076/ticTacToe>

MVC

M(Model): 데이터 + 상태 + 비즈니스 로직

- 뷰나 컨트롤러에 묶이지 않으며 재활용 가능함

V(View): 표현, 사용자가 앱과 상호작용 할 때 컨트롤러와 통신하는 책임을 맡음

C(Controller): 뷰가 컨트롤러에게 사용자가 버튼을 눌렀다고 알리면, 그에 따라 어떻게 모델과 상호작용할 지 결정, 모델에서 데이터가 변화되는 것에 따라 컨트롤러는 뷰의 상태를 적절하게 업데이트 하도록 결정(액티비티, 프래그먼트)

문제점

- 테스트의 용이성
 - 컨트롤러가 안드로이드 API에 종속되므로 유닛테스트 어려움
- 모듈화 및 유연성
 - 컨트롤러가 뷰에 결합되며 뷰가 변경되면 컨트롤러도 변경해야함
- 유지보수
 - 시간이 지남에 따라 컨트롤러가 비대해 짐



MVP

컨트롤러의 책임에 묶이지 않고도 뷰와 컨트롤러가 자연스럽게 결합하도록 함

M(Model): 동일

- 네트워크, data(로컬데이터) 등 담당

V(View): 액티비티/프래그먼트가 이제는 뷰의 일부로 간주 됨, 액티비티가 뷰 인터페이스를 구현해서 P(Presenter)가 코드를 만들 인터페이스를 갖도록 함, 이렇게 하면 특정 뷰와 결합되지 않고 가상 뷰를 구현해서 간단한 유닛테스트를 진행할 수 있음

- 완전한 View형태를 가지도록 설계

P(Presenter): Controller와 비슷하지만 뷰에 연결되는 것이 아니라 그냥 인터페이스라는 점이 다름, MVC가 가진 테스트 가능성 문제와 함께 모듈화/유연성 문제 역시 해결 됨, 프리젠티가 절대로 안드로이드 API나 코드라도 참조해선 안된다고 주장하기도 함

- 계산을 하거나 데이터를 가져와서 가고하고 view에게 다시 전달하는 행위

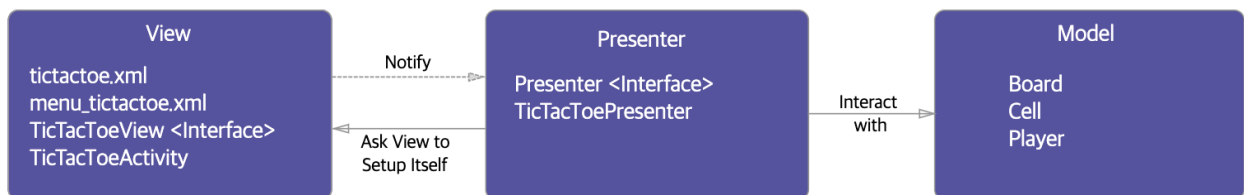


액티비티를 프리젠티에 묶지 않고 작업을 수행하면 액티비티가 구현할 인터페이스를 생성해야 함, 테스트에서는 이 인터페이스를 기반으로 한 가상 객체를 만들어서 프리젠티의 뷰와 상호작용을 테스트 함

MVC보다는 깔끔한 형태, 안드로이드 고유의 뷰와 API에 연결되지 않으므로 인터페이스를 구현했다면 어떤뷰와도 작업 할수 있어서 프리젠티 로직을 쉽게 테스트 할 수 있음

문제점

- 유지보수: 컨트롤러처럼 프리젠티에도 시간이 지남에 따라 추가 비즈니스 로직이 모이는 경향이 있음, 시간이 흐른 후 개발자는 거대하고 다루기 어려운데다 문제가 발생하기 쉽고 분리하도 어려운 프리젠티를 발견하기 됨 (MVVM 에서 해결)

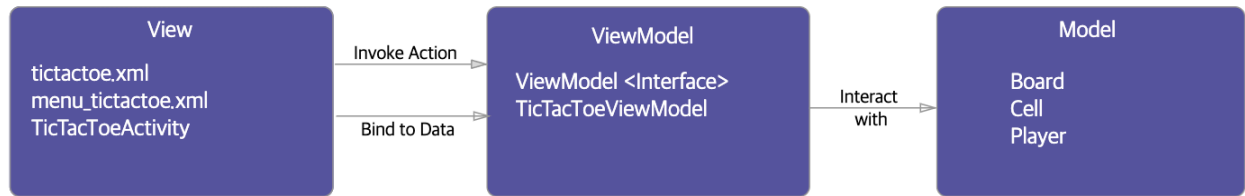


MVVM

M:동일 함

V: 뷰모델에 의해 보여지는 옵저버블 변수와 액션에 유연하게 바인딩 됨

VM(ViewModel): 뷰모델은 모델을 래핑하고 뷰에 필요한 옵저버블 데이터를 준비, 뷰가 모델에 이벤트를 전달할 tn 있도록 훅(hook)을 준비, 그러면서도 뷰모델이 뷰에 종속되지는 않음



뷰에 대한 의존성이 전혀 없어지므로 유닛테스트가 더 쉬워짐, MVP 패턴에서처럼 테스트를 위한 가상 뷰를 만들 필요 없이 테스트 할때 모델이 변경되는 시점에 옵저버블 변수가 제대로 설정됐는지 확인하면 됨

문제점

- 유지관리: 뷰가 변수와 표현식 모두에 바인딩 될수 있으므로 시간이 지남에 따라 관계없는 프리젠테이션 로직이 늘어나 XML에 코드를 추가하게 될수 있음, 이를 방지하려면 뷰 바인딩 표현식에서 값을 계산하거나 파생하지 말고 항상 뷰 모델에서 직접 값을 가져오는것이 좋음, 이방식으로 유닛 테스트 가능함

결론

MVC에 비해 MVP와 MVVM은 앱을 보다 모듈화 하고 구성 요서를 단일 용도로 분해한다는 점에서 발전된 모습이지만, 이 구조 때문에 앱이 더 복잡해질 수 있음, 한 두개의 화면으로만 구성된 간단한 앱이라면 MVC만으로도 충분 함, 한편 데이터 바인딩을 사용하는 MVVM은 보다 빠른 프로그래밍 모델을 따르고 적은 코드를 사용한다는 점에서 매력 적임

세미나2

TDD를 적용하는 것은 높은 수준의 프로그래밍 능력을 요구한다.

실패하는 테스트 케이스를 만들고 그것을 성공하도록(implement)한다. 설계에 대한 정답은 없지만 새로 구현된 코드는 기존 코드와 합쳐져 리팩토링을 해야한다. 기본적인 디자인패턴 지식과 예쁜? 코드를 만들도록 기술을 익혀야 한다.

이런과정을 잘 이해하고 있다면 TDD로 접근하는 것이 빠르겠지만 나는 익숙하지 않다. $\pi\pi$

기능에 대한 테스트 케이스

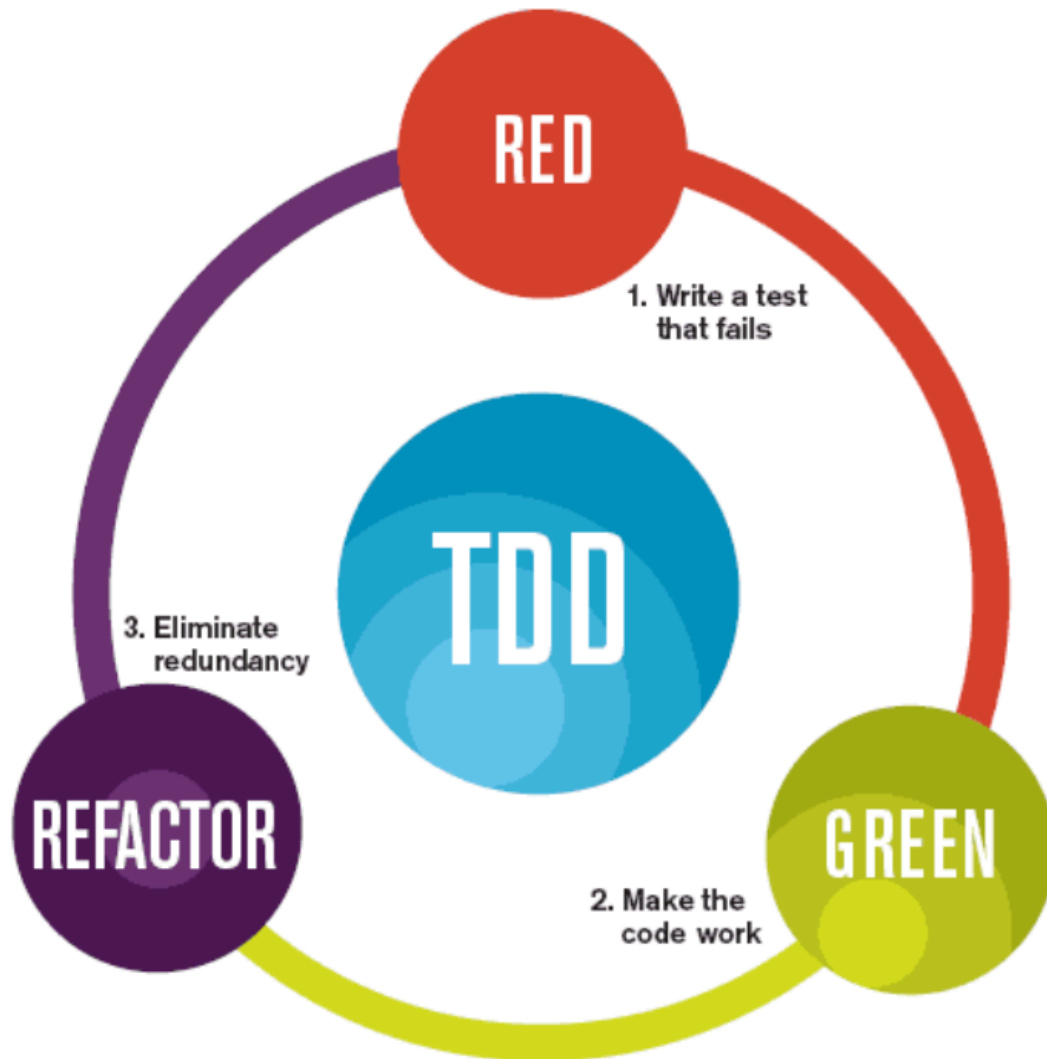
인터페이스 정의

- TDD : 테스트 주도 개발
- 업무코드를 짜기전에 테스트코드를 먼저 만드는것
- 예상결과를 코드로 표현해 놓고 해당 코드를 업무코드에 적용시킴
- 잘 동작하는 깔끔한 코드
- 일반적으로 TDD에서 말하는 단위테스트는 메소드 단위의 테스트

아래 세 단계가 반복적으로 이루어 진다.

질문 -> 응답 -> 정제 반복

- 테스트 수행 결과 실패
- 테스트 통과하는 코드 작성 후 테스트 성공
- 리팩토링
- 다음 질문 계속 진행...



The mantra of Test-Driven Development (TDD) is “red, green, refactor.”

MVP

코드의 구현을 분리하함으로써 복잡도를 감소시키고 함수의 생산성을 향상시킬 수 있는 효율적인 패턴

M(Model): 내부 구성을 담당하여 로직과 개념을 담당

- 네트워크, data(로컬데이터) 등 담당

View : 유저들에게 직접 보여지는 인터페이스로 주로 인터페이스의 정의 구현과 같은 역할

- 완전한 View형태를 가지도록 설계

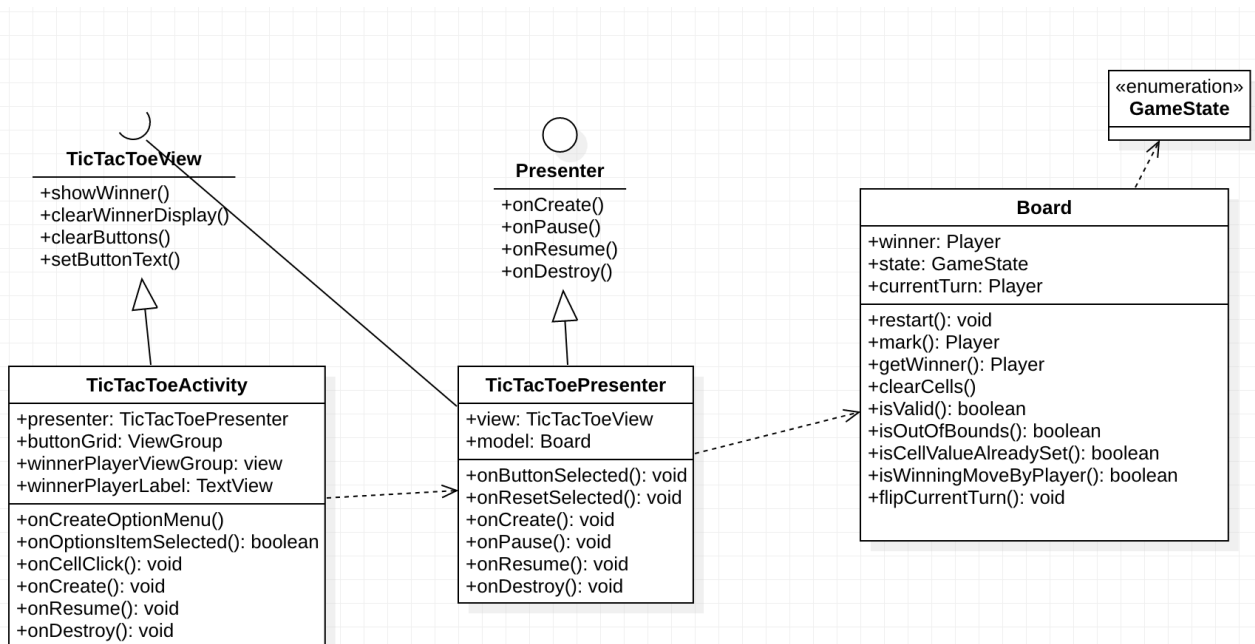
Presenter : 사용자의 UI 행위를 추상화한 메서드로 기능을 정의하는 역할

- 계산을 하거나 데이터를 가져와서 가고하고 view에게 다시 전달하는 행위



<모델과 뷰의 의존성 제거>

1. view로 사용자의 입력이 들어온다.
 - X 또는 O 게임 시작
2. view는 presenter에게 작업을 요청한다.
 - 버튼을 순서대로 터치한다.
3. presenter는 필요한 데이터를 Model에 요청에 대한 응답을 받아 view로 데이터를 응답한다.
 - 사용자 터치에 대한 로직을 처리한다.
4. view는 응답받은 데이터를 화면에 보여준다.
 - Winner가 누구인지 보여준다.



액티비티를 프리젠터에 묶지 않고 작업을 수행하면 액티비티가 구현할 인터페이스를 생성해야 함, 테스트에서는 이 인터페이스를 기반으로 한 가상 객체를 만들어서 프리젠터의 뷰와 상호작용을 테스트 함

MVC보다는 깔끔한 형태, 안드로이드 고유의 뷰와 API에 연결되지 않으므로 인터페이스를 구현했다면 어떤뷰와도 작업 할수 있어서 프리젠터 로직을 쉽게 테스트 할 수 있음

1. Mockito, Espresso
2. 테스트 케이스 작성 및 UI 테스트 진행(빙고게임)

Unit Testing의 목표중 하나는 테스트가 독립적이어야 하며 다른 클래스에 side effect를 주면 안된다.

1. Mockito, Espresso

Mockito

@RunWith(MockitoJUnitRunner.class

- Mock객체를 사용하기 위해서 지정 @Mock private User user;

@Mock mock인스턴스를 생성하여 Mock객체를 만들었는데 @mock어노테이션을 이용해서 mock객체 생성
User user = mock(ArrayList.class);

verify()

- mock 객체에 대한 원하는 메소드가 특정조건으로 실행되었는지 검증 verify(T mock).method();

ex)

List.mockedList = mock(List.class);

mockedList.add("one"); mockedList.clear();

verify(mockedList).add("one"); verify(mockedList).clear();

verify(T mock, VerificationMode mode).method();

atLeastOnce(); 적어도 한번 수행했는지 검증 atLeast(int n); 적어도 n번 수행했는지 검증 times(int n); 무조건 n번 수행했는지 검증(n보다 크거나 작으면 오류로 간주) atMost(int n); 최대한 n번 수행했는지 검증 never(); 수행되지 않았는지 검증(수행했으면 오류로 간주)

when()

제대로 된 mock의 역할을 수행하기 위해서는 원하는 값을 리턴하는 기능 When()메소드는 Mock이 감싸고 있는 메소드가 호출되었을 때 Mock 객체의 메소드를 위해 선언할때 사용 (OngoingStubbing 인터페이스 리턴)

thenReturn(Answer<?>answer) theCallRealMethod() thenReturn(T value) thenReturn(T value, T... values) thenThrow(java.lang.Throwable...throwables)....

inOrder

메소드에 호출 시 넘긴 값 뿐만 아니라 메소드 호출 순서도 검증


inOrder.verify(mock.method()) 형식으로 사용

Espresso

실제처럼 단위 테스트용 클래스 만들기

ActivityTestRule을 이용한 테스트케이스 작성법(구글에서 권장)

onView(**Matcher**)
 .perform(**ViewAction**)
 .check(**ViewAssertion**)



Matchers

USER PROPERTIES
withId(...)
withText(...)
withTagKey(...)
withTagValue(...)
hasContentDescription(...)
withHint(...)
withSpinnerText(...)
hasLinks()
hasEllipsizedText()
hasMultilineText()

UI PROPERTIES
isDisplayed()
isCompletelyDisplayed()
isEnabled()
hasFocus()
isClickable()
isChecked()
isNotChecked()
withEffectiveVisibility(...)
isSelected()

COMMON HAMCREST MATCHERS
allOf(Matchers)
anyOf(Matchers)
is(...)
not(...)
endsWith(String)
startsWith(String)

HIERARCHY
withParent(Matcher)
withChild(Matcher)
hasDescendant(Matcher)
isDescendantOfA(Matcher)
hasSibling(Matcher)
isRoot()

INPUT
supportsInputMethods(...)
hasImeAction(...)

CLASS
isAssignableFrom(...)
withClassName(...)

ROOT MATCHERS
isFocusable()
isTouchable()
isDialog()
withDecorView(...)
isPlatformPopup()

SEE ALSO
Preference matchers
Cursor matchers

View Actions

CLICK/PRESS
click()
doubleClick()
longClick()
pressBack()
pressImeActionButton()
pressKey([int/EspressoKey])
pressMenuKey()
closeSoftKeyboard()
openLink(...)

GESTURES
scrollTo()
swipeLeft()
swipeRight()
swipeUp()
swipeDown()

TEXT
clearText()
typeText(String)
typeTextIntoFocusedView(String)
replaceText(String)

View Assertions

MATCHES
matches(Matcher)
doesNotExist()
selectedDescendantsMatch(...)

LAYOUT ASSERTIONS
noEllipsizedText(Matcher)
noMultilineButtons()
noOverlaps([Matcher])

POSITION ASSERTIONS
isLeftOf(Matcher)
isRightOf(Matcher)
isLeftAlignedWith(Matcher)
isRightAlignedWith(Matcher)
isAbove(Matcher)
isBelow(Matcher)
isBottomAlignedWith(Matcher)
isTopAlignedWith(Matcher)

2. 테스트 케이스 시나리오

Unit Testing의 목표중 하나는 테스트가 독립적이어야 하며 다른 클래스에 side effect를 주면 안된다.

Model, View, Presenter 동일한 내용의 테스트 케이스

1. X 플레이어가 승리하는 시나리오
2. O 플레이어가 승리하는 시나리오

Model :: junit 사용

- assertNull(), assertEquals() 사용하여 결과값 체크

Presenter :: junit + mockito 사용

- When/thenReturn 값으로 초기값 설정
- verify() 를 이용하여 mock 객체에 대한 원하는 메소드가 특정조건으로 실행되었는지 검증

View :: espresso 사용

- @Rule 해당 Activity를 사용하기 위한 어노테이션
- onView(Matcher).perform(ViewAction).check(ViewAssertion)

세미나 3

MVP에 대해서 다시 한번 알아보겠습니다.

view에서 발생하는 이벤트를 직접 핸들링하는 것이 아니라 presenter에 위임하도록 한다.

view.java

- Activity.java에서 presenter.funtion(); 호출

위임하는 방법은 액티비티가 뷰 인터페이스를 구현해서

presenter에서 코드를 만들 인터페이스를 갖도록 하면 된다.

presenter.java

- private LoginConstants.View view;
- view.fucntion();

특정뷰와 결합하지 않고 가상 뷰를 구현해서 간단한 유닛테스트를 실행할수 있다.

presenter는 controller와 같지만 뷰에 연결되는 것이 아니라 인터페이스로 연결되는 점이 다르다.

이에 따라 mvc가 가진 테스트 가능성 문제와 모듈화/유연성 역시 해결된다.

presenter

뷰와 모델 사이에서 data를 전달 역할을 해준다.

repository(model)

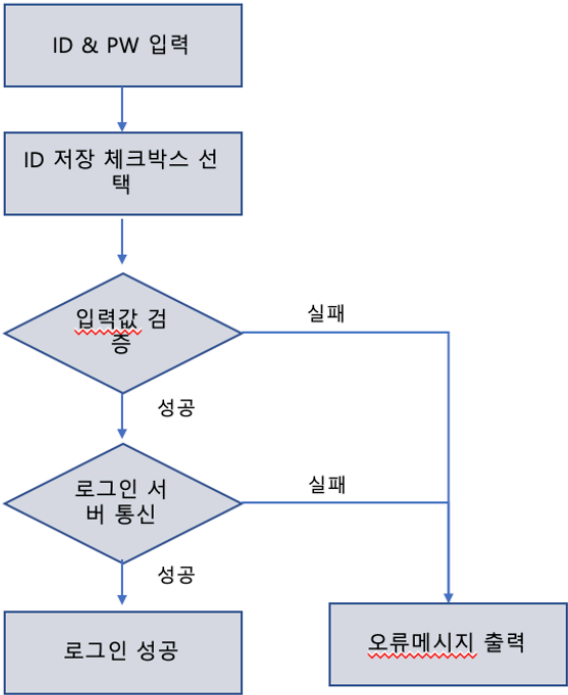
앱 데이터 및 상태에 대한 비즈니스 로직을 수행한다.

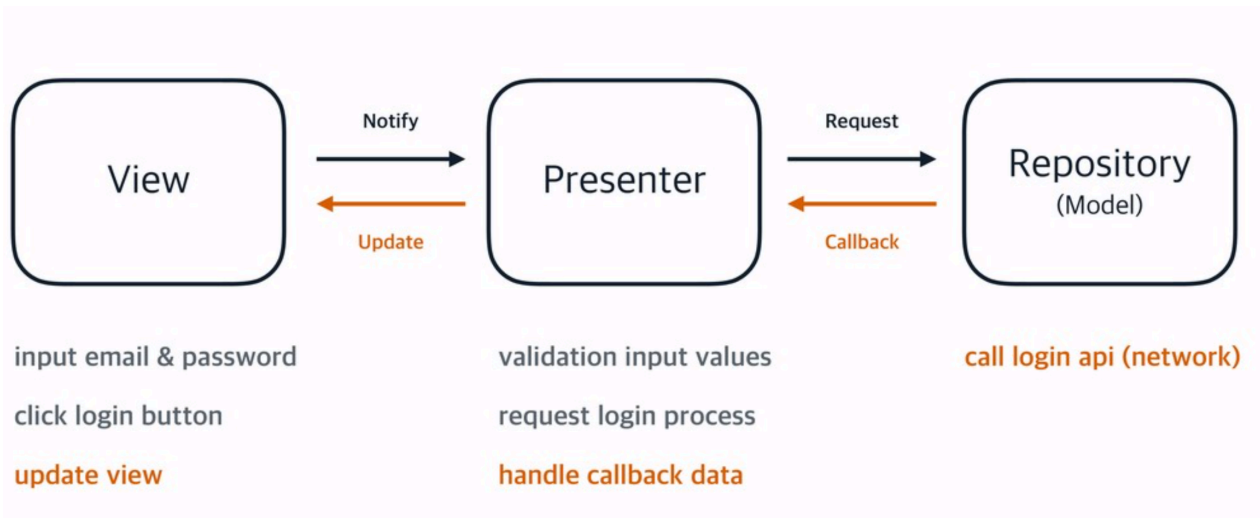
MVC와 비슷하지만 View와 Model을 완전히 분리하여 사용하기 위해서이다

분리된 View와 Model은 각각 테스트 케이스를 진행 할 수 있다.

원가드 어드민 로그인 기능에 대한 TDD 적용 사례

. 로그인 플로우 차트





View

- 에러 팝업띄우기, 서버재설정 팝업 띄우기
- 서버 설정 재시작 시 어플리케이션 재시작
- 입력필드 초기화
- 로그인 완료시 메인(메뉴)화면으로 이동
- ID저장 체크 값 가져오기
- ID/PW/CompanyKey 가져오기, 세팅하기
- 입력 키패드 내리기
- 로딩 다이얼로그 팝업(프로그래스바) 보이기/ 숨기기

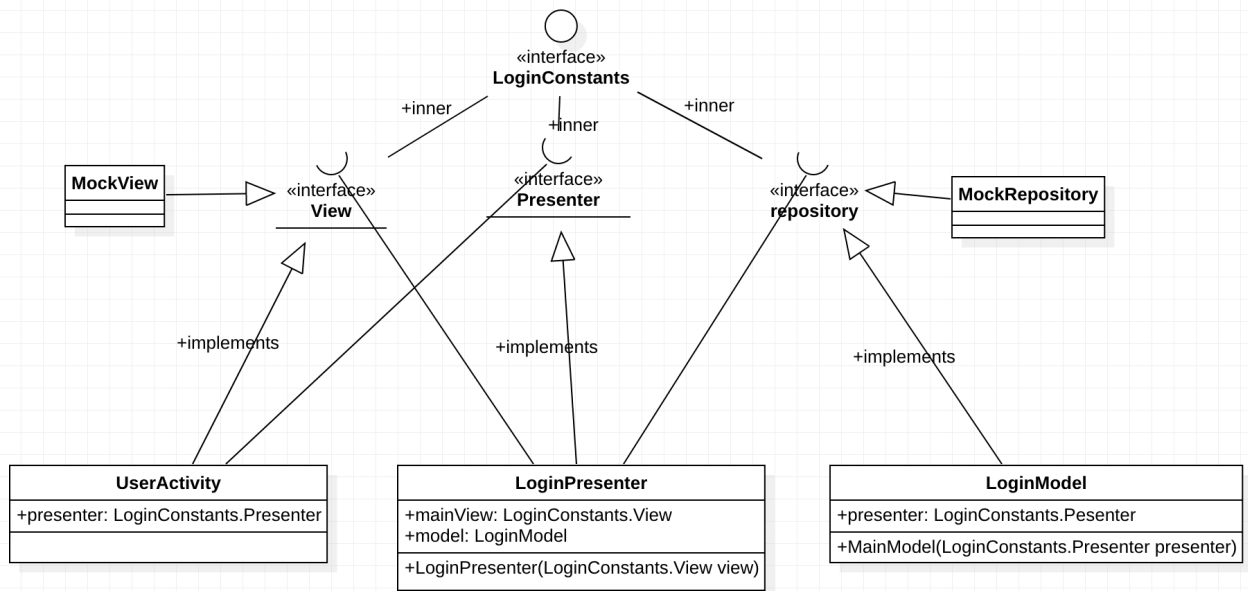
Presenter

- 로그인 로직
- ID 저장 상태 값에 따른 로직
- 서버 재설정 시 로직 처리

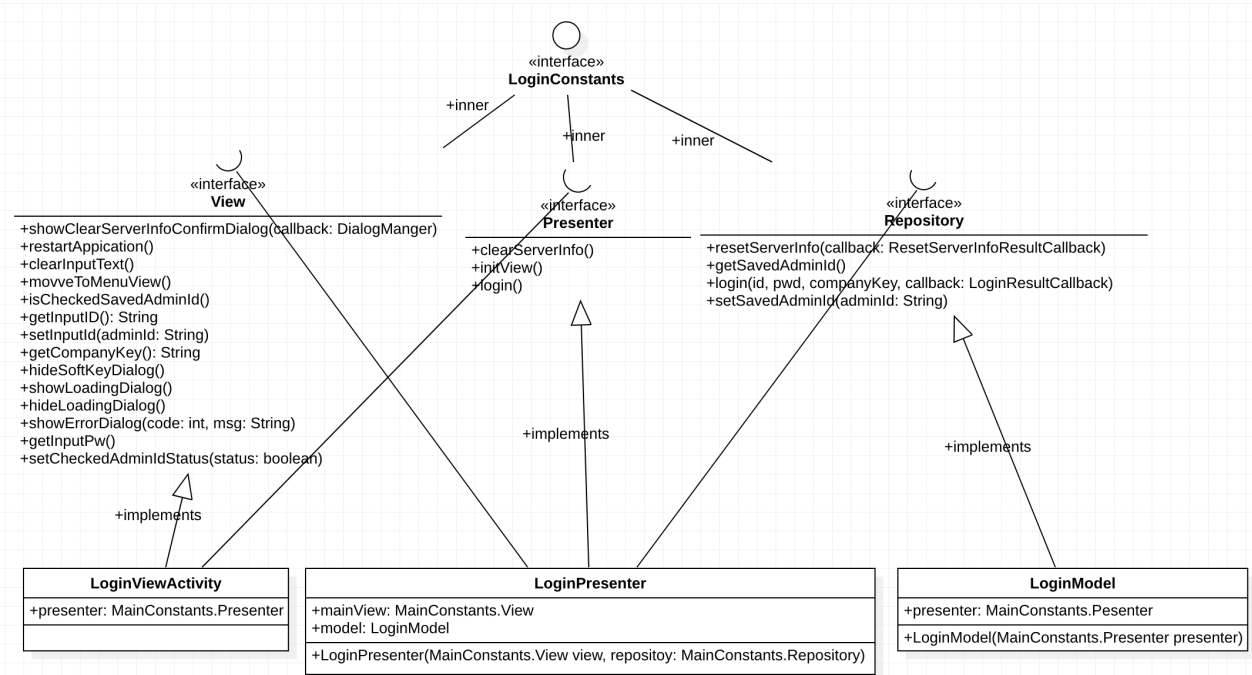
Repository

- 서버 재설정 시 값 초기화
- 어드민 ID 저장/가져오기
- 로그인 서버통신

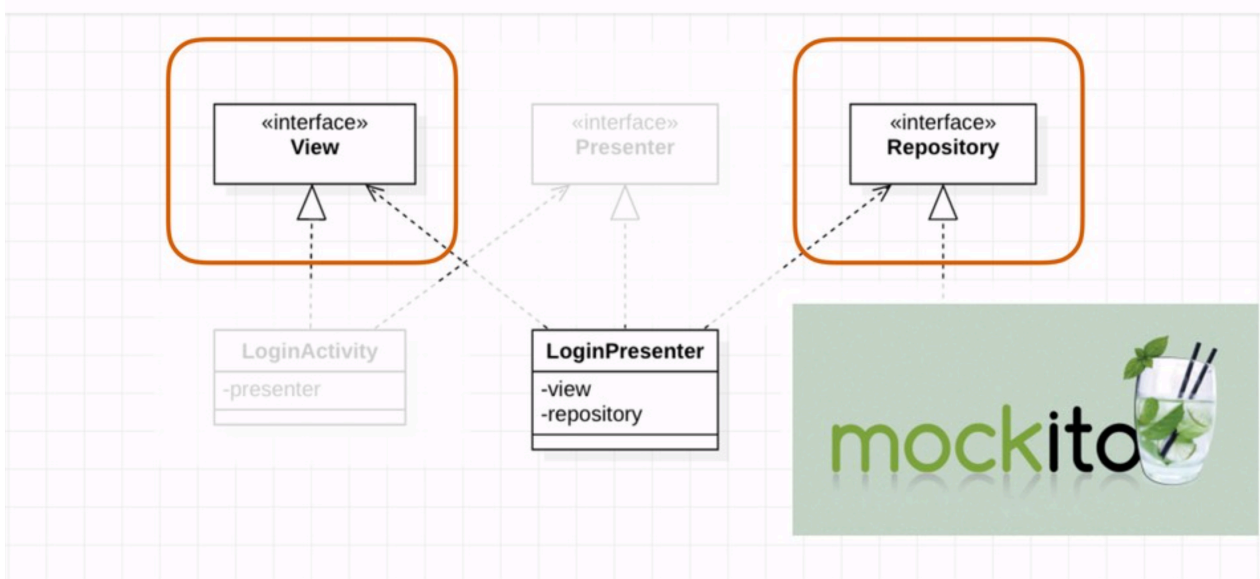
• MVP 로그인 ClassDiagram(기본)



• MVP 로그인 ClassDiagram(완성)



• Mockito 적용 (view, repository)하여 TDD 실습



테스트 코드 작성 전 초기 변수 설정

```

@Mock
private LoginConstants.View view;

private LoginConstants.Repository repository;

private LoginConstants.Presenter presenter;

@Before
public void setUp() {
    // 테스트 케이스 시 가짜 repository와 연결
    repository = mock(LoginFakeModel.class);

    presenter = new LoginPresenter(view, repository);
}
  
```

테스트 케이스 실습

1. ID/PW 초기값 설정

```

when(view.getInputID()).thenReturn("admin");
when(view.getInputPW()).thenReturn("12345");
when(view.getCompanyKey()).thenReturn("1");
  
```

2. 체크박스 초기 설정

```

when(view.isCheckedSavedAdminId()).thenReturn(true);
  
```

3. 로그인 수행

```

resenter.login();
  
```

4. 입력값 검증

```

assertTrue("ID 유효성 검증 실패", Validator.isValidData(view.getInputID()));
assertTrue("PW 유효성 검증 실패", Validator.isValidData(view.getInputPW()));
  
```

```

5. 로딩다이얼로그 보이기
verify(view).showLoadingDialog();

6. 키패드 숨김
verify(view).hideSoftKeyboard();

7. 로그인 비즈니스 로직 처리
inOrder.verify(repository).doLogin(eq(id),eq(pwd),eq(companyKey),
loginResultListener.capture());

8. 로그인 성공 콜백
loginResultListener.getValue().onSuccess();

9. 어드민 ID 저장
verify(repository).setSavedAdminId(anyString());

10. 입력필드값 초기화
verify(view).clearInputText();

11. 로딩다이얼로그 숨김
verify(view).hideLoadingDialog();

```

View

```

@Override
public void clearInputText() {
    mHandler.sendMessageDelayed(GO_CLEAR_TEXT, Property.DELAY_TIME);
}

@Override
public void showClearServerInfoConfirmDialog(final
DialogManager.OnDialogCompleted callback) {
    DialogManager.showDialog(this, " 서버주소를 재설정 하시겠습니까?", new
DialogManager.OnDialogCompleted() {

        @Override
        public void onCompleted() {
            Toast.makeText(getApplicationContext(), "서버주소가 리셋되었습니다. 앱을 재
실행해 주세요.", Toast.LENGTH_SHORT).show();
            callback.onCompleted();
        }

        @Override
        public void onCancelled(boolean isClickButton) {
            callback.onCancelled(isClickButton);
        }
    });
}

```



```

        }, true);
    }

    @Override
    public void restartApplication() {
        finish();
    }

    @Override
    public String getInputID() {
        return editLoginID.getText().toString();
    }

    @Override
    public void setInputID(String adminId) {
        editLoginID.setText(adminId);
    }

    @Override
    public String getInputPW() {
        return editLoginPWD.getText().toString();
    }

    @Override
    public String getCompanyKey() {
        if (companyCodeList.length() > 0 && companyListAdapter != null) {
            return arrCompanyCodeKeys[companyListAdapter.getSelectedIndex()];
        }
        return "";
    }

    @Override
    public void hideSoftKeyboard() {
        InputMethodManager imm = (InputMethodManager)
getSystemService(INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(editLoginID.getWindowToken(), 0);
        imm.hideSoftInputFromWindow(editLoginPWD.getWindowToken(), 0);
    }

    @Override
    public void showLoadingDialog() {
        customDialogProgress = new CustomDialogProgress(this,
getString(R.string.app_name), getString(R.string.txt_progress_message_wait));
        customDialogProgress.show();
    }

    @Override
    public void hideLoadingDialog() {

```

```

        if (customDialogProgress != null) {
            customDialogProgress.dismiss();
        }
    }

    @Override
    public void showErrorDialog(int errorCode, String errorMessage) {
        Bundle bundle = new Bundle();
        Message msg = new Message();
        bundle.putString(ERR_MESSAGE, errorMessage);
        msg.setData(bundle);
        msg.what = errorCode;
        mHandler.sendMessage(msg);
    }

    @Override
    public void setCheckedAdminIdStatus(boolean status) {
        sw_idsave.setChecked(status);
    }

    @Override
    public void moveToMenuView() {
        Intent intent = new Intent(LoginViewActivty.this, ActivityMenu.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP |
Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);
        intent.putExtra("agentVersionCheck",true);
        startActivity(intent);
        finish();
    }

    @Override
    public boolean isCheckedSavedAdminId() {
        return sw_idsave.isChecked();
    }
}

```

Presenter

```

private LoginConstants.View view;
private LoginConstants.Repository repository;

public LoginPresenter(LoginConstants.View view, LoginConstants.Repository
repository) {
    this.view = view;
    this.repository = repository;
}

```

```

@Override
public void clearServerInfo() {
    view.showClearServerInfoConfirmDialog(new
DialogManager.OnDialogCompleted() {
        @Override
        public void onCompleted() {
            repository.resetServerInfo(new ResetServerInfoResultCallback() {
                @Override
                public void onCompleted() {
                    view.restartApplication();
                }
            });
        }

        @Override
        public void onCancelled(boolean isClickButton) {
            // do Nothing...
        }
    });
}

@Override
public void initView() {
    // ID저장 유무에 따른 처리
    String savedAdminId = repository.getSavedAdminId();
    Log.d(TAG, "savedAdminId: "+savedAdminId);
    if (!StringUtil.isNull(savedAdminId) && savedAdminId.trim().length() != 0)
    {
        view.setInputID(savedAdminId);
        view.setCheckedAdminIdStatus(true);
    } else {
        view.setInputID("");
        view.setCheckedAdminIdStatus(false);
    }
}

@Override
public void login() {
    final String adminId = view.getInputID();
    String pwd = view.getInputPW();
    String companyKey = view.getCompanyKey();

    Log.d(TAG, "adminId: "+adminId+",pwd: "+pwd+", companyKey: "+companyKey);

    if (!Validator.isValidData(adminId)) {
        view.showErrorDialog(LoginViewActivty.REQUEST_ERR_MESSAGE,"ID를 입력해주세요.");
    }
}

```

```

        return;
    }

    if (!Validator.isValidData(pwd)) {
        view.showErrorDialog(LoginViewActivty.REQUEST_ERR_MESSAGE, "PW를 입력해주세요.");
        return;
    }

    view.showLoadingDialog();
    view.hideSoftKeyboard();
    repository.doLogin(adminId,pwd,companyKey, new LoginResultCallback() {
        @Override
        public void onSuccess(LoginItem item) {
            boolean status = view.isCheckedSavedAdminId();
            if (status) {
                repository.setSavedAdminId(adminId);
            } else {
                repository.setSavedAdminId("");
            }
            view.clearInputText();
            view.moveToMenuView();
            view.hideLoadingDialog();
        }

        @Override
        public void onFail(int code, String msg) {
            Log.d(TAG,"onFail code "+code+", message "+msg);
            view.clearInputText();
            view.showErrorDialog(code, msg);
            view.hideLoadingDialog();
        }
    });
}

```

Model

```

@Override
public void resetServerInfo(final ResetServerInfoResultCallback callback) {
    SharedPreferencesManager.setServerSave(BaseContext.getContext(), "");
    callback.onCompleted();
}

@Override
public void setSavedAdminId(String userId) {
    SharedPreferencesManager.setPrefUserId(BaseContext.getContext(), userId);
}

```

```

// 저장된 로그인
@Override
public String getSavedAdminId() {
    return SharedPreferencesManager.getPrefUerID(BaseContext.getContext());
}

@Override
public void doLogin(final String id, final String pwd, final String
companyKey, final LoginResultCallback callback) {
    StringBuffer param = new StringBuffer();
    try {
        if (!StringUtil.isNull(companyKey)
            && !StringUtil.isEmpty(companyKey)) {
            param.append("companyKey=").append(companyKey).append("&");
        }
        param.append("id=")
            .append(id)
            .append("&password=")
            .append(pwd)
            .append("&authType=ADMIN");
        Log.d(TAG, "performLoginOperation param: "+param);
    } catch (Exception e) {
        Log.e(TAG, " Exception " + e.toString());
    }

    RaonHttpRequest httpRequest = new
RaonHttpRequest(BaseContext.getContext(), HttpRequestConstant.URI_ADMIN_LOGIN,
param.toString(), new RaonHttpRequest.IHttpRequest() {
        @Override
        public void callBackResponse(String strResponse, String strURI) {
            if (strResponse.equals(String.valueOf(HttpURLConnection.HTTP_OK)))
            {
                LoginItem userCredentials = new LoginItem(id,pwd,companyKey);
                callback.onSuccess(userCredentials);
            } else {
                callback.onFail(LoginViewActivty.REQUEST_ERR_MESSAGE, "로그인에
실패하였습니다. id,pw를 정확히 입력해주세요.");
            }
        }
    });
    httpRequest.start();
}

```