



잠실개발자

Refactoring

M E E T U P

CONTENTS

1. 개발환경설정
2. Legacy Code 소개
3. Mock Framework를 활용한 Refactoring (Pair Programming)
4. Pattern을 활용한 Refactoring (Mob Programming)
5. 회고

잠실개발자
Refactoring
MEETUP



★
★ 유병현(삼성SDS)
SW★엔지니어링팀 / 코드품질그룹

- 전문영역 : 코드품질, 변경영향분석
 - . Code Quality, Code Review, Refactoring
 - . Software Architecture Analysis

byunghyun.yu@samsung.com

오늘 할 일

Legacy Code Test Case 작성 - 1부

- Test Case Coverage 100% 완성
- Testable 클래스를 만들어 Test Case 작성
- Mock 프레임워크를 사용하지 않고 Test Case 작성

Legacy Code Refactoring – 2부

- 지난 시간 복습 : **Testable 클래스** 작성해보기
- 좀 더 똑똑하게 Test Case 작성하기 : **Mock Framework** 사용하기
- 디자인 패턴 적용하여 Legacy Code Refactoring : **Observer 패턴**



개발환경설정

실습환경

소스 다운로드 1 : <https://github.com/CODARI-JAMSIL/RestaurantBooking>
(master 브랜치)

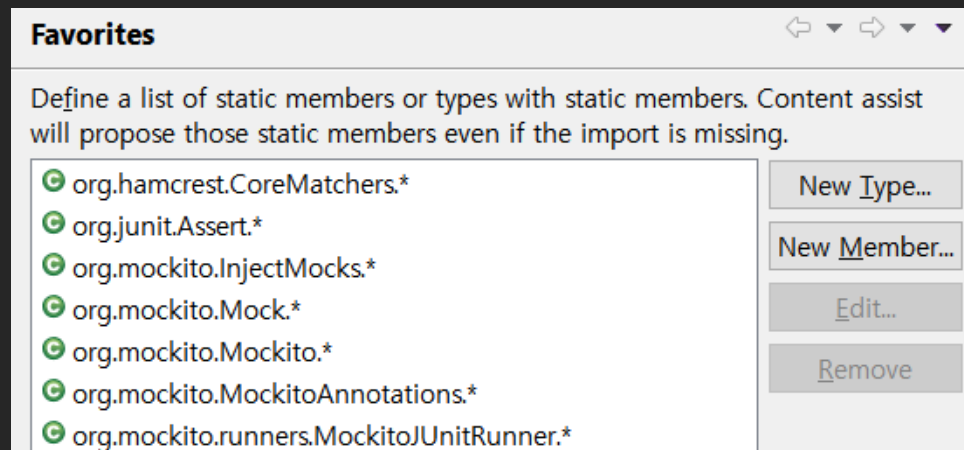
Static import 추가

Windows -> Preferences 메뉴

Java -> Editor -> Content Assist -> Favorites 항목

“New Type...” 버튼 : “ 클릭하여 아래내용 추가

```
org.hamcrest.CoreMatchers
org.junit.Assert
org.mockito.InjectMocks
org.mockito.Mock
org.mockito.runners.MockitoJUnitRunner
org.mockito.Mockito
org.mockito.MockitoAnnotations
```



실습환경

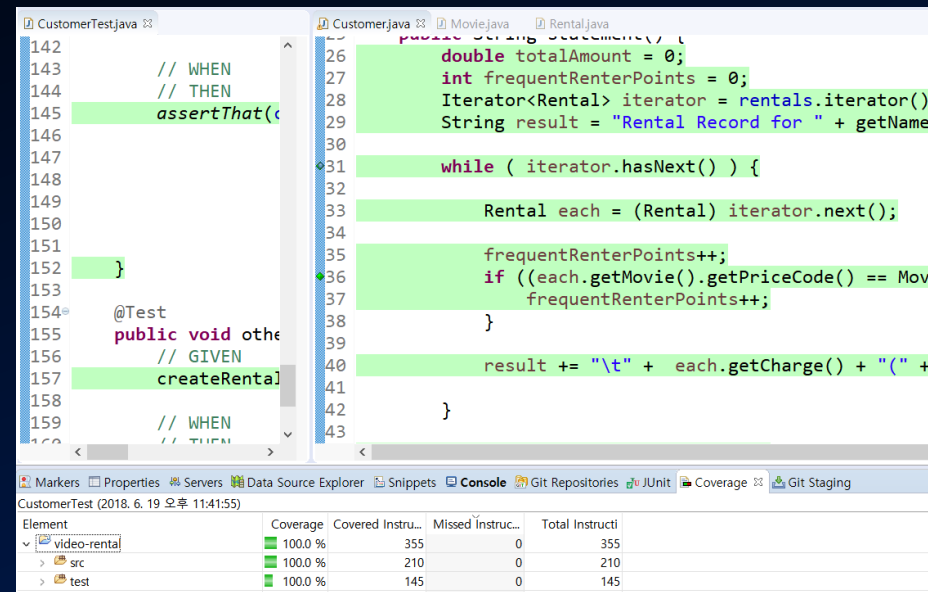
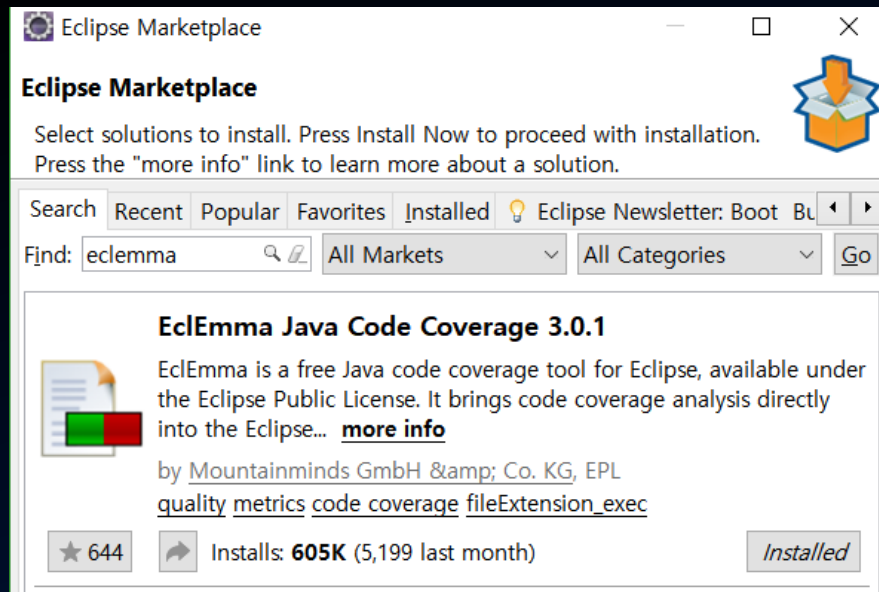
소스 다운로드 2 : <https://github.com/CODARI-JAMSIL/MockitoTraining>
(master 브랜치)

실습환경

테스트케이스 커버리지 : Eclemma

이클립스 - Help

- Eclipse Marketplace – **eclemma** – 설치 - 재기동





Legacy Code 소개

Restaurant Booking



레스토랑 예약 시스템

- BookingScheduler를 통해 시간대별 예약관리
- 예약은 정시에만 가능하다.
 - ex. 09:00(0), 09:03(x)
- 시간대별 수용가능 인원을 정할 수 있다.
 - 모든 시간대에 동일한 인원수 적용
- 일요일은 예약이 불가하다.
 - ex. '20180916(일)'에 '20180917(월)' 이용 예약 불가
 - ex. '20180917(월)'에 '20180923(일)' 이용 예약 가능
- 예약완료 시 SMS발송
- 이메일 주소가 있는 경우는 메일 발송



Mock Framework를 활용한 Refactoring (Pair Programming)

지난 미팅에 한 일

- Test Case Coverage 100% 완성
- Testable 클래스를 만들어 Test Case 작성
- Mock 프레임워크를 사용하지 않고 Test Case 작성

테스트를 하는 목적은

Software Testing이 필요한 이유는? 품질 = 돈

- 소프트웨어는 인간이 만든 어떤 제품보다 비즈니스에 더 큰 악영향을 끼칠 수 있다.
- 나쁜 소프트웨어의 품질 이슈는 인간 역사상 가장 비싼 이슈이다.
(미국에서 연간 1500억 달러이상, 전세계적으로 연간 5000억 달러 이상)
- 좋지 않은 소프트웨어 품질로 인해 취소된 프로젝트는 동일규모, 동일유형의 프로젝트 보다 15%이상 더 많은 비용을 사용한다.
- 소프트웨어의 품질을 향상 시키는 것은 모든 산업계에서 가장 중요시 되는 문제이다.
- 낮은 품질을 유지하는 것은 코딩 단계까지는 더 낮은 비용이 들지만,
결국 테스트와 유지보수 단계로 가면 더 높은 비용이 들게 된다.

출처 : IBM Rational Technical Summit 2013

Capers Jones, 2010

테스트의 분류

1. 유형별 분류

1. BlackBox Test : 프로젝트 후반에 가능, **높은 결함 검출율**
2. WhiteBox Test : 프로젝트 초반에 가능, **낮은 결함 검출율**

2. 단계별 분류

1. Design Review
2. Code Review, Code Inspection
3. Unit Testing
4. Integration Testing
5. System Integration Testing
6. User Acceptance Testing
7. Operational Acceptance Testing

저비용

문제점 발생시
해결하는데 드는 비용

고비용

테스트 케이스란

IEEE Standard 610 (1990) 의 정의

- "A set of **test inputs**, **execution conditions**, and **expeted result**
Developed for a particular objective, such as to exercise a particular
Program path or to verify compliance with a specific requirement."
- 테스트 케이스의 주요 요소
 - **응용프로그램 또는 시스템의 사전 실행조건, 상태**
 - **입력값**
 - **예상결과 값**
- 테스트 자동화의 핵심은
 - **예상 결과 값을 아는 것!**

Triple A

- Triple A (Arrange, Act, Assert)

```
@Test
public void 정시에_예약할_경우_예약성공() {
    // Arrange
    Schedule schedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_MAIL);

    // Act
    booking.addSchedule(schedule);

    // Assert
    assertThat(booking.hasSchedule(schedule), is(true));
}
```

Quadruple A

- Quadruple A (Arrange, Act, Assert, Assert)

```
@Test
public void 정시에_예약할_경우_예약성공() {
    // Arrange
    Schedule schedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_MAIL);

    // Assert
    assertNotNull(schedule.getCustomer());

    // Act
    booking.addSchedule(schedule);

    // Assert
    assertThat(booking.hasSchedule(schedule), is(true));
}
```

반드시 AAA, AAAA 여야 하나?

```
@Test
public void 정시에_예약할_경우_예약성공() {
    // Arrange
    Schedule schedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_MAIL);

    // Act
    booking.addSchedule(schedule);

    // Assert
    assertThat(booking.hasSchedule(schedule), is(true));
}
```

```
@Test
public void 스케줄_생성() {
    // Arrange
    Schedule schedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_MAIL);

    // Assert
    assertNotNull(schedule);
}
```

```
@Test
public void 스케줄_생성후_예약인원_확인() {
    // Arrange
    Schedule schedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_MAIL);

    // Assert
    assertThat(schedule.getNumberOfPeople(), is(UNDER_CAPACITY));
}
```

```
@Test
public void 스케줄_생성_생성후_고객정보확인() {
    // Arrange
    Schedule schedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_MAIL);

    // Assert
    assertNotNull(schedule.getCustomer());
}
```

위의 정시에_예약할_경우_예약성공
테스트 케이스 전에
왼쪽과 같은 테스트 케이스를 추가하여
검증해 볼 수 있음

Test Case의 효과

1. 수정시의 생산성 향상
2. 버그 잡기가 빨라진다.
3. 버그 잡기가 쉬워진다.
4. 시스템 구조가 좋아진다.
5. 리팩토링의 조건이 된다.
6. 회귀테스트를 제공한다.
7. 하위호환성 보증의 방법을 제공한다.
8. 전체 시스템의 이해 없이 부분의 수정이 가능하다.
9. 샘플로 활용된다.
10. 코드리뷰시에 부담감이 준다.
11. CI가 제대로 활용된다.
12. 설계와 구현을 분리할 수 있다.

테스트 Coverage

```
BookingSchedulerTest.java
39 Schedule schedule= new Schedule(NOT_ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_EMAIL);
40 booking.addSchedule(schedule);
41 fail();
42 }
43
44 @Test
45 public void 정시에_예약할_경우_예약성공() {
46 // Arrange
47 Schedule schedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_EMAIL);
48
49 // Act
50 booking.addSchedule(schedule);
51
52 // Assert
53 assertThat(booking.hasSchedule(schedule), is(true));
54 }
55
56 @Test
57 public void 시간대별_인원제한() {
58 Schedule fullSchedule= new Schedule(ON_THE_HOUR, MAX_CAPACITY, CUSTOMER_WITHOUT_EMAIL);
59 booking.addSchedule(fullSchedule);
60
61 try {
62 Schedule newSchedule= new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT_EMAIL);
63 booking.addSchedule(newSchedule);
64 fail();
65 } catch (RuntimeException e) {
66 assertThat(e.getMessage(), is("Number of people is over restaurant capacity per hour"));
67 }
68 }
69
70 @Test
71 public void 시간대가_다르면_예약가능() {
72 Schedule schedule= new Schedule(ON_THE_HOUR, MAX_CAPACITY, CUSTOMER_WITHOUT_EMAIL);
73 booking.addSchedule(schedule);
74
75 DateTime anotherTime= ON_THE_HOUR.plusHours(UNDER_CAPACITY);
76 }
```

```
BookingScheduler.java
1 package com.sds.cleancode.restaurant;
2
3 import java.util.ArrayList;
4
5 public class BookingScheduler {
6     private int capacityPerHour;
7     private List<Schedule> schedules;
8     private SmsSender smsSender;
9     private MailSender mailSender;
10
11     public BookingScheduler(int capacityPerHour) {
12         this.schedules = new ArrayList<Schedule>();
13         this.capacityPerHour = capacityPerHour;
14         this.smsSender = new SmsSender();
15         this.mailSender = new MailSender();
16     }
17
18     public void addSchedule(Schedule schedule) {
19         // throw an exception when booking time is on the hour.
20         if(schedule.getDateTime().getMinuteOfHour() != 0){
21             throw new RuntimeException("Booking should be on the hour.");
22         }
23
24         // throw an exception when capacity per hour is over
25         int numberOfPeople = schedule.getNumberOfPeople();
26         for (Schedule bookedSchedule : schedules) {
27             if (bookedSchedule.getDateTime().isEqual(schedule.getDateTime())) {
28                 numberOfPeople += bookedSchedule.getNumberOfPeople();
29             }
30         }
31         if (numberOfPeople > capacityPerHour){
32             throw new RuntimeException("Number of people is over restaurant capacity per hour");
33         }
34
35         // throw an exception on sunday
36         DateTime now = getNow();
37         if(now.getDayOfWeek().isSunday()){
38             throw new RuntimeException("Booking is not allowed on Sunday");
39         }
40     }
41 }
```

테스트 케이스, 테스트 Coverage는
리팩토링의 조건이 된다.
(But 100% 신뢰할 순 없음)

Finished after 0.219 seconds

Runs: 9/9 Errors: 0 Failures: 0

com.sds.cleancode.restaurant.BookingSchedulerTest [Run] Failure Trace

- 일요일인_경우_예외처리 (0.016 s)
- 시간대별_인원제한 (0.000 s)
- 예약완료시_sms_발송 (0.000 s)
- email이_없는_경우_mail_미발송 (0.000 s)
- email이_있는_경우_mail_발송 (0.000 s)
- 시간대가_다르면_예약가능 (0.000 s)
- 정시에_예약할_경우_예약성공 (0.000 s)
- 정시에_예약하지_않을시_예외처리 (0.000 s)
- 일요일_아닌경우_예약성공 (0.000 s)

Element	Coverage	Covered Instru...	Missed Instruc...	Total Instructi
RestaurantBooking	100.0 %	169	0	169
src/main/java	100.0 %	169	0	169
com.sds.cleancode.restaurant	100.0 %	169	0	169
BookingScheduler.java	100.0 %	116	0	116
BookingScheduler	100.0 %	116	0	116
BookingScheduler(int)	100.0 %	21	0	21
addSchedule(Schedule)	100.0 %	74	0	74
getNow()	100.0 %	4	0	4
hasSchedule(Schedule)	100.0 %	5	0	5
setMailSender(MailSender)	100.0 %	4	0	4
setSchedules(List<Schedule>)	100.0 %	4	0	4
setSmsSender(SmsSender)	100.0 %	4	0	4
Customer.java	100.0 %	24	0	24

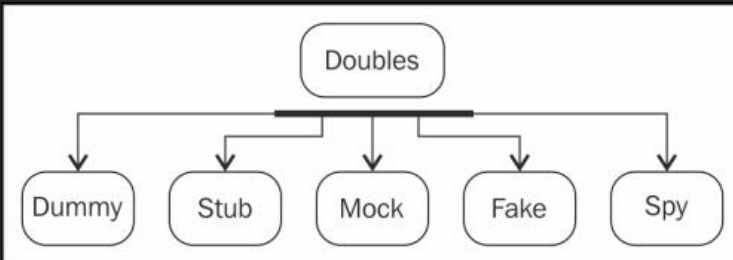
Test Double, Testable 클래스

- 오리지널 클래스의 대역

영어사전 단어·속어 1-5 / 521건

대역 (代役)

(연극·영화의) stand-in (for), understudy; (위험한 연기의) (body) double



```
public class TestableBookingScheduler extends BookingScheduler {
    private String dateTime;
    DateTimeFormatter DATE_TIME_FORMATTER = DateTimeFormat.forPattern("YYYY/MM

    public TestableBookingScheduler(int capacityPerHour, String dateTime) {
        super(capacityPerHour);
        this.dateTime= dateTime;
    }

    @Override
    public DateTime getNow() {
        return DATE_TIME_FORMATTER.parseDateTime(this.dateTime);
    }
}
```



- 오리지널 클래스를 Extends 하여
- Double (대역) 처리 하고 싶은 메서드를
- Override 선언하고 구현

Mock

넓은 의미

Mock Object는 **실제 객체의 행동을 의도한 방향으로 흉내 내도록 설계 된 객체**이다.

프로그래머는 일반적으로 다른 어떤 객체의 행동을 테스트하기 위해서 만든다.

어학사전

영어사전 단어·속어 1-5 / 132건

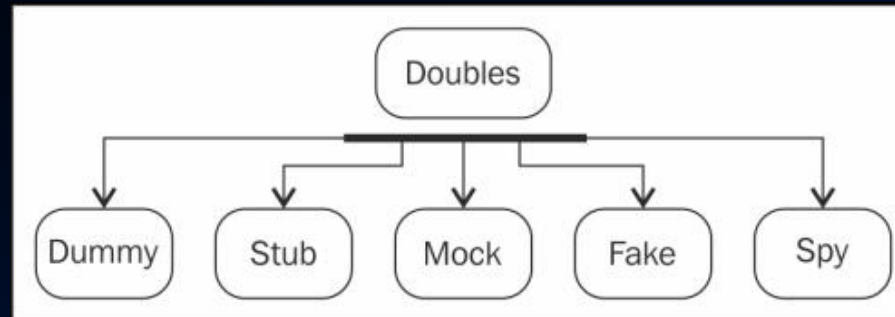
mock 미국식 [ma:k] 영국식 [mɒk] ★

1. (특히 흉내를 내며) 놀리다
2. 무시하다
3. 거짓된, 가짜의

영어 → 한국어

mock māk	모조품 mojopum
--------------------	-----------------------

번역 8개 더보기



헝거게임 Mocking Jay

Testable 클래스 작성 복습 : MailSender에 대한 테스트

오리지널 Mail Sender 업무 클래스가 아래와 같은 상황이라면
Testable 클래스를 사용할 필요가 있지 않을까?

- Mail Sender 가 기능구현이 되지 않은 상태 or 인터페이스만 구현되어 있는 경우
- Mail Sender 클래스가 기능구현이 되어 있지만 테스트시 시간이 너무 많이 걸리는 상태
- Mail Sender 테스트시, 실제 클래스를 호출하면 실제로 Sms를 보내므로 안되는 상태

지난 시간에 우리가 했던 Test Double 작성 방법은?

```
1 package com.sds.cleancode.restaurant;
2
3 public class SmsSender {
4
5     public void send(Schedule schedule) {
6         // send message to PhoneNumber
7     }
8
9 }
10
```

오리지널 클래스를 extends 한
Testable 클래스를
Testcase 에서 사용

오리지널 클래스에 구현 내용이
없었기 때문에
구체적인 내용을 검증하기 힘들

```
1 package com.sds.cleancode.restaurant;
2
3 public class TestableSmsSender extends SmsSender {
4     private boolean sendMethodIsCalled;
5
6     @Override
7     public void send(Schedule schedule) {
8         sendMethodIsCalled = true;
9     }
10
11     public boolean isSendMethodIsCalled() {
12         return sendMethodIsCalled;
13     }
14 }
```

TestableMailSender

대역 클래스 작성 복습

```
3 public class TestableMailSender extends {  
4     /*  
5     * #1 mail 전송한 count 변수 선언  
6     */  
7  
8     /*  
9     * #2 MailSender의 sendMail이 구현한 내용이 없으므로 MailSender를 extends하여, sendMail을 Override 하고  
10    * Override된 sendMail 함수가 호출될 때마다 count 증가  
11    */  
12  
13    /*  
14    * #3 count 리턴 메서드  
15    */  
16 }
```

TestableMailSender

```
1 package com.sds.cleancode.restaurant;
2
3 public class TestableMailSender extends MailSender {
4
5     /*
6      * #1. mail 전송한 countSendMailMethodIsCalled 변수 선언 (int 형)
7      */
8     private int countSendMailMethodIsCalled= 0;
9
10    /*
11     * #2. MailSender의 sendMail이 구현한 내용이 없으므로 MailSender를 extends하여, sendMail을 Override 하고
12     * Override된 sendMail 함수가 호출될 때마다 countSendMailMethodIsCalled 변수 증가
13     */
14    @Override
15    public void sendMail(Schedule schedule) {
16        countSendMailMethodIsCalled++;
17    }
18
19    /*
20     * #3. countSendMailMethodIsCalled 변수를 리턴 하는 getCountSendMailMethodIsCalled 메서드
21     */
22    public int getCountSendMailMethodIsCalled() {
23        return countSendMailMethodIsCalled;
24    }
25 }
```

이런 Test Double 작성 방식은 좀 불편하지 않나?

그러면 어떻게?



Framework를 활용해 보자

Mock Framework 활용 주요전략

Mockito Framework를 활용하여, Testable 클래스 삭제

Mock

Spy

InjectMocks

Setter 삭제



Java.util.List 에 Mockito 적용해 보기 (1/2)

now you can **verify** interactions

```
import static org.mockito.Mockito.*;
```

```
// mock creation
```

```
List mockedList = mock(List.class);
```

```
// using mock object - it does not th
```

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

```
mockedList.clear();
```

```
// selective, explicit, highly readable verification
```

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

```
verify(mockedList).clear();
```

이 예제와 관련해서 아래의 경우를 상상해 봅시다.

- 우리가 하는 Test는 UI 부터 일어나는 테스트가 아니기 때문에, 입력받은 값이 없는데, 테스트를 해야할 소스코드에 서는 UI로 부터 받은 값의 List가 필요한 상태
- 그 List의 대역을 만들어 사용

Java.util.LinkedList 에 Mockito 적용해 보기 (2/2)

and **stub** method calls

```
// you can mock concrete classes, not only interfaces
LinkedList mockedList = mock(LinkedList.class);

// stubbing appears before the actual execution
when(mockedList.get(0)).thenReturn("first");

// the following prints "first"
System.out.println(mockedList.get(0));

// the following prints "null" because get(999) is not stubbed
System.out.println(mockedList.get(999));
```

이 예제와 관련해서 아래의 경우를 상상해 봅시다.

- 내가 테스트해야할 Service가 UI로 부터 List를 전달받는데, Unit TestCase는 정적으로 이루어져서 UI를 통한 Test를 하지 않으므로 UI로 부터 전달받은 List가없는 상태
- 내가 테스트해야할 Service는 UI로 부터 받은 List중 get(0)을 했을때, "first" 라는 값이 나오냐, 나오지 않느냐에 따라 분기문을 처리하는 로직을 가지고 있음
- 그 List의 대역을 만들어 테스트

Test Double을 언제쓰나?

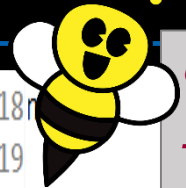
- Library나 Legacy 코드등 과의 **Dependency**를 끊고 싶을때
- 테스트할 클래스가 **기능구현이 되지 않은 상태 (interface 등)**
- 기능구현이 되어 있지만 **테스트시 시간이 너무 많이 걸리는 상태**
- 테스트할때, 실제 DB값을 변경하므로 **실제 클래스를 호출하면 안되는 상태**
- etc..

[실습] Mock을 사용하여 테스트 코드 리팩토링 (1/5)

mock 메서드 활용

• Step1. mock기능을 사용하여 Customer dummy, mock 객체 생성

- CUSTOMER_WITHOUT_EMAIL은 **필드사용이 필요 없으므로** 단순 mock을 통해 null 반환
- CUSTOMER_WITH_EMAIL은 **email 필드를 사용하므로** Mockito.RETURNS MOCKS를 사용하여 null이 아닌 mock반환
- (실제객체가 아닌 .class나 인터페이스를 사용 , new 로 선언하는 구현체는 안됨)



*Customer 클래스의 getEmail() 메서드 결과가 null이 되지 않도록 RETURNS MOCKS 사용
(사용 시 "" 반환)*

```
18  
19  
20 final Date  
21 final DateTime ON_THE_HOUR = DATE_TIME_FORMATTER.parseDateTime("2017/06/19 17:00");  
22 final DateTime NOT_ON_THE_HOUR = DATE_TIME_FORMATTER.parseDateTime("2017/06/19 17:05");  
23 final Customer CUSTOMER_WITHOUT_EMAIL = mock(Customer.class);  
24 final Customer CUSTOMER_WITH_EMAIL = mock(Customer.class, Mockito.RETURNS MOCKS);  
25 final int CAPACITY_PER_HOUR = 3;  
26 final int UNDER_CAPACITY = 1;  
27 BookingScheduler bookingScheduler = new BookingScheduler(CAPACITY_PER_HOUR);  
28 List<Booking> addedSchedules = new ArrayList<>();  
29 TestableMailSender testableMailSender = new TestableMailSender();  
30 TestableSmsSender testableSmsSender = new TestableSmsSender();  
31
```

Dummy
Mock

*Biz 로직이 null, not null이 아닌, 특정 이메일 주소등을 가지고
분기하고 있다면 when, thenReturn 등을 써야했음. (Stubbing)*

*그냥 mock()을 쓰면.. default로 "RETURNS_DEFAULTS"가 지정
이것은... 0, empty collections, null 을 리턴
객체면 null을 리턴.. int 형 등이면.. 0,*

```
17  
18  
20 final DateTime ON_THE_HOUR = DATE_TIME_FORMATTER.parseDateTime("2017/06/19 17:00");  
21 final DateTime NOT_ON_THE_HOUR = DATE_TIME_FORMATTER.parseDateTime("2017/06/19 17:05");  
22 final Customer CUSTOMER_WITHOUT_EMAIL = new Customer("Fake name", "010-1234-5678");  
23 final Customer CUSTOMER_WITH_EMAIL = new Customer("Fake name", "010-1234-5678", "010-1234-5678");  
24 final int CAPACITY_PER_HOUR = 3;  
25 final int UNDER_CAPACITY = 1;  
BookingScheduler bookingScheduler = new BookingScheduler(CAPACITY_PER_HOUR);  
List<Booking> addedSchedules = new ArrayList<>();  
TestableMailSender testableMailSender = new TestableMailSender();  
TestableSmsSender testableSmsSender = new TestableSmsSender();  
30  
31
```

[실습] Mock을 사용하여 테스트 코드 리팩토링 (2/5)

InjectMocks, Spy 활용

• Step2. setSchedules를 없애기 위한 InjectMocks

- @Spy를 활용하여 bookingScheduler, addedSchedules를 spy
- @InjectMocks를 사용하여 Spy객체 Inject
- @RunWith(MockitoJUnitRunner.class) 추가하여 실행
(RunWith... 선언해줘야 InjectMock, Spy등 어노테이션이 활성화됨)



```
private static final int CAPACITY_PER_HOUR = 3;
private static final int UNDER_CAPACITY = 1;

@InjectMocks
@Spy
private BookingScheduler bookingScheduler = new BookingScheduler(CAPACITY_PER_HOUR);
@Spy
private List<Schedule> addedSchedules = new ArrayList<>();
private TestableMailSender testableMailSender = new TestableMailSender();
private TestableSmsSender testableSmsSender = new TestableSmsSender();

@Before
public void setUp(){
    bookingScheduler.setSmsSender(testableSmsSender);
    bookingScheduler.setMailSender(testableMailSender);
}
```

실제객체만 가능

setUp 메서드에서 setSchedules 삭제?

→ mock을 활용하면 setMethod가 없어도 Inject 가능

→ InjectMocks 선언하면 이하 @Spy 어노테이션 선언한 부분들의 객체가 InjectMock의 객체로 Injection 됨

→ 그래서 setMethod 안해줘도 됨

```
26 private static final int UNDER_CAPACITY = 1;
27 private BookingScheduler bookingScheduler = new BookingScheduler(CAPACITY_P
28 private List<Schedule> addedSchedules = new ArrayList<>();
29 private TestableMailSender testableMailSender = new TestableMailSender();
30 private TestableSmsSender testableSmsSender = new TestableSmsSender();
31
32 @Before
33 public void setUp(){
34     bookingScheduler.setSchedules(addedSchedules);
35     bookingScheduler.setSmsSender(testableSmsSender);
36     bookingScheduler.setMailSender(testableMailSender);
37 }
38
39 @Test(expected = RuntimeException.class)
```

[실습] Mock을 사용하여 테스트 코드 리팩토링 (3/5)

Spy 하루

• Step3. setSmsSender를 없애기 위한 InjectMocks

- @Spy를 활용하여 SmsSender를 spy
- 필요없는 TestableSmsSender 클래스 삭제
- verify를 이용하여 메소드 호출 테스트



setup 메서드에서 setSmsSender 삭제?

→ mock을 활용하면 setMethod가 없어도 Inject 가능

→ InjectMocks 선언하면 이하 @Spy 어노테이션 선언한 부분들의 객체가 InjectMock의 객체로 Injection 됨

→ 그래서 setMethod 안해줘도 됨

```
37 @Spy
38 private SmsSender smsSender = new SmsSender();
39
40 @Before
41 public void setUp(){
42     bookingScheduler.setMailSender(testableMailSender);
43 }
```

```
101 @Test
102 public void sendSmsWhenScheduleIsAdded() throws Exception {
103     //arrange
104     Schedule schedule = new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT);
105
106     //act
107     bookingScheduler.addSchedule(schedule);
108
109     //assert
110     verify(smsSender, times(1)).send(any(Schedule.class));
111 }
112
```



```
38 private TestableSmsSender testableSmsSender = new TestableSmsSender();
39
40 @Before
41 public void setUp(){
42     bookingScheduler.setSmsSender(testableSmsSender);
43     bookingScheduler.setMailSender(testableMailSender);
44 }
```

Testable 클래스 대신 Spy를 썼을때 다른점 중의 하나는

Testable 클래스에서는 단순히 send메서드가 호출되었는지 알기 위해 사용되었는데, mockito의 Spy 사용하면 verify 기능을 통해서 호출 횟수등을 검증할 수 있음.

```
110 //assert
111 assertThat(testableSmsSender.isSendMethodCalled(), is(true));
112 }
113
```

[실습] Mock을 사용하여 테스트 코드 리팩토링 (4/5)

Spy 활용

• Step4. setMailSender를 없애기 위한 InjectMocks

- spy 메서드를 활용하여 MailSender를 spy
- 필요없는 TestableMailSender 클래스 삭제
- verify를 이용하여 메소드 호출 테스트

```
34 @Spy
35 private List<Schedule> addedSchedules = new ArrayList<>();
36
37 private MailSender mailSender = spy(new MailSender());
38
39 @Spy
40 private SmsSender smsSender = new SmsSender();
41
42 @Before
43 public void setUp(){
44
45 }
```

실제객체만 가능

- *setup 메서드에서 setMailSender 삭제?*
→ *mock을 활용하면 setMethod가 없어도 Inject 가능하다*
- *호출 확인을 위해 Testable 클래스를 사용하지 않아도 된다.*

```
//assert
verify(mailSender, never()).sendMail(any(Schedule.class));
```

```
//assert
verify(mailSender, times(1)).sendMail(any(Schedule.class));
```

```
33 p
34 @Spy
35 private List<Schedule> addedSchedules = new ArrayList<>();
36 private TestableMailSender testableMailSender = new TestableMailSender();
37
38 @Spy
39 private SmsSender smsSender = new SmsSender();
40
41 @Before
42 public void setUp(){
43     bookingScheduler.setMailSender(testableMailSender);
44 }
```

verify를 사용하여 메서드 호출여부 및 호출횟수 확인 (times(1), never())

```
119 //assert
120
121 assertThat(testableMailSender.getCountSendMailMethodIsCalled(), is(0))
122
123
```

```
132 //assert
133 assertThat(testableMailSender.getCountSendMailMethodIsCalled(), is(1))
134
135
```


[실습] Mock을 사용하여 테스트 코드 리팩토링 (5/5)

Stubbing 해보기

- Step5. TestableBookingSchedule을 없애기 위해 Stub사용

- Method Stub을 사용하여 Testable클래스 삭제
- when.thenReturn을 사용하여 getNow 리턴값을 Sunday로 반환
- doReturn.when을 사용하여 getNow 리턴값을 Monday로 반환



- Stub을 사용해서 메소드를 오버라이드 하는
- Testable 클래스를 대체해볼 수 있다.

```
140 public void throwAnExceptionOnSunday() throws Exception {
141     //arrange
142     DateTime sunday = DATE_TIME_FORMATTER.parseDateTime("2017/06/18 17:00");
143     when(bookingScheduler.getNow()).thenReturn(sunday);
144     try {
145         // act
146         Schedule newSchedule = new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WIT
147         bookingScheduler.addSchedule(newSchedule);
148         fail();
149     } catch (RuntimeException e) {
150         // assert
151         assertThat(e.getMessage(), is("Booking system is not available on sunday"));
152     }
153 }
154
155 @Test
156 public void scheduleShouldBeAddedOnAnyDaysExceptSunday() throws Exception {
157     //arrange
158     DateTime monday = DATE_TIME_FORMATTER.parseDateTime("2017/06/19 17:00");
159     doReturn(monday).when(bookingScheduler).getNow();
160
161     //act
162     Schedule newSchedule = new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITHOUT
163     bookingScheduler.addSchedule(newSchedule);
164 }
```

```
140 public void throw
141     //arrange
142     DateTime sunday = DATE_TIME_FORMATTER.parseDateTime("2017/06/18 17:00");
143     bookingScheduler = new TestableBookingScheduler(CAPACITY_PER_HOUR, sunday);
144     try {
145         // act
146         Schedule newSchedule = new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_
147         bookingScheduler.addSchedule(newSchedule);
148         fail();
149     } catch (RuntimeException e) {
150         // assert
151         assertThat(e.getMessage(), is("Booking system is not available on sunday"));
152     }
153 }
154
155 @Test
156 public void scheduleShouldBeAddedOnAnyDaysExceptSunday() throws Exception {
157     //arrange
158     DateTime monday = DATE_TIME_FORMATTER.parseDateTime("2017/06/19 17:00");
159     bookingScheduler = new TestableBookingScheduler(CAPACITY_PER_HOUR, monday);
160
161     //act
162     Schedule newSchedule = new Schedule(ON_THE_HOUR, UNDER_CAPACITY, CUSTOMER_WITH
163     bookingScheduler.addSchedule(newSchedule);
164 }
```



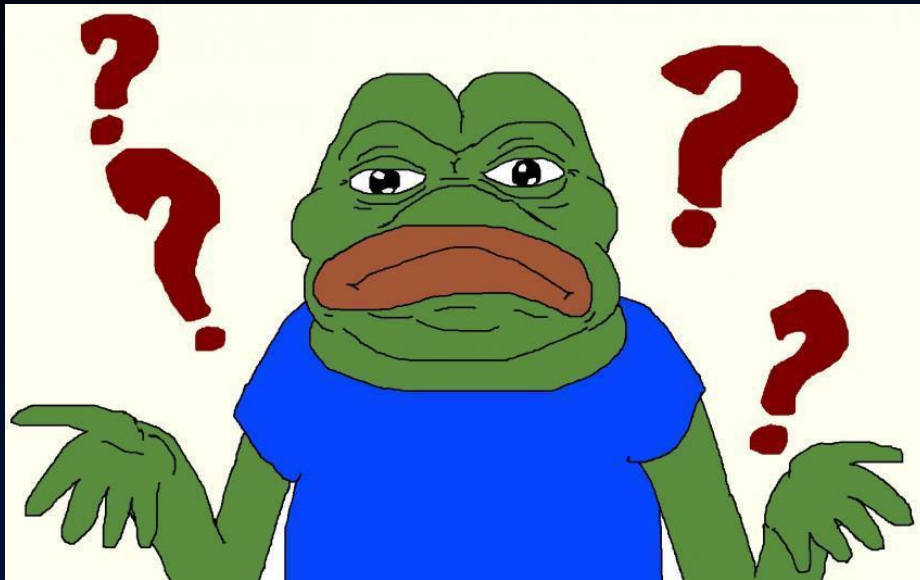

패턴을 활용한 리팩토링 (Mob Programming)

디자인 패턴이란

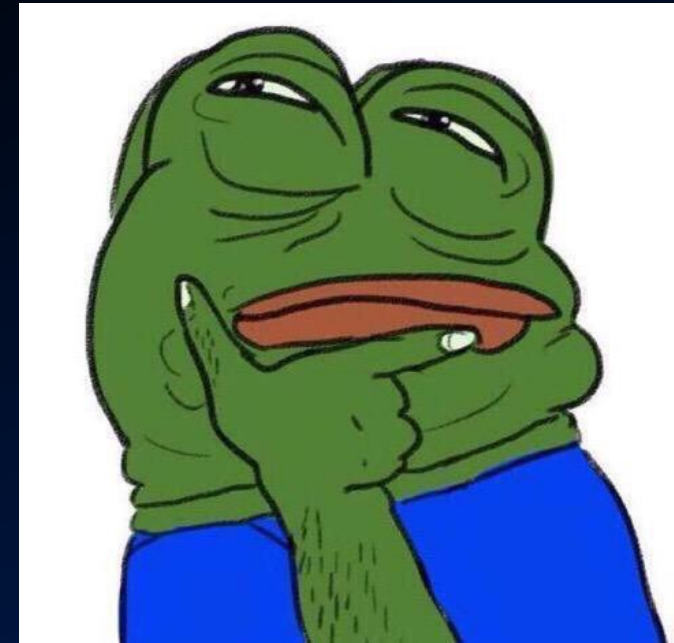
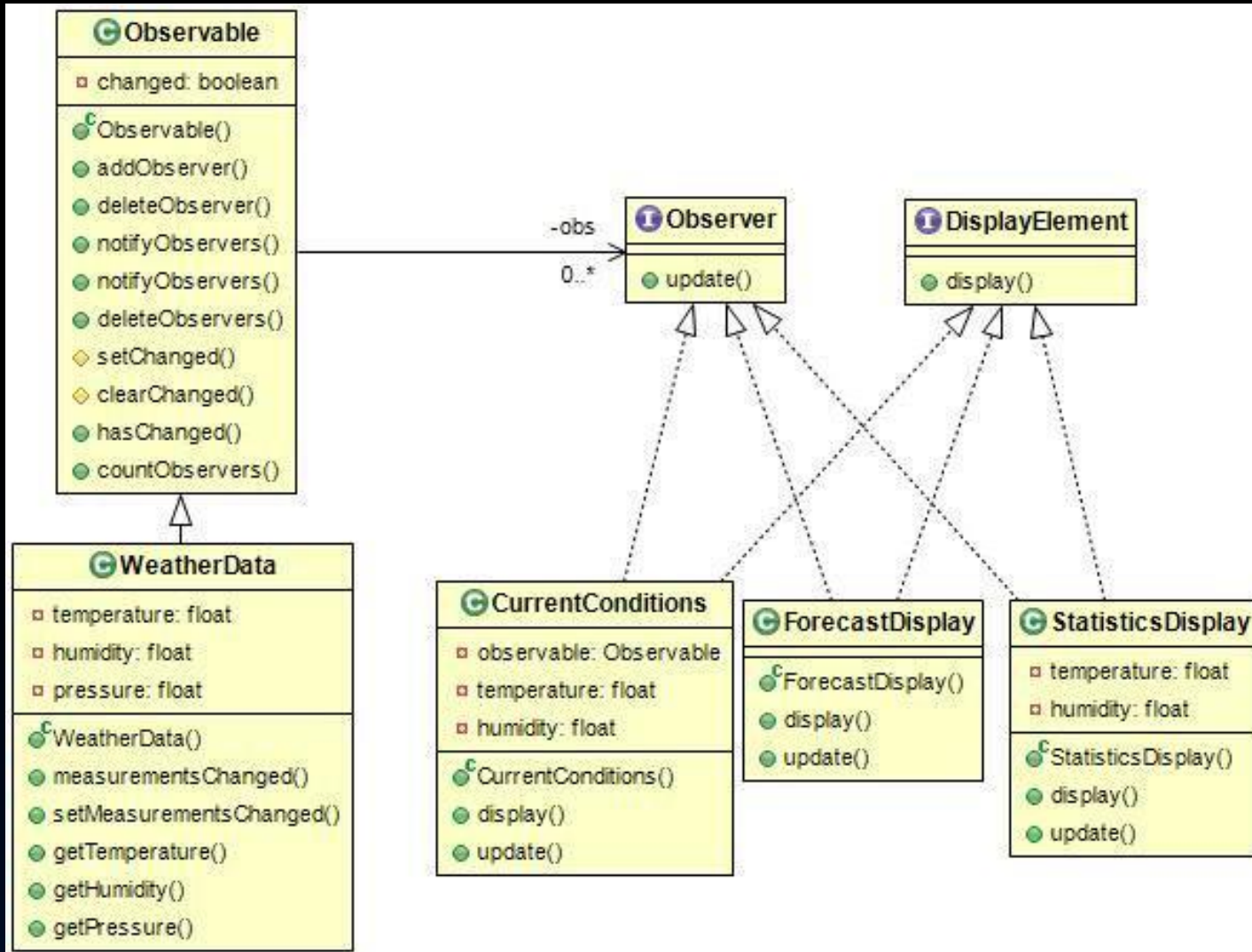
- 설계 문제에 대한 해답을 문서화 하기 위해 고안된 형식 방법(넓은 의미)
- 프로그램 개발에서 자주 나타나는 과제를 해결하기 위한 방법 중 하나
- 과거의 소프트웨어 개발 과정에서 발견된 **설계의 노하우**를 축적하여 이름을 붙여, 이후에 **재이용**하기 좋은 형태로 **특정의 규약**을 묶어서 정리한 것
- 알고리즘과 같이 프로그램 코드로 바로 변환될 수 있는 형태는 아니지만, 특정한 상황에서 구조적인 문제를 해결하는 방식을 설명해 준다.

Observer 패턴

- 객체의 **상태 변화를 관찰**하는 관찰자들(**Observer**)
- 옵저버들의 목록을 객체에 등록하여 상태 변화가 있을 때마다 메서드 등을 통해 **객체가 직접** 목록의 각 **옵저버에게 통지(Notify)**
- '발행(Publication)-구독(Subscription)모델'으로도 불린다.



Observer 패턴의 예



Observer 패턴 구현을 말로 풀어보면...

①

*Observer를 implements 하면
상태변화를 관찰할 수 있다.*

②

*Observer들이 관찰가능한 클래스는
Observable을 extends 한
클래스들이다.*

③ *Observer가 Observable extends한
클래스들을 관찰은 하지만
Observable에서 setChanged(),
notifyObservers() 를 실행시켜줘야
Observer가 반응한다.*

④

*이때 Observer는 Observable에서
notifyObservers() 메서드를 통해
전달되는 Parameter를 Observer에서
활용할 수 있다.*

⑤

*Observer는 이 Parameter를 update
메서드를 구현함으로써
전달 받을 수 있다.*

⑥

*Observer의 update 메서드에
구현하고자 하는 기능을 작성한다.*

⑦

*하지만, Glue Code에서
addObservers를 해줘야
Observable에서 notify될때
Observable이 호출된다.*



*Observer : SmsSender, MailSender
Observable : BookingScheduler
Glue Code : Test Case 클래스*



[실습] Mob Programming

아바타와 조종사 선발



조종사



아바타

<https://www.youtube.com/watch?v=QceSt62NUfk>
(20초 ~ 3분 10초)

[실습] Observer 패턴 실습 (1/8)

- Step1. 옵저버들은 `java.util.Observer`를 implements

```
1 package com.sds.cleancode.restaurant;
2
3 import java.util.Observer;
4
5 public class SmsSender implements Observer {
6
7     public void send(Schedule schedule) {
8         // send message to PhoneNumber
9     }
10
11 }
12
```

```
1 package com.sds.cleancode.restaurant;
2
3 import java.util.Observer;
4
5 public class MailSender implements Observer {
6
7     public void sendMail(Schedule schedule) {
8         // send mail
9     }
10 }
```


[실습] Observer 패턴 실습 (2/8)

- Step2. 옵저버블 클래스는 java.util.Observable을 extends

```
7 import org.joda.time.DateTimeConstants;
8
9 public class BookingScheduler extends Observable {
10     private int capacityPerHour;
11     private List<Schedule> schedules;
12     private SmsSender smsSender;
13     private MailSender mailSender;
14
15     public BookingScheduler(int capacityPerHour) {
16         this.capacityPerHour = capacityPerHour;
17     }
18
19     public void addSchedule(Schedule schedule) {
20
```

[실습] Observer 패턴 실습 (3/8)

- Step3. 옵저버블 클래스에서 `setChanged()`, `notifyObservers()` 를 실행

```
BookingScheduler.java BookingSchedulerTest.java SmsSender.java MailSender.java
30
37 // throw an exception on sunday
38 DateTime now = getNow();
39 if(now.getDayOfWeek() == DateTimeConstants.SUNDAY){
40     throw new RuntimeException("Booking system is not available on sunday");
41 }
42
43 schedules.add(schedule);
44
45 setChanged();
46 notifyObservers();
47
48
49 // send SMS to customer
50 smsSender.send(schedule);
```

[실습] Observer 패턴 실습 (4/8)

- Step4. Observer는 Observable에서 notifyObservers() 메서드를 통해 전달되는 Parameter를 Observer에서 활용할 수 있다.

```
BookingScheduler.java BookingSchedulerTest.java SmsSender.java MailSender.java
37 // throw an exception on sunday
38 DateTime now = getNow();
39 if(now.getDayOfWeek() == DateTimeConstants.SUNDAY){
40     throw new RuntimeException("Booking system is not available on sunday");
41 }
42
43 schedules.add(schedule);
44
45 setChanged();
46 notifyObservers(schedule);
47
48
49 // send SMS to customer
50 smsSender.send(schedule);
```

[실습] Observer 패턴 실습 (5/8)

- Step5. Observer는 update 메서드를 구현함으로써 Observable로부터 Parameter를 전달 받을 수 있다.

```
6 public class SmsSender implements Observer {
7
8     public void send(Schedule schedule) {
9         // send message to PhoneNumber
10    }
11
12    @Override
13    public void update(Observable o, Object arg) {
14    }
15 }
```

```
6 public class MailSender implements Observer {
7
8     public void sendMail(Schedule schedule) {
9         // send mail
10    }
11
12    @Override
13    public void update(Observable o, Object arg) {
14    }
15 }
```

[실습] Observer 패턴 실습 (6/8)

- Step6. Observer의 update 메서드에 구현하고자 하는 기능을 작성한다.

```
3 import java.util.Observable;
4 import java.util.Observer;
5
6 public class SmsSender implements Observer {
7
8     public void send(Schedule schedule) {
9         // send message to PhoneNumber
10    }
11
12    @Override
13    public void update(Observable o, Object arg) {
14        send((Schedule)arg);
15    }
16 }
```

```
6 public class MailSender implements Observer {
7
8     public void sendMail(Schedule schedule) {
9         // send mail
10    }
11
12    @Override
13    public void update(Observable o, Object arg) {
14        Schedule schedule= (Schedule)arg;
15        if(schedule.getCustomer().getEmail() != null){
16            sendMail(schedule);
17        }
18    }
19 }
```

[실습] Observer 패턴 실습 (7/8)

- Step7. Glue Code에서 addObserver를 해줘야 Observable에서 notify될때 Observable이 호출된다.

```
BookingScheduler.java  BookingSchedulerTest.java  SmsSender.java  MailSender.java
45
46 @Spy
47 private MailSender mailSender= new MailSender();
48
49 @Spy
50 private SmsSender smsSender= new SmsSender();
51
52 @Before
53 public void setUp() {
54     booking.addObserver(smsSender);
55     booking.addObserver(mailSender);
56 }
57
```

[실습] Observer 패턴 실습 (8/8)

- Step8. 패턴 적용전 로직을 삭제한다.



작업이 완료되면
Alt + Shift + E 누른상태에서
잠깐쉬고
T 를 눌러
JUnit 과 Test Coverage를 확인해 보세요

BookingScheduler.java BookingSchedulerTest.java SmsSender.java M

```
46 notifyObservers(schedule);
```

```
47
```

```
48
```

```
49 // send SMS to customer
```

```
50 smsSender.send(schedule); 삭제
```

```
51
```

```
52 // send E-mail to customer when e-mail is valid
```

```
53 if(schedule.getCustomer().getEmail() != null){
```

```
54     mailSender.sendMail(schedule);
```

```
55 }
```

```
56 }
```

```
57
```



회고



회고

(Plus, Minus, Insight)



마치며

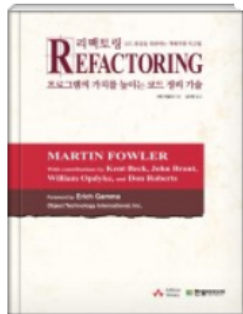


• References

- "테스트 자동화 및 서비스 가상화를 통한 끊임 없는 테스트 환경 구축 전략", http://www-903.ibm.com/edm/J1309/0904_wyk/2.pdf
- "Observer 패턴의 예", <http://jusungpark.tistory.com/8>
- "Mockito 프레임워크 사용예", <https://site.mockito.org/>
- "Glue Code", https://en.wikipedia.org/wiki/Glue_code
- "Mock", https://en.wikipedia.org/wiki/Mock_object
- "테스트 케이스의 효과", <https://www.slideshare.net/dhrim/test-case>
- "Mockito 사용법", <https://jdm.kr/blog/222>
- "doReturn, thenReturn 차이설명", <https://code.i-harness.com/ko-kr/q/1369336>
- <https://m.blog.naver.com/PostView.nhn?blogId=inho1213&logNo=80110527396&proxyReferer=https%3A%2F%2Fwww.google.com%2F>
- "Mock", https://en.wikipedia.org/wiki/Mock_object
- "Mock 연습해보기 좋은 사이트", <http://static.javadoc.io/org.mockito/mockito-core/2.23.0/org/mockito/Mockito.html>

• 책소개



리팩토링 코드 품질을 개선하는 객체지향 사고법



★★★★★ 9.4 | 네티즌리뷰 18건

저자 마틴 파울러 | 역자 김지원 | 한빛미디어 | 2012.11.09

원제 Refactoring : improving the design of existing code

페이지 500 | ISBN  9788979149715 | 판형 B5, 188*257mm | 더보기 

도서

27,000원 ~~30,000원~~ -10%

가격정보

♡ 38



패턴을 활용한 리팩터링



★★★★★ 9.33 | 네티즌리뷰 3건

저자 조슈아 케리예브스키 | 역자 윤성준, 조상민 | 인사이트 | 2006.07.25

원제 Refactoring to patterns

페이지 476 | ISBN  9788991268203 | 판형 규격외 변형

도서

22,500원 ~~25,000원~~ -10%

가격정보

♡ 1

