

A large white triangle containing the text "Unit TEST" in a bold, sans-serif font. Below "TEST" is a horizontal line, followed by the word "BOOSTCAMP" in a smaller, all-caps, sans-serif font. The background is dark grey with faint, overlapping words like "time", "team", "bug", "web", "dev", and "practices".

Team:Missile

장철운

Step.1



Test의 종류.



Test의 종류에
대해 간략히 설명한다.

Step.2



*Test를
왜
하는가?*



Test를 작성하는 이유와
작성했을 때의 이점을
알 수 있다.

Step.3



*어떻게
작성하는가?*



개발하면서 작성했던
Test들을 예시로
Bad - Good Case를
볼 수 있다.

Step.4



*설계관점에서의
Test*



Test를 설계측면에서
작성하는 방법론
'TDD'에 대하여
설명한다.

Why?

검증

검증을 위한
테스트

—
의도한 대로 작동하는지
검증

신뢰성 및 안전성 향상

검증

검증을 위한
테스트

의도한 대로 작동하는지
검증

신뢰성 및 안전성 향상

설계

설계를 위한
테스트

설계의 도구로
Test를 사용

TDD를 이용한 개발 방법론

| Why

Controller 및 JSP 작성



Server 가동



Error !!

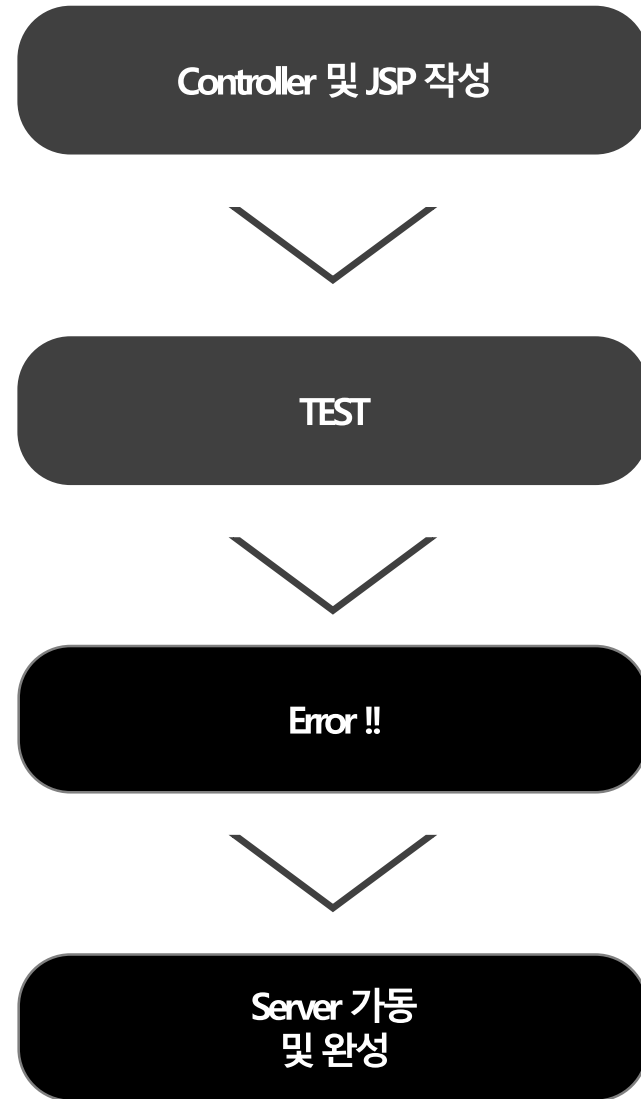
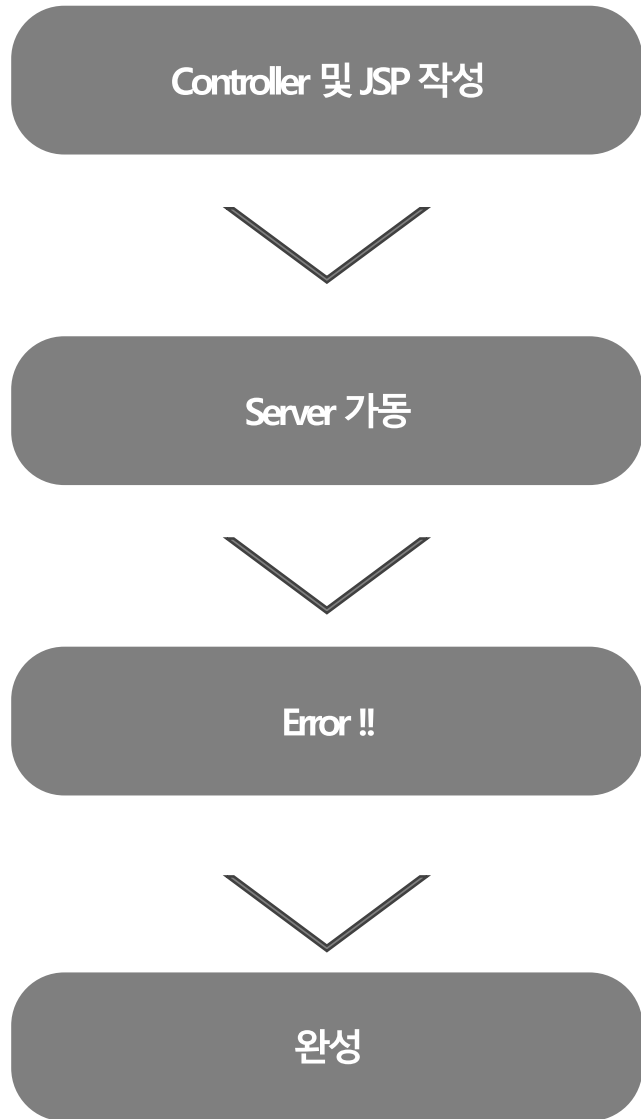


완성



디버깅

| Why



인수 테스트 (Acceptance Test)

사용자 요구사항 처리에 대한 검증으로 사용자가 요구기능을 입력하고 기능이 정확하게 수행하는지 확인

시스템 테스트 (System Test)

전체 시스템 동작에 대한 검증.

통합 테스트 (Integration Test)

컴포넌트간의 상호 작용에 대한 검증으로 테스트 입력 값을 만들어 실행한 후 결과를 확인

단위 테스트 (Unit Test)

분리된 기능에 대한 검증으로 단위 테스트 프레임워크를 이용하여 개발자가 테스트

인수 테스트 (Acceptance Test)

사용자 요구사항 처리에 대한 검증으로 사용자가
요구기능을 입력하고 기능이 정확하게 수행하는지 확인

시스템 테스트 (System Test)

전체 시스템 동작에 대한 검증.

통합 테스트 (Integration Test)

컴포넌트간의 상호 작용에 대한 검증으로
테스트 입력 값을 만들어 실행한 후 결과를 확인

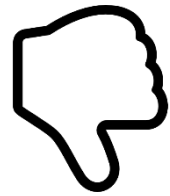
단위 테스트 (Unit Test)

분리된 기능에 대한 검증으로
단위 테스트 프레임워크를 이용하여 개발자가 테스트



My BAD Case

```
@Test
public void shouldSelectCount() {
    Long count = service.selectCount();
    log.info("shouldSelectCount: " + count.toString());
    Assert.assertNotNull(count);
}
```



Happy Test

```
@Test
public void shouldSelectCount() {
    Long count = service.selectCount();
    log.info("shouldSelectCount: " + count.toString());
    Assert.assertNotNull(count);
}
```

이 테스트는 메소드를 정확히 테스트 하고 있는가?



Happy Test

```
@Test
public void shouldSelectCount() {
    Long count = service.selectCount();
    log.info("shouldSelectCount: " + count.toString());
    Assert.assertNotNull(count);
}
```

```
@Override
public Long selectCount() {
    Long count = cachedCount.get((long) 0);
    if (count == null) {
        count = dao.selectCount();
        cachedCount.put((long) 0, count); return count;
    } return count;
}
```



Happy Test

```
@Test
public void shouldSelectCount() {
    ...
    Assert.assertNotNull(count);
}
```



```
@Test
public void shouldSelectCachedCount() {
    ...

    verify(dao, times(0)).selectCount();
    verify(cachedCount, times(1)).get(TOTAL_CATEGORY_ID);
    Assert.assertThat(count, is(RESULT));
}
```



```
@Test
public void shouldSelectNotCachedCount() {
    ...

    verify(dao, times(1)).selectCount();
    verify(cachedCount, times(1)).get(TOTAL_CATEGORY_ID);
    verify(cachedCount, times(1)).put(TOTAL_CATEGORY_ID, count);
    Assert.assertThat(count, is(RESULT));
}
```



Layer

```
@Test
public void shouldSelectCount() {
    Long count = service.selectCount();
    log.info("shouldSelectCount: " + count.toString());
    Assert.assertNotNull(count);
}
```

이 테스트는 정확히 ProductService의
selectCount() 메소드를 테스트 하는가?





테스트 더블 (Test Double)



테스트 목적에 따라 실제 코드의 기능을 대체하는 비슷하면서도 다른 객체.

종류

- 테스트 스텝
- 가짜 객체
- 테스트 스파이

테스트 더블이 필요한 이유

- 테스트 대상 코드를 격리한다.
- 테스트 속도를 개선한다.
- 예측 불가능한 실행 요소를 제거한다.
- 특수한 상황을 시뮬레이션한다.
- 감춰진 정보를 얻어낸다.

```

public class ProductDaoMock {
    public int selectCount() {
        return 10;
    }
}
  
```

Mock 객체

라이브러리에서 제공하는 다양한 메소드를 가진 테스트 더블 객체.

verify()

```
verify(dao, times(0)).selectCount();
```

When() thenReturn()

```
when(cachedCount.get(TOTAL_CATEGORY_ID)).thenReturn(RESULT);
```

```
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
```

```
public class ProductServiceTest {
```

```
@Mock
```

```
ProductDao dao;
```

```
@Mock
```

```
Map<Long, Long> cachedCount;
```

```
@Before
```

```
public void setUp() {
```

```
    MockitoAnnotations.initMocks(this);
```

```
    service = new ProductServiceImpl();
```

```
    service.setDao(dao);
```

```
    service.setCachedCount(cachedCount);
```

```
}
```



Layer

```
@Test
public void shouldSelectCachedCount() {
    long count;
    long TOTAL_CATEGORY_ID = 0;
    long RESULT = 10;
    when(cachedCount.get(TOTAL_CATEGORY_ID))
        .thenReturn(RESULT);

    count = service.selectCount();

    verify(dao, times(0)).selectCount();
    verify(cachedCount, times(1)).get(TOTAL_CATEGORY_ID);
    Assert.assertThat(count, is(RESULT));
}
```

```
@Test
public void shouldSelectNotCachedCount() {
    long count;
    long TOTAL_CATEGORY_ID = 0;
    long RESULT = 10;
    when(cachedCount.get(TOTAL_CATEGORY_ID)).thenReturn(null);
    when(dao.selectCount()).thenReturn(RESULT);

    count = service.selectCount();

    verify(dao, times(1)).selectCount();
    verify(cachedCount, times(1)).get(TOTAL_CATEGORY_ID);
    verify(cachedCount, times(1)).put(TOTAL_CATEGORY_ID, count);
    Assert.assertThat(count, is(RESULT));
}
```

Service 계층에 집중 할 수 있다.

DAO Test



DB Unit 이용

@Before

```
public void setUp() throws Exception {  
    MockitoAnnotations.initMocks(this);  
  
    DataSource dataSource = TestDBConnect.getDataSource();  
    IDatabaseConnection connection = new  
    DatabaseDataSourceConnection(dataSource);  
    connection.getConfig().setProperty(DatabaseConfig.FEATURE_ALL  
    OW_EMPTY_FIELDS, true);  
    dataSet = new XmlDataSet(  
        new FileInputStream("src/test/resource/testData/  
        commentDaoTestData.xml"));  
  
    this.dao = new CommentDaoImpl(dataSource);  
    DatabaseOperation.CLEAN_INSERT.execute(connection,  
    dataSet);  
}
```

@Test

```
public void shouldSelectList() throws DataSetException {  
    List<ReservationUserComment> list = dao.selectList(1, 0, 3);  
  
    String dataComment = list.get(1).getComment();  
    String xmlDataComment = (String)  
    dataSet.getTable("reservation_user_comment").getValue(1,  
    "comment");  
  
    assertThat(dataComment, is(xmlDataComment));  
}
```

| Controller Test





Spring-Test


















```
@Test
public void shouldSelectList_Context() throws Exception {
    int start = 1, amount = 3;
    long productId = 1;
    List<Product> list = new ArrayList<Product>();
    list.add(new Product().setId(productId));
    when(service.selectList(start, amount)).thenReturn(list);

    mockMvc.perform(get("/api/product?start=1&amount=3")).andExpect(status().isOk())
        .andExpect(content().contentType("application/json;charset=UTF-8"))
        .andExpect(jsonPath("$", hasSize(1)))
        .andExpect(jsonPath("$[0].id").value((int) productId));
    verify(service, times(1)).selectList(start, amount);
}
```

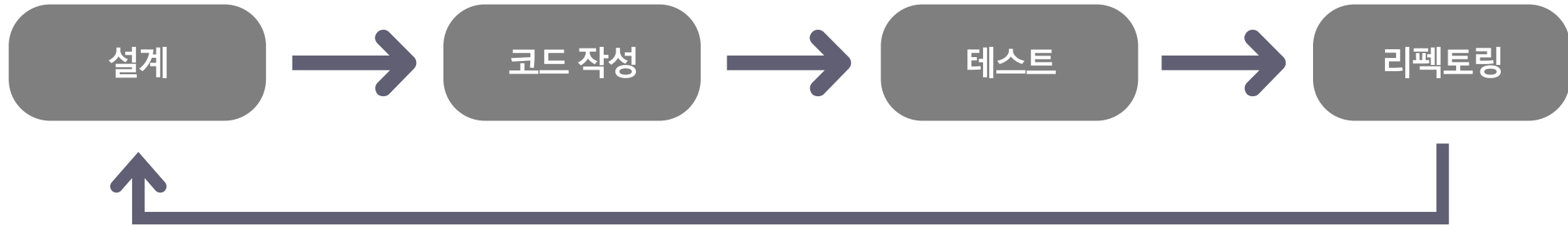
검증완료

 reservation-service >  kr.or.reservation.service.impl

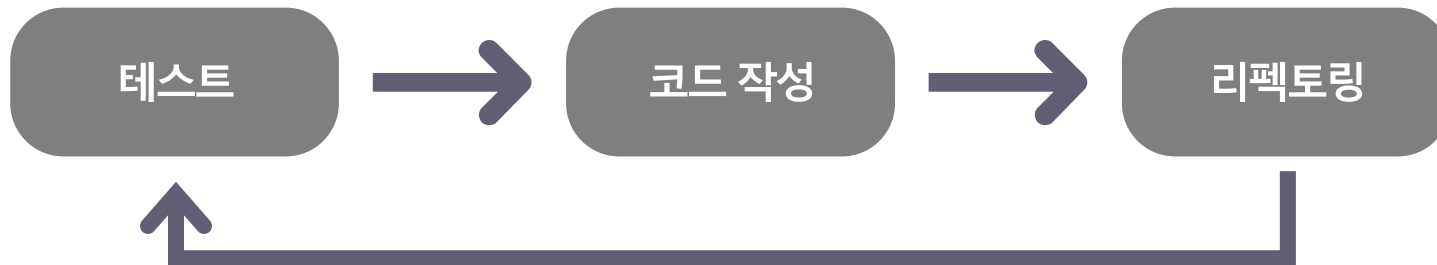
kr.or.reservation.service.impl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 ProductServiceImpl		94%		60%	8	19	5	38	0	9	0	1
 CategoryServiceImpl		87%		50%	2	7	2	10	0	5	0	1
 CommentServiceImpl		100%		75%	3	12	0	14	0	6	0	1
 UserServiceImpl		100%		100%	0	7	0	9	0	6	0	1
 ReservationServiceImpl		100%		100%	0	5	0	9	0	4	0	1
 ImageServiceImpl		100%		n/a	0	3	0	5	0	3	0	1
Total	14 of 329	95%	13 of 40	67%	13	53	7	85	0	33	0	6

| Test-driven development



기존 방법론



Test-driven development

| Test-driven development



설계에 대해
깊게 고민할 수 있다.



품질이 검증된
코드를
갖게 된다.



잘 작성된 테스트는
API문서의 기능을
대신할 수 있다.

Q/A
