

소프트웨어 운영 및 테스트 실무

박민재

mjpark@daelim.ac.kr

수업 개요

- 산학협력강의
 - 융복합학습 구성
- 8월 26일 (오후) 첫 수업
 - 8월 28일 수업 보강
- 반학기제 수업
 - 각 수업은 1주일 단위
 - 정규/보강으로 구성
 - 추석/휴일 등으로 인한 보강계획 협의중
 - 오전 10시~, 오후 2시~
- 각 수업 공지
 - 학과 공지사항 및 단톡방 참고(주의)

강의 주차	월/일	성 명
1	8/28	박민제
2	8/28	박민제
3	9/4	박민제
4	9/4	박민제
5	9/11	오상일 (엑셀테크)
6	9/11	오상일 (엑셀테크)
7	9/18	박민제
8	9/18	박민제
9	9/25	원근주 (인재아이엔씨)
10	9/25	원근주 (인재아이엔씨)
11	10/2	김동환 (오토소프트)
12	10/2	김동환 (오토소프트)
13	10/9 10/16	박민제
14	10/9 10/16	박민제
15	10/23 (기말고사)	박민제

강의 주차	월/일	성 명
1	8/30	최종원 (아이엠테크놀로지)
2	8/30	최종원 (아이엠테크놀로지)
3	9/6	장재호 (토마토시스템)
4	9/6	장재호 (토마토시스템)
5	9/13 (보강일 필요)	박민제
6	9/13 (보강일 필요)	박민제
7	9/20	박민제
8	9/20	박민제
9	9/27	박민제
10	9/27	박민제
11	10/4 10/11	박민제
12	10/4 10/11	박민제
13	10/18	박민제
14	10/18	박민제
15	10/25	박민제

- 각 산업체 전문가 기반 학습
 - 각 교육 내용에 대해 이해
 - 회사에서 진행중인 업무 SW 산업 등에 대한 이해
 - 이해된 내용을 기반으로
- 소프트웨어 운영 및 테스트 실무
 - 소프트웨어 프로세스에 대해 이해 (소프트웨어 인증 등)
 - 소프트웨어 품질 관리 활동에 대한 이해
- 소프트웨어 개발 실무
 - Problem Based Learning
 - 문제 중심 학습
 - 현재 진행중인 작품활동 마무리
 - 작품 기반 가이드 문서 제출/작성
 - 참여형 티칭 가이드북 작성
 - 본 프로젝트(작품활동)을 위해, 활용할 수 있는 문서 작성
 - 개발 가이드/설치 가이드/운영 가이드/트러블 슈팅 가이드 등 필요한 문서를 선택하여 작성
 - 작성 문서를 서로 평가 (10월 10일까지 제출)

SW 테스트 (소프트웨어 테스트)

- 소프트웨어 내에서 버그를 찾는 것(틀린 표현은 아니지만 큰 범주로 본다면 많은 결함을 찾는 것은 소프트웨어의 품질을 높이는 데에 부분적인 영향을 미치는 것에 불과함.)
- Application 이나 System의 정상적인 동작이나 성능이 요구사항에 맞게 올바르게 작동하는지 확인하는 활동.
- 단순히 결함을 찾는 것만이 아닌 기술적인 기능 및 성능을 향상시키고 사용자의 만족도에 맞게 개발되었는지를 확인하고 신뢰도를 높이는 역할.
→ 다시 말해, 테스트를 단순히 제품에 대한 동작을 통한 기능 확인에만 초점을 맞추어 결함을 많이 찾는 것이라고 생각하지 말고,
- **최초의 제품 기획의도와 부합하는지 여부와 실제로 해당 제품의 품질 상태를 기준으로 출시 여부를 결정할 수 있는 다양한 지표를 제공하는 것.**

SW 테스트가 필요한 이유

- IT가 발전함에 따라 높은 기능 수준의 SW가 요구 → 그에 따라 SW품질에 대한 중요성이 현안으로 대두됨.

- “소프트웨어의 결함이 존재함을 보이는 과정”
- 소프트웨어가 문제가 없음을 보이는 것이 아니라 **문제가 있음을 밝히는 과정**이다.
- SWLC(소프트웨어 생명주기)의 프로세스 [요구사항 분석 - 설계 - 구현 - 테스트 - 유지보수] 에서 거의 마지막 단계이다.
- 소프트웨어 테스터들은 “이 소프트웨어가 완벽하군요!” 라고 하는 것이 아니라,
- “이 소프트웨어는 결함이 없군요!” 라고 말해야 한다.

SW 테스트 유형

- 테스트 목적에 따른 테스트 종류

종류	내용
기능 테스트	소프트웨어가 수행하는 기능에 대한 테스트
비기능 테스트	신뢰성, 사용성(UI/UX), 호환성, 성능, 보안 등 비기능적 품질 특성 테스트
구조 테스트	소프트웨어에 내재되어 있는 코드, 시스템의 구조 또는 아키텍처의 테스트
확인 테스트	테스트 도중 발견된 결함이 수정되었는지 확인하는 테스트
회귀 테스트	결함 수정 및 코드 수정으로 인해 다른 결함이 발생되었는지 확인하는 테스트
모니터링 테스트	제품 출시(상용화) 제품 이상 여부를 확인하는 테스트

- 테스트 단계에 따른 테스트 종류

종류	내용
단위/모듈 테스트	하나의 소프트웨어 모듈이 정상적으로 작동하는지 확인하는 테스트
통합 테스트	각 모듈을 하나로 결합하여 구성 요소 간의 연결 및 기능이 정상 작동하는지 확인하는 테스트
시스템 테스트	통합된 시스템의 테스트로 구현된 제품이 정해진 조건에 적합한지 여부를 평가하기 위하여 실제 운용과 같은 환경에서 시스템 전체 에 대해 확인하는 테스트
인수 테스트	사용자가 수용할 만한 품질을 갖추었는지 여부 를 테스트하는 것으로 실제 운영환경에서 사용될 준비가 되었는지 확인하는 테스트

- 테스트 설계 기법에 따른 테스트 종류

종류	내용
명세 기반	요구사항 기반으로 요구사항서(SRS)와 개발 설계서로부터 테스트 케이스 도출 및 테스트하는 방법
구조 기반	소프트웨어 소스 코드의 적합성과 구조적 건전성을 테스트하는 방법.
경험 기반	테스터, 개발자, 사용자 등의 지식과 경험을 중심으로 문서화하여 결함을 추정하여 테스트 케이스를 도출하는 방법

SW 테스트 유형

- 테스트 방법에 따른 테스트 종류

종류	내용
블랙박스 테스트	소프트웨어를 내면을 알 수 없는 블랙박스로 규정하고 외부에서 기능과 성능 등을 테스트
화이트박스 테스트	소프트웨어 내면을 테스트

- 테스트 실행 여부에 따른 테스트 종류

종류	내용
동적 테스트	실제 구현된 제품 또는 시스템을 직접 실행하며 테스트
정적 테스트	제품 또는 시스템이 제작되기 전에 개발 산출물을 테스트하는 것으로 리뷰나 정적분석 같은 것들이 속함

- 소스 코드를 워크스루나 인스펙션 같은 형태로 점검하는 것.
- 최근 소프트웨어는 점점 복잡해지고 빠르게 릴리즈 되고 지속적으로 통합되며 배포됨. 하지만 여러 개발자들이 개발한 컴포넌트를 통합하여 배포하는 것은 많은 시간이 요구됨.
- 이러한 이슈를 기반으로 CI(ex> 젠킨스)가 유행하기 시작했으며, 형상 관리 서버에 체크인을 하기 전에 소스 코드의 점검은 개발에 있어서 중요한 사항임.
- 직관에 의존, 리뷰어로 경험이 풍부한 개발자 필요.
- **코드 리뷰 목적**
 - 지식, 프로그램 로직 공유.
 - 공통의 코딩가이드라인을 구축
 - 결함률을 낮춤.
 - 결과적으로 코드 품질을 높여 줌.

테스트의 수행

- 테스트 케이스의 작성과 수행

유스케이스명	테스트항목	테스트케이스	테스트 케이스 ID	입력값	예상결과	사전조건	요구사항 ID	비고
I01-01	현황판	현황판 간트차트	TIA001-01	N/A	간트차트의 내용이 올바르게 표시된다.	N/A	Req1-1	
IA0-01	현황판	현황판 이슈 통계	TIA001-02	N/A	현황판의 이슈 통계가 올바르게 계산된다.	N/A	Req1-2	
IA0-01	현황판	현황판의 산출물 목록	TIA001-03	N/A	현황판의 산출물 목록이 올바르게 표시된다.	N/A	Req1-3	

단위 테스트

- 단위테스트는 단위 코드에서 문제 발생 소지가 있는 모든 부분을 테스트 하는 작업이다.
- 보통 클래스의 public method 를 테스트 한다.
- 좋은 단위 테스트란 모든 메서드를 테스트 하는 것이 아니라, 상식 수준의 확인을 통해 단위 코드가 의도한 대로 동작하는지 여부를 판단하는 단계이다.

단위 테스트 (예제로 실행해 보기)

- Junit 예제

- <숫자 목록에서 가장 큰 숫자르르 찾는 프로그램>에 대한 단위 테스트를 통해 결함을 찾아 코드를 수정하는 과정에 대한 설명.
- 1.코드 (단위 테스트 대상 클래스)구현

```
public class Largest {  
  
    public static int largest(int[] list) {  
        int index, max=Integer.MAX_VALUE;  
        for (index= 0; index <list.length-1; index++) {  
            if (list[index] > max) {  
                max = list[index];  
            }  
        }  
        return max;  
    }  
}
```

단위 테스트 (예제로 실행해 보기)

- 2.단위 테스트(Unit Test)작성 코드
-) 테스트 대상 코드 선택
-) File → New → Junit Test Case
-) 단위 테스트 작성: 코드가 원하는 대로 동작하는지 확인하기 위해 assertion 사용. 이 경우는 assert Equals 사용

```
import junit.framework.TestCase;

public class LargestTest extends TestCase {

    protected void setUp() throws Exception {
        super.setUp();
    }

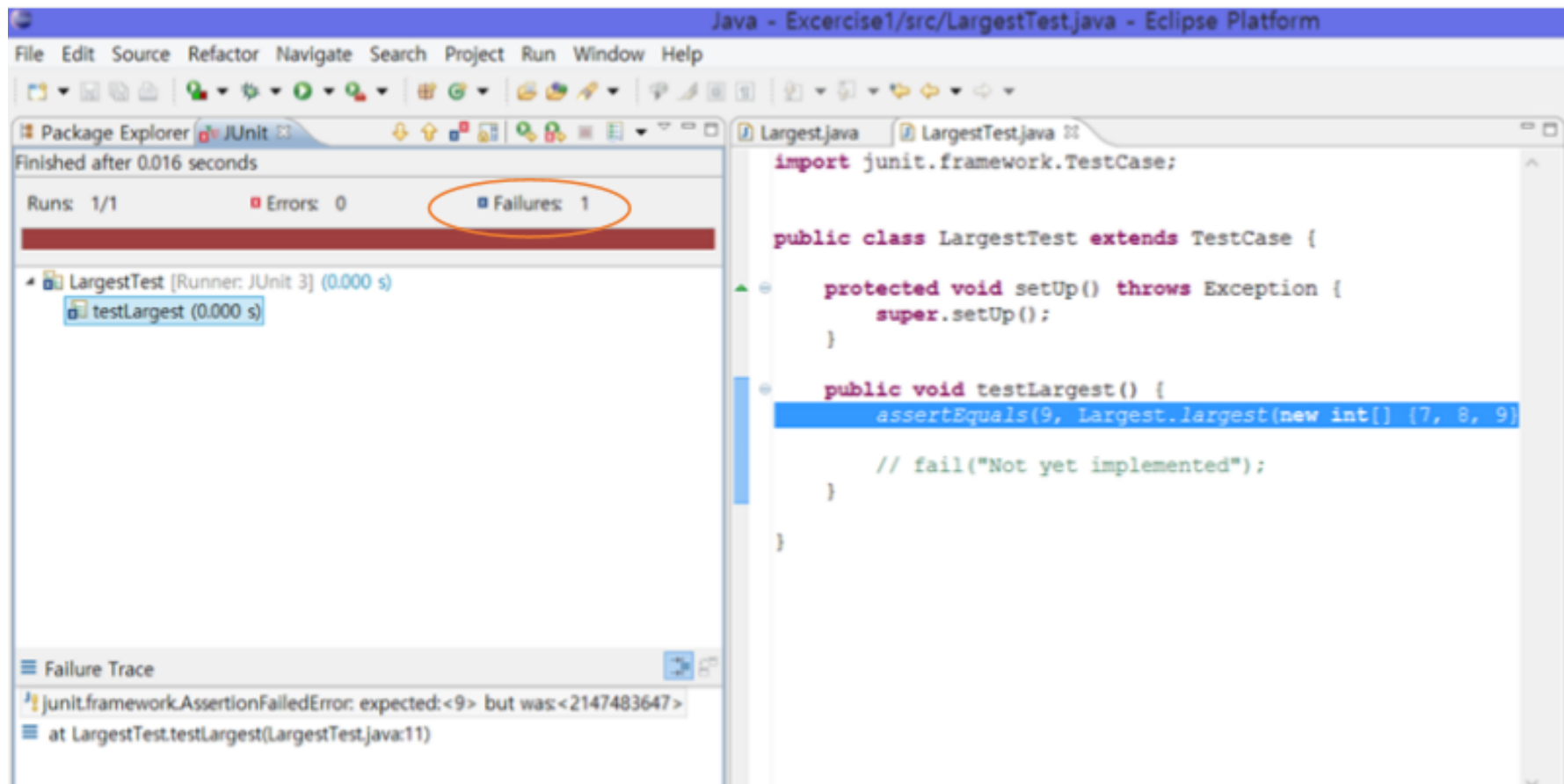
    public void testLargest() {
        assertEquals(9, Largest.largest(new int[] {7, 8, 9}));

        // fail("Not yet implemented");
    }

}
```


단위 테스트 (예제로 실행해 보기)

- 3. Junit Test 수행 (9가 나와야 하는데 2147483647이라는 쓰레기값이 출력됨.)



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left indicates that the test 'testLargest' failed. The main editor displays the source code of 'LargestTest.java', which extends 'junit.framework.TestCase'. The test method 'testLargest()' is highlighted, showing an assertion failure: 'assertEquals(9, Largest.largest(new int[] {7, 8, 9})'.

```

import junit.framework.TestCase;

public class LargestTest extends TestCase {

    protected void setUp() throws Exception {
        super.setUp();
    }

    public void testLargest() {
        assertEquals(9, Largest.largest(new int[] {7, 8, 9})

        // fail("Not yet implemented");
    }
}
    
```

The failure trace at the bottom shows: 'junit.framework.AssertionFailedError: expected:<9> but was:<2147483647>' at 'LargestTest.testLargest(LargestTest.java:11)'.

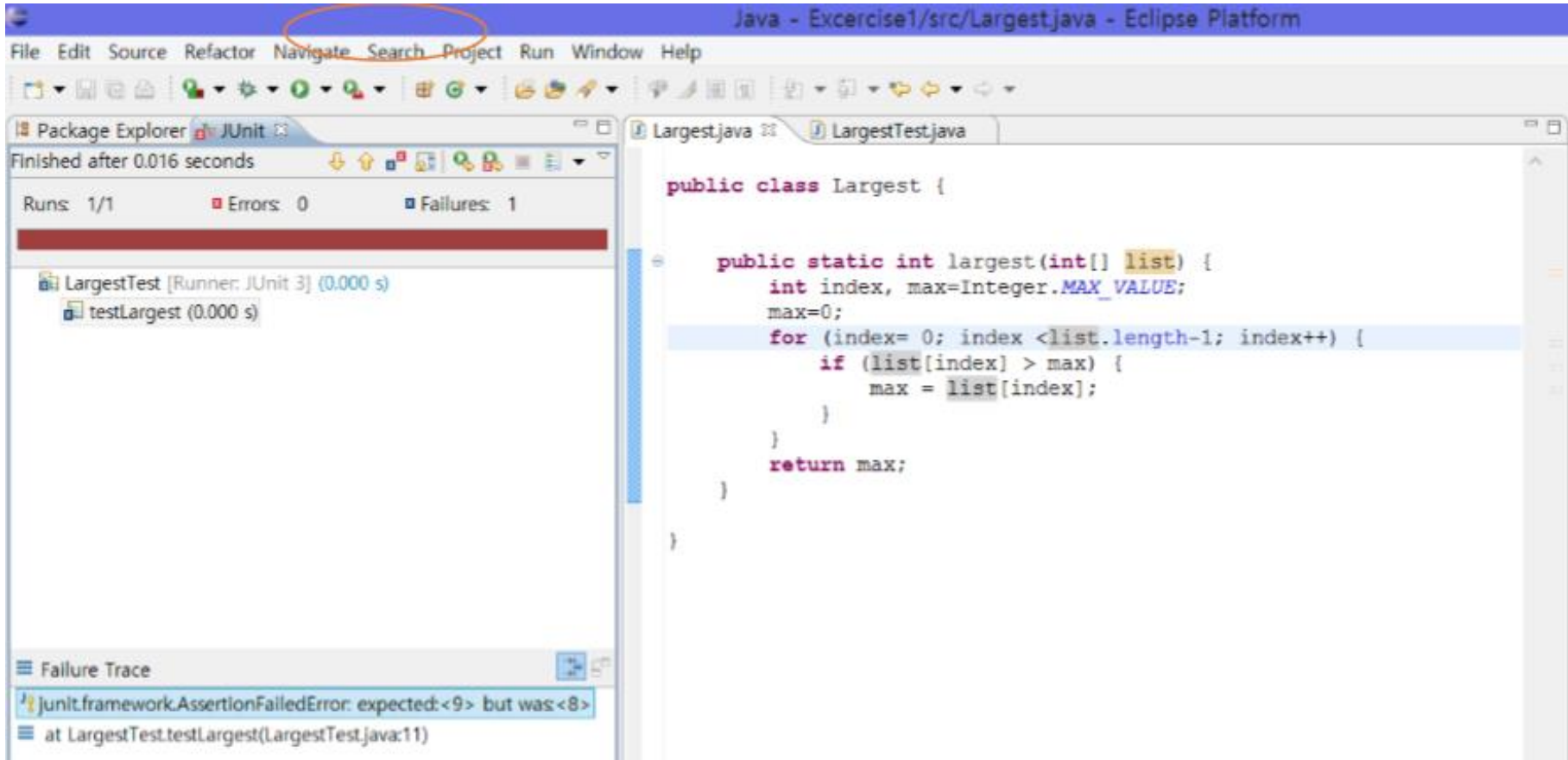
단위 테스트 (예제로 실행해 보기)

- 4. 테스트 대상 코드 수정.
- max값에 초기화가 필요. max =0 코드 추가

```
public class Largest {  
  
    public static int largest(int[] list) {  
        int index, max=Integer.MAX_VALUE;  
        max=0;  
        for (index= 0; index <list.length-1; index++) {  
            if (list[index] > max) {  
                max = list[index];  
            }  
        }  
        return max;  
    }  
}
```

단위 테스트 (예제로 실행해 보기)

- 5. Junit Test 재 수행 (9가 나와야 하는데 8이 나와서 또 fail.)



Java - Exercise1/src/Largest.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 0.016 seconds

Runs: 1/1 Errors: 0 Failures: 1

LargestTest [Runner: JUnit 3] (0.000 s)

testLargest (0.000 s)

Failure Trace

JUnit4.framework.AssertionFailedError: expected:<9> but was<8>
at LargestTest.testLargest(LargestTest.java:11)

```
public class Largest {  
  
    public static int largest(int[] list) {  
        int index, max=Integer.MAX_VALUE;  
        max=0;  
        for (index= 0; index <list.length-1; index++) {  
            if (list[index] > max) {  
                max = list[index];  
            }  
        }  
        return max;  
    }  
}
```

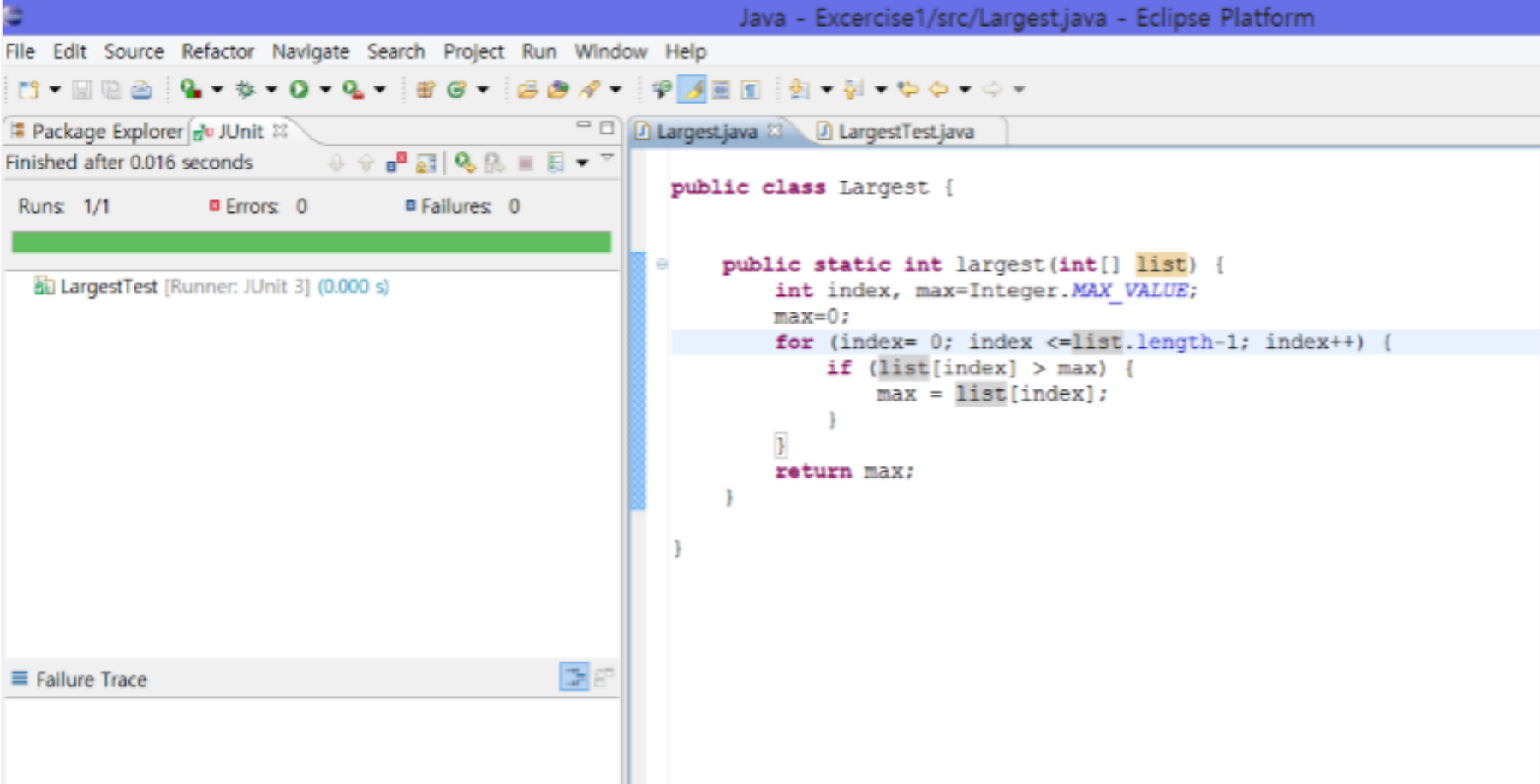
단위 테스트 (예제로 실행해 보기)

- 6. 테스트 대상 코드 재 수정
- list 배열의 마지막 값의 크기가 비교 되지 않음 확인
- $\text{index} < \text{list.length}-1$ 를 $\text{index} \leq \text{list.length}-1$ 또는 $\text{index} < \text{list.length}$ 로 변경

```
public class Largest {  
  
    public static int largest(int[] list) {  
        int index, max=Integer.MAX_VALUE;  
        max=0;  
        for (index= 0; index <=list.length-1; index++) {  
            if (list[index] > max) {  
                max = list[index];  
            }  
        }  
        return max;  
    }  
}
```

단위 테스트 (예제로 실행해 보기)

- 7. Junit Test 재 수행. (성공)



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Package Explorer on the left shows the project structure with 'JUnit' and 'LargestTest' visible. The JUnit runner output shows 'Finished after 0.016 seconds' and 'Runs: 1/1', 'Errors: 0', 'Failures: 0'. The main editor displays the source code for 'Largest.java' and 'LargestTest.java'. The code for 'Largest.java' is as follows:

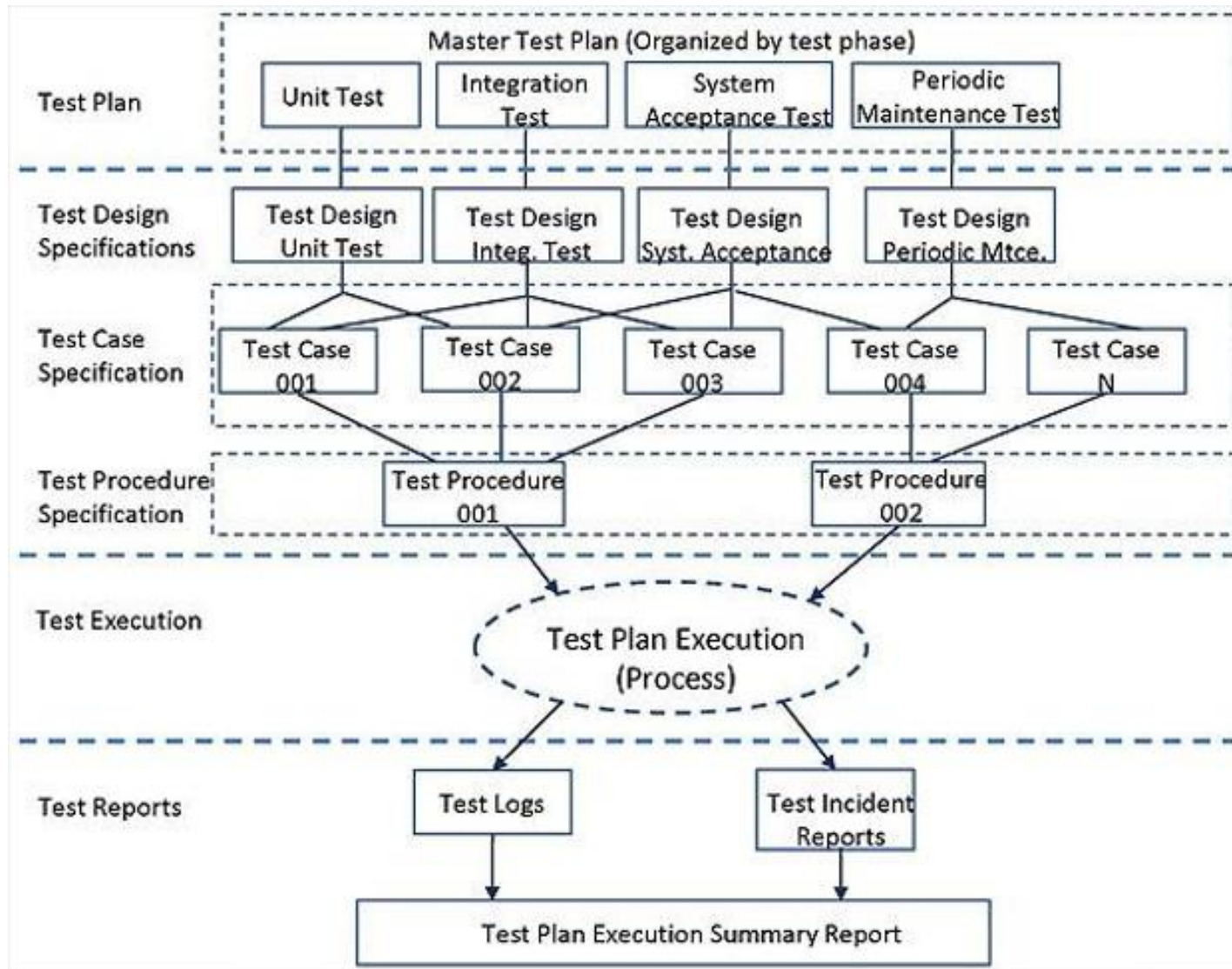
```
public class Largest {  
  
    public static int largest(int[] list) {  
        int index, max=Integer.MAX_VALUE;  
        max=0;  
        for (index= 0; index <=list.length-1; index++) {  
            if (list[index] > max) {  
                max = list[index];  
            }  
        }  
        return max;  
    }  
}
```

테스팅 vs 디버깅.

- SW테스트: 테스트는 소프트웨어에 존재하는 오류를 발견하는 행위
- 디버깅: SW에 존재하는 오류에 발생원인을 분석하는 행위



테스트 문서화 (Test Documentation)



테스트 문서화 (Test Documentation)

- IEEE 829 및 ISO 29119 표준에 명시됨.
- Test Specification
 - Test Plan (테스트 계획서) :테스트를 관리, 예약 및 실행하는 방법에 대해 설명.
 - Test Design Specification (테스트 설계서) : 논리적으로 요구 사항이나 기능을 검토하여 테스트해야 할 사항을 정의.
 - Test case specification (테스트 케이스): 사전 조건 및 예상 결과를 추가하여 테스트 조건을 테스트 케이스로 변환.
 - Test procedure (테스트 절차서): 실전에서 테스트가 어떻게 실행되는지 설명.
 - Test item transmittal report (테스트 항목 전송 보고서): 테스트 항목 전송 보고서는 발표 된 테스트 항목을 지정.
- Test Execution
 - 테스트 로그는 테스트의 세부 사항을 시간순으로 기록하는 감사 추적.
 - 테스트 사건 보고서에는 조사해야 할 예상치 못한 사건 및 행동에 대한 세부 정보가 기록됨.
- Test reporting
 - Test Summary Report (테스트 결과 보고서) : 테스트 요약 및 평가.

- 소프트웨어 개발 생명주기(SDLC)
- 소프트웨어를 개발해 나가는 단계나 과정을 말함.
- 생명주기 모델 대표 : V Model
- → V 모델의 경우 코딩 단계 이전의 개발 단계와 테스트 단계가 서로 대응되어서 진행된다.
코딩 이후의 테스트 단계들을 테스트 레벨 이라고 부르며 각 레벨별로 수행하는 테스트 기법 또는 방법이 달라지게 된다.

V-Model

