

프로그래밍 역량 강화 전문기관, 민코딩

클린코드 - 2부



목차

1. Comment
2. Formatting
3. Unit Test

주석

Comment

주석은 유지보수 되지 않는다.

주석은 유지보수 되지 않는다.

- 동작에 영향을 끼치지 않기 때문
- 코드 작성자가 아닌 사람이,
다른사람이 작성한 주석을 건드리지 않음
- 주석을 작성한 이후,
코드는 추가되었지만, 주석에 설명이 빠진다.

유지보수를 하다보면
부정확해지는 주석이 있는 것 보다는,
없는 것이 더 좋다.

두 코드를 볼 때, 독자를 위한 코드는?

```
// 직원이 복지 혜택을 받을 조건이 되는지 검사한다.  
if ((employee.flags & HOURLY_FLAG) &&  
    (employee.age > 65))
```

```
if (employee.isPossible복지혜택( ))
```

메서드명으로 표현될 수 있는 경우

주석대신 메서드로 설명이 가능하면, 메서드로 표현을 권장

```
// 직원이 복지 혜택을 받을 조건이 되는지 검사한다.  
if ((employee.flags & HOURLY_FLAG) &&  
    (employee.age > 65))
```

```
if (employee.isPossible복지혜택( ))
```

피하면 좋을 주석 1

팀에서 정한 의무적으로 다는 주석

- 모든 함수에 Javadocs를 달거나,
모든 변수에 주석을 달아야 한다는 규칙은 피하는 것이 좋다.

이력 남기기

- 주석으로 날짜별 이력 남기는 경우, 유지보수 안됨

주저리 주저리 주석

- 같은말 반복하고,
이해하는데 큰 도움 안되는 의미 없는 주석

피하면 좋을 주석 2

저자를 표시하는 주석

- `//이런 기능 내가 추가함 by inho.choi ^^v`
- 버전관리도구를 믿고, 남기지말자.

주석으로 처리한 코드

- 타인이 보기에는 무언가 사연이 있는 듯하여, 지울 수 없다.
- 버전관리도구를 믿고, 남기지말자.

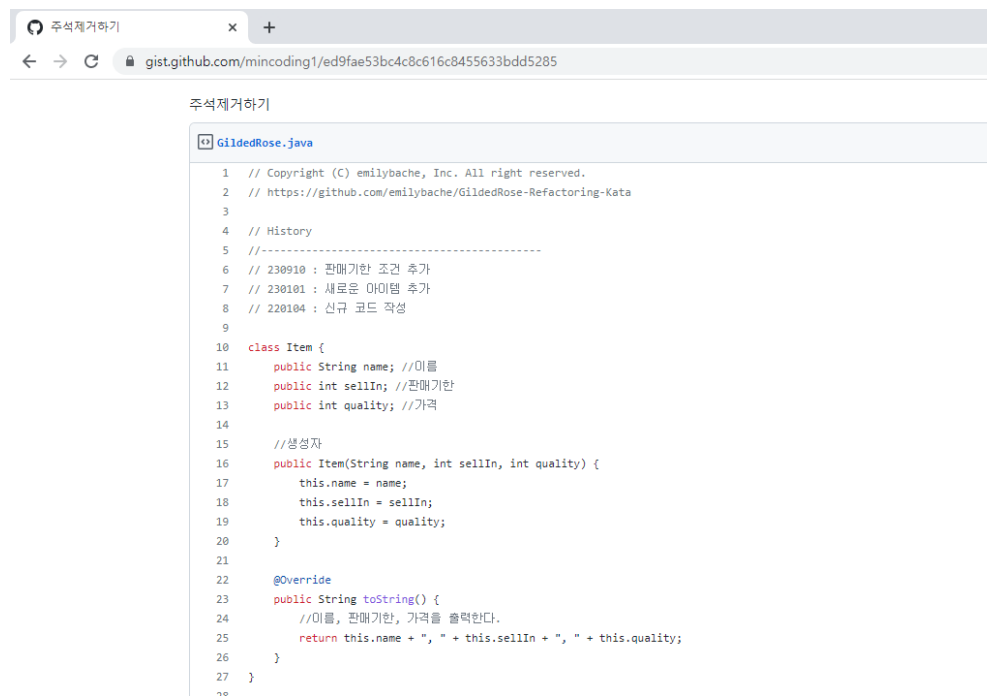
괄호에 다는 주석

- `} //while`
- `} //for`

[도전] 주식 정리하기

필요하다고 생각하는 주식은 남기고,
불필요한 주식은 삭제하자.

- <https://gist.github.com/mincoding1/ed9fae53bc4c8c616c8455633bdd5285>



```
주식제거하기
GildedRose.java
1 // Copyright (C) emilybache, Inc. All right reserved.
2 // https://github.com/emilybache/GildedRose-Refactoring-Kata
3
4 // History
5 //-----
6 // 230910 : 판매기한 조건 추가
7 // 230101 : 새로운 아이템 추가
8 // 220104 : 신규 코드 작성
9
10 class Item {
11     public String name; //이름
12     public int sellIn; //판매기한
13     public int quality; //가격
14
15     //생성자
16     public Item(String name, int sellIn, int quality) {
17         this.name = name;
18         this.sellIn = sellIn;
19         this.quality = quality;
20     }
21
22     @Override
23     public String toString() {
24         //이름, 판매기한, 가격을 출력한다.
25         return this.name + ", " + this.sellIn + ", " + this.quality;
26     }
27 }
28
```

Formatting

뉴스 기사처럼

뉴스 기사

질서 정돈된 뉴스 기사는
독자들이 필요한 내용을 빠르게 이해할 수 있다.



뉴스 기사의 장점을 소스코드에 적용

Title 은 메서드 명으로 비유할 수 있다.

- 어떤 내용인지 파악할 수 있도록한다.
- 반전이 없도록한다.

타이틀로 설명이 부족하다면,
주석으로 부제를 적절히 넣어주자.



질서 정돈

코드를 읽는 사람들을 위해
질서와 정돈속에서 소스코드를 구현한다.

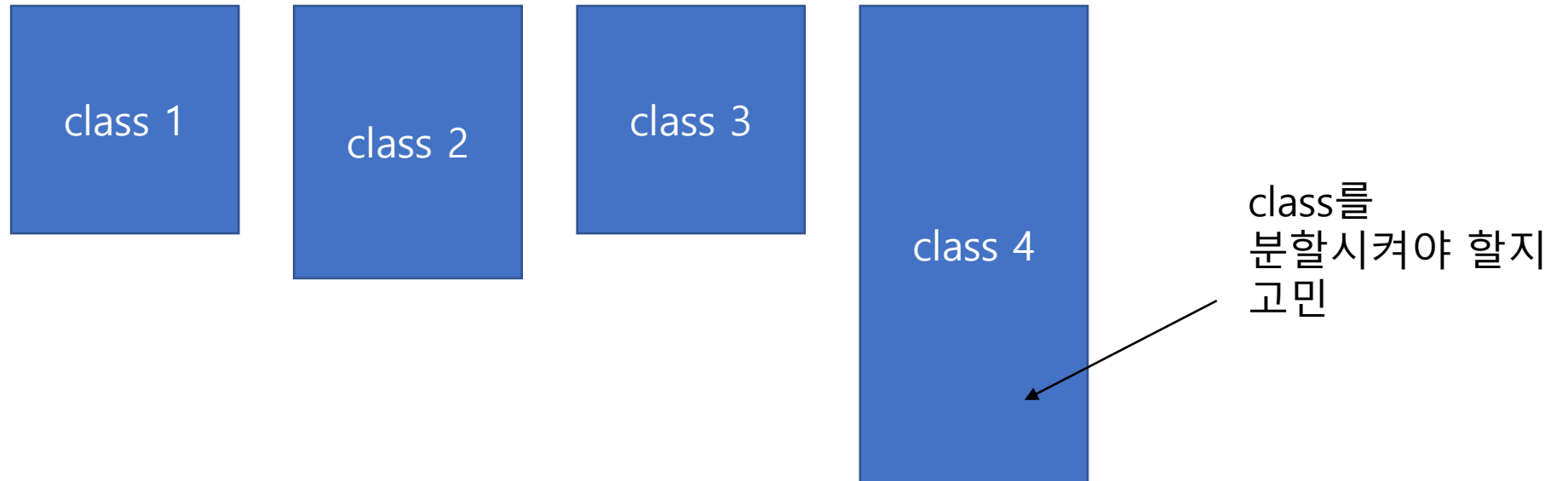
- ✓ 코드 Format에 대한 팀 규칙을 정하고,
- ✓ 그 규칙을 착실히 따르고,
- ✓ IDE에 그 규칙을 지정하는 것을 권장한다.



이번 챕터는 코드 내용 보다는,
코드의 질서 & 정돈에 관한 권장 규칙

Class 의 크기가 무리하게 크지 않도록 한다.

클래스 파일의 크기가 유별나게 크지 않도록 한다.



소스코드, 가로 스크롤

모니터 사이즈는 다들 다르지만,
일반적인 화면 크기에서 스크롤을 넘기지 않도록 한다.

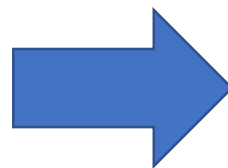
ex) 120 글자 수준

소스코드 소스코드 소스코드 소스코드 소스코드 소스코드 소스코드...
소스코드

정렬

소스코드를 세로 라인으로 정렬하려고 노력하지 않는다.
끊임없이 IDE 자동 기능과 싸워야한다.

```
int      sock;  
Socket   socketInstance;  
OutputStream outStream;  
InputStream inputStream;
```



```
int sock;  
Socket socketInstance;  
OutputStream outStream;  
InputStream inputStream;
```


글작성시 단락의 중요성

단락 : 내용상 끊은 하나의 토막

- 각 단락별로 엔터로 구분해야, 독자들이 빠르게 읽기 좋다.

어쩌면 산문이 그 사람의 글쓰기 실력을 더 적나라하게 잘 보여주는 것 같다.

어떻게 할까.

나는 고민한다. 일단 내 방식대로 편집하고 나서, 교정본을 보낼까, 아니면 일단 작가의 초집해서 그냥 보내버릴까.

엔터 친 곳을 모두 붙이면서, 붙어 있는 곳을 모두 엔터 치면서 수제비 반죽 뜨듯 적당한 크이를 만드는 일, 글을 몇 번이나 다시 읽어야 한다.

1안은 나의 수고를 전제로 해야하는 것이고, 2안은 눈감아버리는 것이지만, 오랜 경험상 전해서 보내든, 후자를 편집해서 보내든 작가는 생각보다 관심이 없다.

왜냐고? 작가 눈에는 모든 글이 예뻐보이니까!

책을 기다리는 작가는, 글을 보는 것이 아니라, (안성필) 책을 본다. 책을 준비하는 편집자는 기 전에 (완성시켜야 하는)글을 본다.

글이 완성되어야 다음 페이지로 넘어갈 수 있다. 원고가 출판사로 넘어가는 순간부터, 이제 편집자가 더 글을 많이 읽고 더 고민하게 된다.

시나 소설을 쓰는 작가들이 출판사에 오래 재직하지 못하고 방향하는 이유가 바로 여기에 있다. 글쓰기할 에너지를 편집에 다 쏟기 때문이다.

물론 어느 정도 이골(?)이 난 베테랑들은 본인의 글쓰기에 전혀 영향을 받지 않는다. 남의

이면서, 붙어 있는 곳을 모두 엔터 치면서 수제비 반죽 뜨듯 적당한 크기의 덩어리를 만드는 일, 글을 몇 번이나 다시 읽어야 한다. 1안은 나의 수고를 전제로 해야하는 것이고, 2안은 눈감아버리는 것이지만, 오랜 경험상 전자를 편집해서 보내든, 후자를 편집해서 보내든 작가는 생각보다 관심이 없다. 왜냐고? 작가 눈에는 모든 글이 예뻐보이니까! 책을 기다리는 작가는, 글을 보는 것이 아니라, (안성필) 책을 본다. 책을 준비하는 편집자는 책을 보기 전에 (완성시켜야 하는)글을 본다. 글이 완성되어야 다음 페이지로 넘어갈 수 있다. 원고가 출판사로 넘어가는 순간부터, 이제 작가보다 편집자가 더 글을 많이 읽고 더 고민하게 된다. 시나 소설을 쓰는 작가들이 출판사에 오래 재직하지 못하고 방향하는 이유가 바로 여기에 있다. 본인 글쓰기할 에너지

작가의 글을 편집하다가, 이마를 손으로 짚는 경우가 종종 있다. 비문이나 맞춤법, 띄어쓰기 등의 문제가 서둘러 수정하면 그만이지만, 이상한 곳에서 행과 연을 나누거나 단락이 나뉘져 있으면, 마우스 포인터가 부들부들 떠다. 어찌지,하고 말이다. 이상한 곳을 어떻게 아냐고? 간단하다. 직접 소리 내어 읽어보면 누구나 쉽게 안다. 한 덩어리로 읽어야 할 곳이 띄어져 있거나, 숨차도록 뻑뻑하게 글로 이어져 있다면, 고개를 갸웃할 수밖에 없다. 이유는 아마 다음의 두 가지 중 하나일 것이다. 글의 흐름을 작가가 컨트롤하고 있지 못하거나, 작가가 전혀 의식하지 못하거나. 시의 경우, 의도적인 행갈이와 연결이가 있으니 '나름' 존중해야겠지만, 산문의 경우는 다르다. 어쩌면 산문이 그 사람의 글쓰기 실력을 더 적나라하게 잘 보여주는 것 같다.

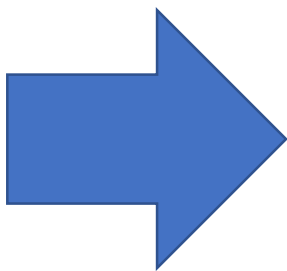
어떻게 할까. 나는 고민한다. 일단 내 방식대로 편집하고 나서, 교정본을 보낼까, 아니면 일단 작가의 초고대로 편집해서 그냥 보내버릴까. 엔터 친 곳을 모두 붙이면서, 붙어 있는 곳을 모두 엔터 치면서 수제비 반죽 뜨듯 적당한 크기의 덩어리를 만드는 일, 글을 몇 번이나 다시 읽어야 한다. 1안은 나의 수고를 전제로 해야하는 것이고, 2안은 눈감아버리는 것이지만, 오랜 경험상 전자를 편집해서 보내든, 후자를 편집해서 보내든 작가는 생각보다 관심이 없다. 왜냐고? 작가 눈에는 모든 글이 예뻐보이니까!

책을 기다리는 작가는, 글을 보는 것이 아니라, (안성필) 책을 본다. 책을 준비하는 편집자는 책을 보기 전에 (완성시켜야 하는)글을 본다. 글이 완성되어야 다음 페이지로 넘어갈 수 있다. 원고가 출판사로 넘어가는 순간부터, 이제 작가보다 편집자가 더 글을 많이 읽고 더 고민하게 된다. 시나 소설을 쓰는 작가들이 출판사에 오래 재직하지 못하고 방향하는 이유가 바로 여기에 있다. 본인 글쓰기할 에너지

개행

여러줄의 묶음은 완결된 하나의 생각을 나타낸다.

```
package fitnessse.wikitext.widgets;
import java.util.regex.*;
public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?''';
    private static final Pattern pattern = Pattern.compile("'''.+?'''",
        Pattern.MULTILINE + Pattern.DOTALL);
    public BoldWidget(ParentWidget parent, String text) throws Exception {
        super(parent);
        Matcher match = pattern.matcher(text); match.find();
        addChildWidgets(match.group(1));
    }
    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```



```
package fitnessse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?''';
    private static final Pattern pattern = Pattern.compile("'''.+?'''",
        Pattern.MULTILINE + Pattern.DOTALL
    );

    public BoldWidget(ParentWidget parent, String text) throws Exception {
        super(parent);
        Matcher match = pattern.matcher(text);
        match.find();
        addChildWidgets(match.group(1));
    }

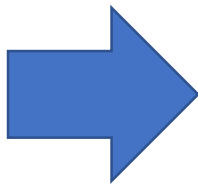
    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```

세로 밀집도

연관성이 있는 내용은 붙여주는 것이 좋다.

- 의미없는 주석으로 인해, 단락이 끊어졌다.

```
public class ReporterConfig {  
    /**  
     * The class name of the reporter listener  
     */  
    private String m_className;  
  
    /**  
     * The properties of the reporter listener  
     */  
    private List<Property> m_properties = new ArrayList<Property>();  
    public void addProperty(Property property) {  
        m_properties.add(property);  
    }  
}
```



```
public class ReporterConfig {  
    private String m_className;  
    private List<Property> m_properties = new ArrayList<Property>();  
  
    public void addProperty(Property property) {  
        m_properties.add(property);  
    }  
}
```

개념의 유사도

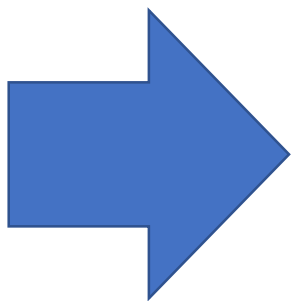
비슷한 개념의 메서드들은
함께 모여 배치하자.

```
public class Assert {  
    static public void assertTrue(String message, boolean condition) {  
        if (!condition)  
            fail(message);  
    }  
  
    static public void assertTrue(boolean condition) {  
        assertTrue(null, condition);  
    }  
  
    static public void assertFalse(String message, boolean condition) {  
        assertTrue(message, !condition);  
    }  
  
    static public void assertFalse(boolean condition) {  
        assertFalse(null, condition);  
    }  
}
```

들여쓰기

Line 수를 줄이기 위한 최고의 노력을 하지말자.

```
public class CommentWidget extends TextWidget {  
    public static final String REGEXP = "^#[^\\r\\n]*(?:(?:\\r\\n)|\\n|\\r)?";  
  
    public CommentWidget(ParentWidget parent, String text){super(parent, text);}  
    public String render() throws Exception {return "";}  
}
```



```
public class CommentWidget extends TextWidget {  
    public static final String REGEXP = "^#[^\\r\\n]*(?:(?:\\r\\n)|\\n|\\r)?";  
  
    public CommentWidget(ParentWidget parent, String text){  
        super(parent, text);  
    }  
  
    public String render() throws Exception {  
        return "";  
    }  
}
```

[도전] Formatting 실습

읽기 좋은 형태로 맞추어보자.

- <https://gist.github.com/mincoding1/316a1f630eda65ee33747ab8fea698bb>



Unit Test 개요

모듈 단위의 Testing

배경지식, Unit Test 란?

✓ **모듈 단위로, Testing을 하는 것**

입력값을 넣어 보았을 때,

결괏값이 기대값과 동일하게 나오는지 확인한다.

✓ **주로 검증팀 보다는, 개발팀에서 Test를 수행**



Unit Test 종류

xUnit

- 스몰토크의 SUnit 에서 파생된 테스트 프레임워크 이름
- 켄트백이 설계함

언어별 가장 인기있는 Unit Test Framework

- C++ : Google Test (xUnit 기반)
- java : junit (xUnit 기반)

Unit Test 도구 목록

- https://en.Wikipedia.org/wiki/List_of_unit_testing_frameworks

Java 진영에서 가장 많이 사용되는 Unit Test Framework

- 켄트백(TDD 창시자)과 에릭감마(GoF 중 한명)가 만든 Java Test Framework
- 최신버전 JUnit5이며, Java 8 이상 버전이 요구됨

JUnit 5 사용 가이드 (Github & Document)

- <https://junit.org/junit5/docs/current/user-guide/>
- <https://github.com/junit-team/junit5/>

JUnit 4

- Legacy 시스템에서 사용하기 있기에, 여전히 사용되는 버전
- 마지막 release : 21년 2월

JUnit 5

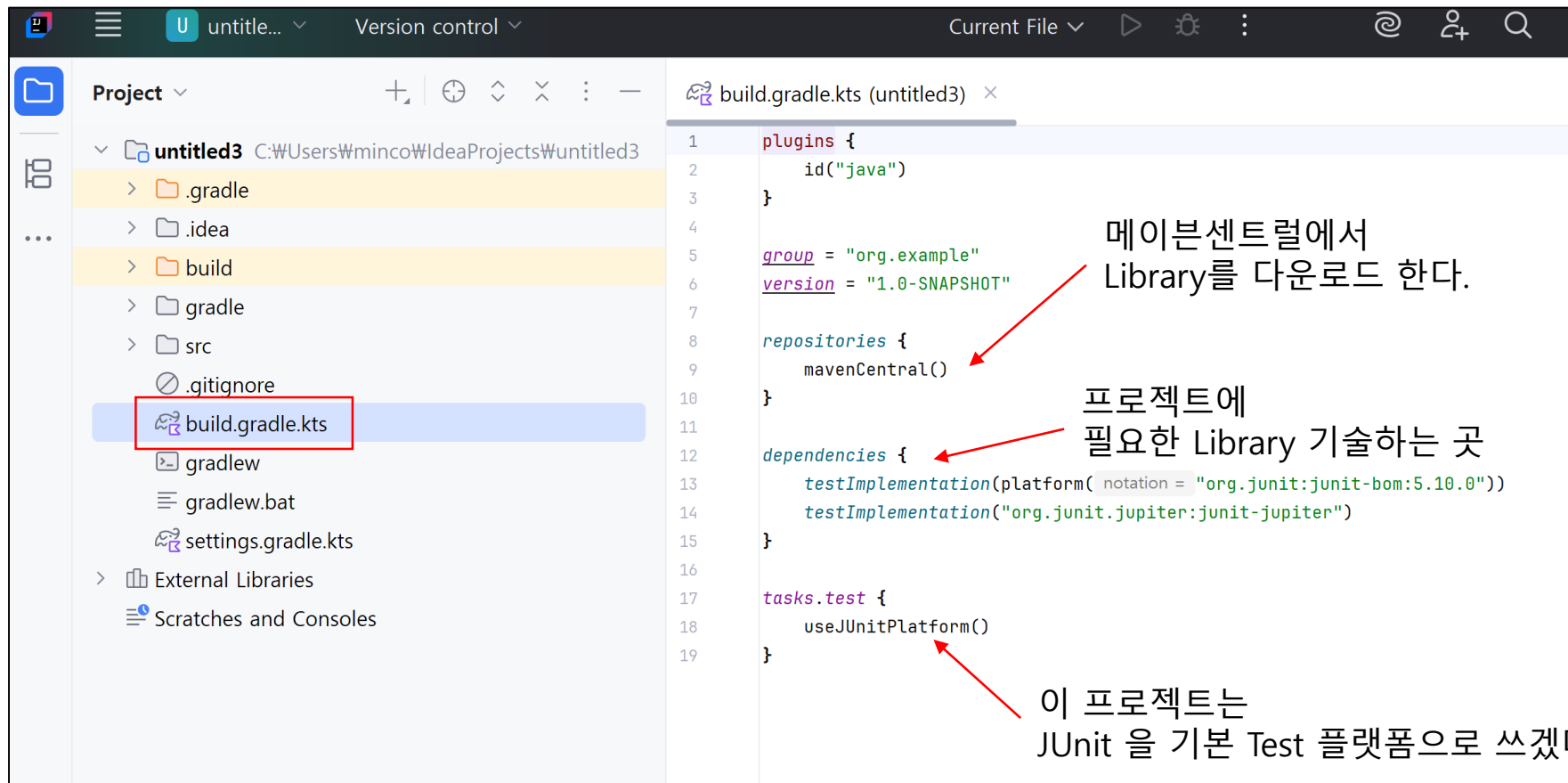


- 2017년 9월 공식 버전 출시 이후 가장 많이 사용되는 버전
- 람다, 스트림을 적극적으로 지원한다.
- @DisplayName으로 가독성 좋은 TC 이름 작성 가능

Gradle 프로젝트 생성시

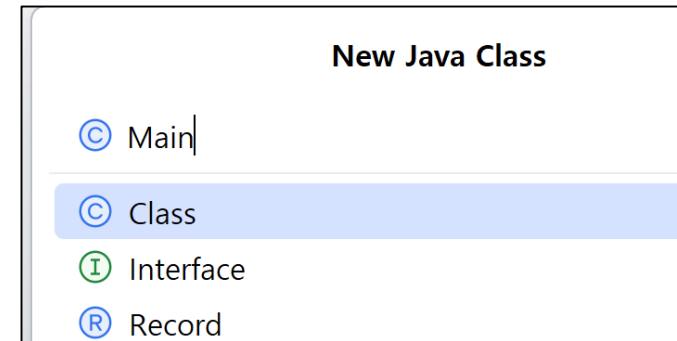
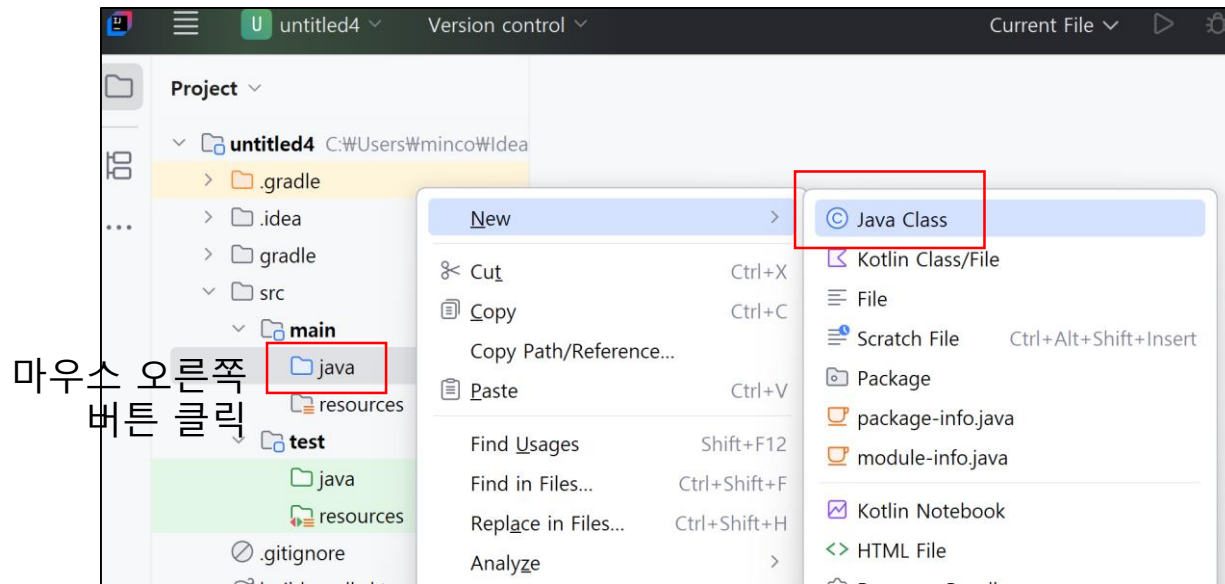
Build.gradle.kts에 이미 Junit 5 버전이 기본으로 적혀 있음

- Maven에서 pom.xml 과 같은 역할을 하는 파일
- 이 파일에서 필요한 Library를 적고, 저장하면 자동으로 다운로드 후 설치된다.



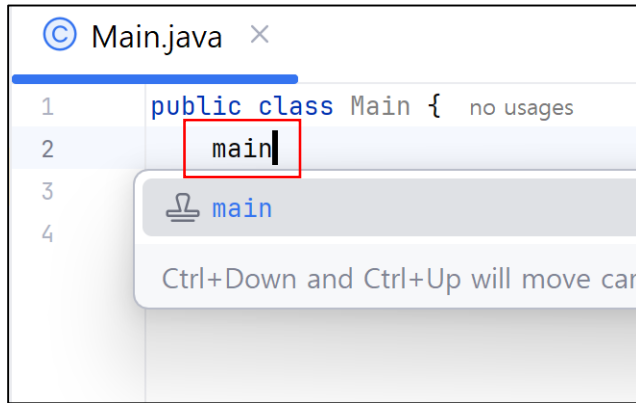
Main.java 생성 / main 함수 생성

Main.java 파일을 src/main/java/ 에 생성한다.



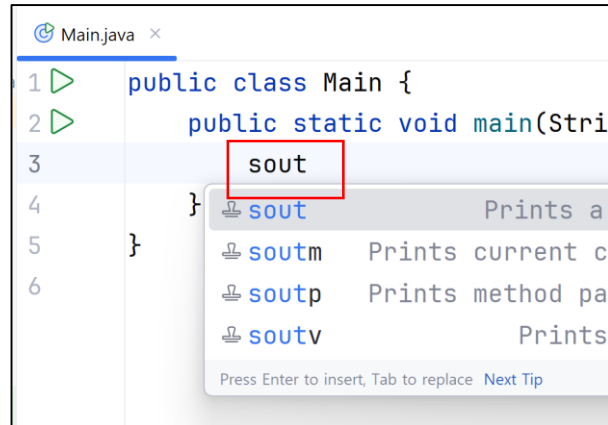
Main.java 파일 생성

빌드 테스트



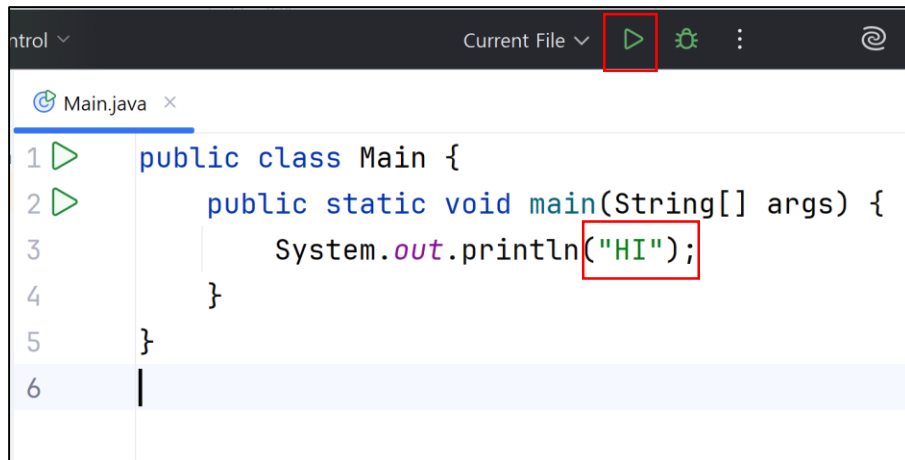
```
1 public class Main { no usages
2     main
3
4
```

main 기입 후 [TAB]키 누르기




```
1 public class Main {
2     public static void main(Stri
3         sout
4     }
5
6
```

sout 기입 후 [TAB]키 누르기



```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("HI");
4     }
5 }
6
```

"HI" 출력 코드 입력 후 Run 버튼 클릭

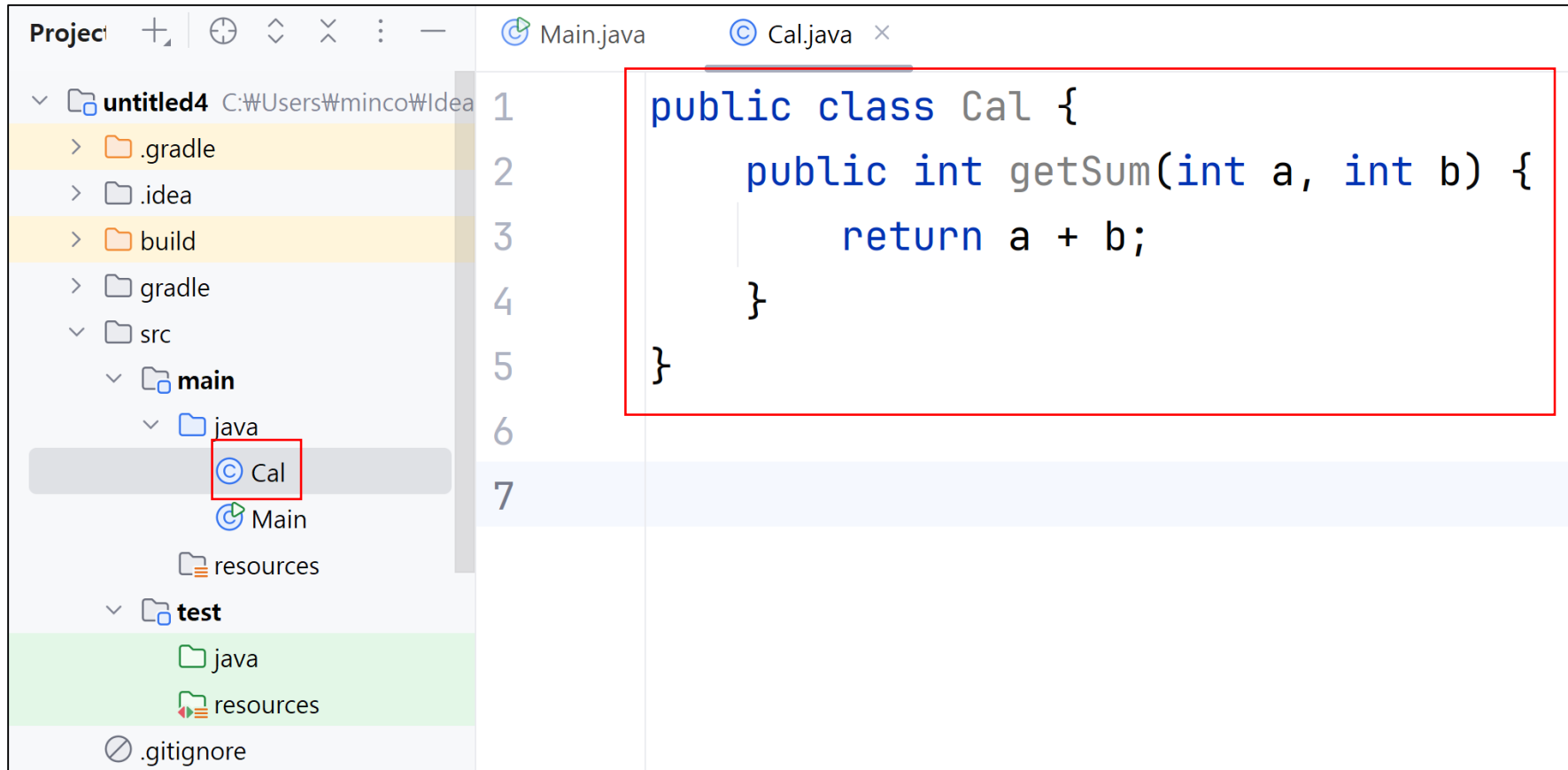


```
Run untitled4 [:Main.main()]
untitled4 [:Main.main()]: successf 2 sec, 439 ms
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :Main.main()
HI
BUILD SUCCESSFUL in 2s
2 actionable tasks: 2 executed
오후 7:30:55: Execution finished ':Main.main()'.
```

동작 확인하기

테스트 대상이 되는 파일 추가하기

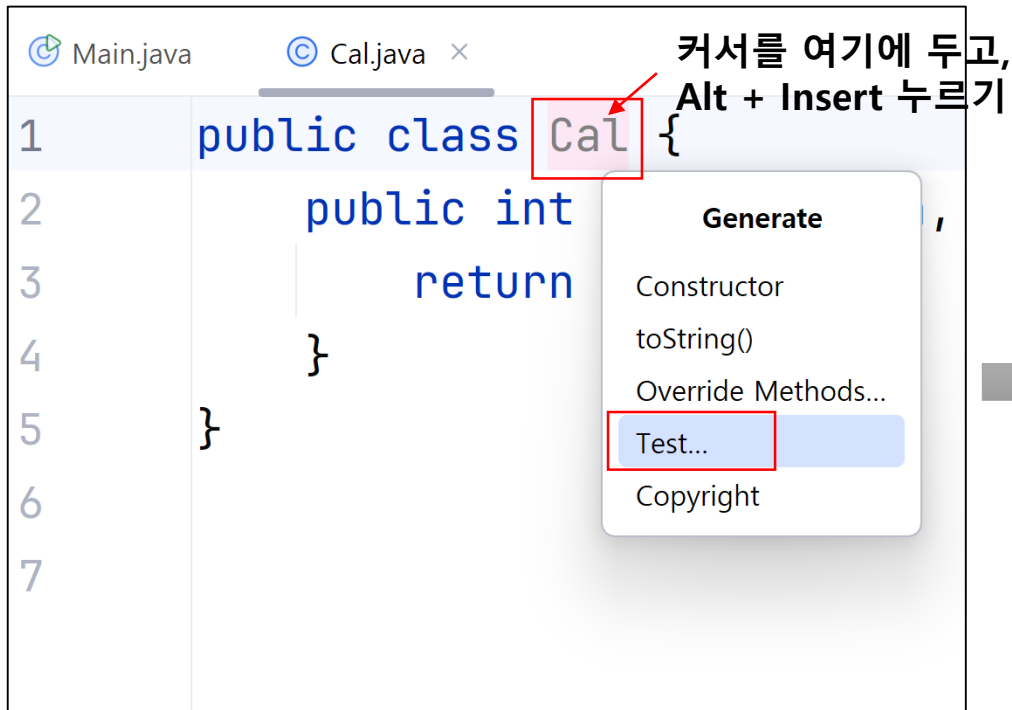
1. Cal.java 파일을 추가한다.
2. 테스트를 위한 기본 코드를 작성한다.



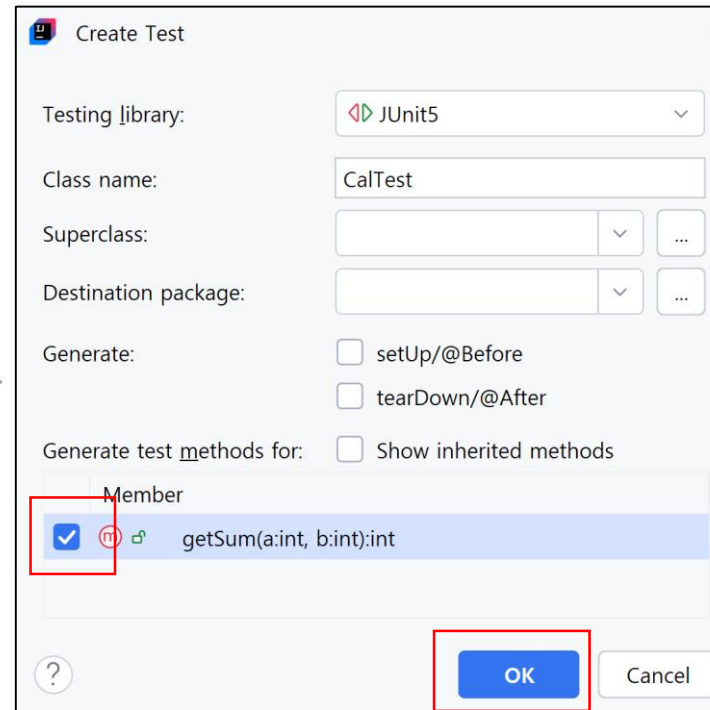
테스트 파일 생성

Cal 클래스를 클릭 후 Alt + Insert 키를 누른다.
메뉴에서 Code > Generate를 눌러도 된다.

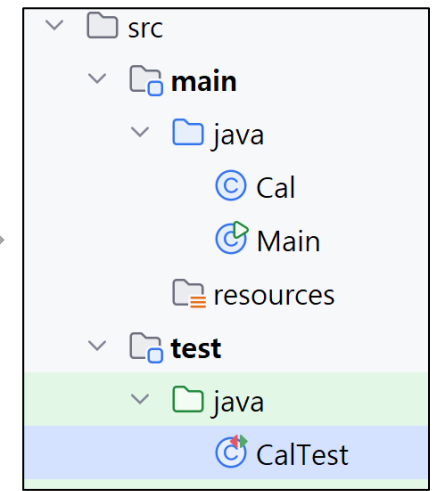
Test 파일이 자동 생성된다.



Generate > Test... 클릭



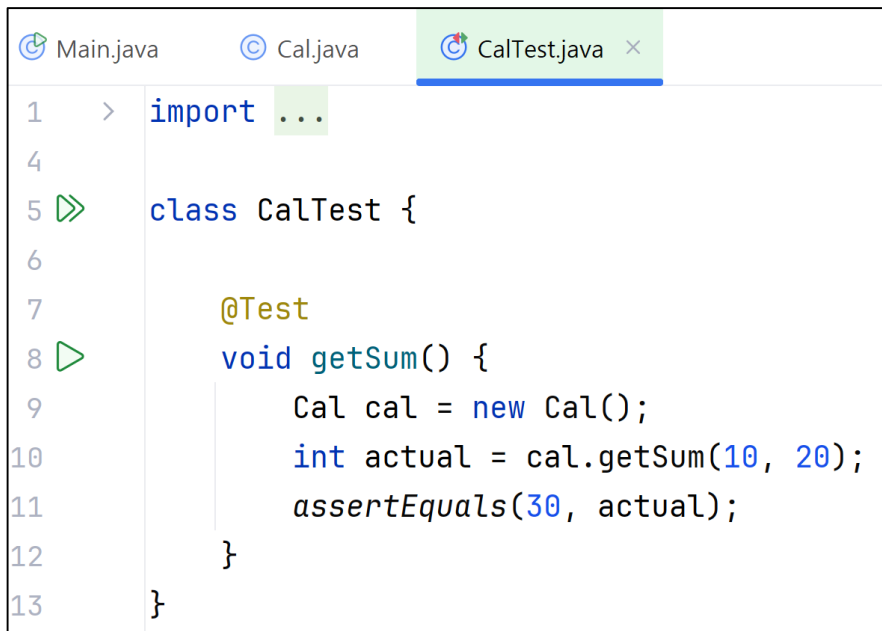
테스트할 대상을 선택한다.



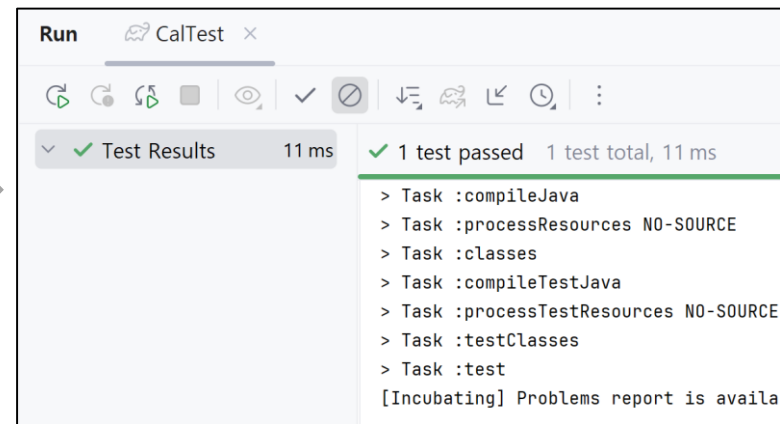
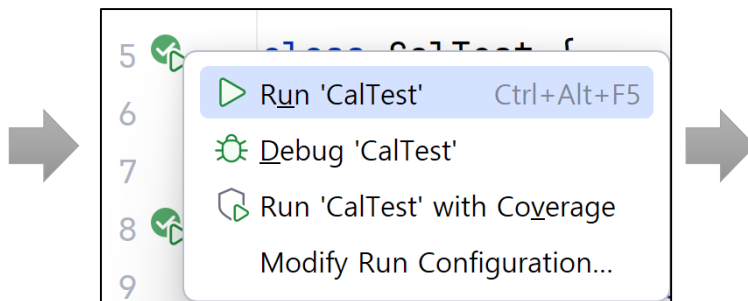
CalTest.java 파일
자동생성 된다.

테스트 기본 코드 작성

1. 테스트 코드를 확인한다.
2. 테스트 실행시 즉시, 결과를 확인할 수 있다.



```
1  > import ...
4
5  >> class CalTest {
6
7      @Test
8  > void getSum() {
9      Cal cal = new Cal();
10     int actual = cal.getSum(10, 20);
11     assertEquals(30, actual);
12 }
13 }
```



Unit Test

깨끗한 유닛테스트 만들기

테스트코드도 깨끗하게 유지하자.

- ✓테스트 코드가 지저분하면, 관리 되고 있는 것 대비, 유지보수 능력이 떨어진다. 결국 실제 코드도 망가질 수 있다.

BUILD - OPERATE - CHECK 패턴

테스트코드를 보기 편하게 구역을 나눈다.

1. BUILD
2. OPERATE
3. CHECK

**AAA (Arrange, Act, Assert) 로 구역을 나누어
Unit Test를 작성한다.**

유의사항 1. 테스트 코드도 유지보수해야한다.

```
public void testGetPageHierarchyAsXml() throws Exception {
    crawler.addPage(root, PathParser.parse("PageOne"));
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));
    crawler.addPage(root, PathParser.parse("PageTwo"));

    request.setResource("root");
    request.addInput("type", "pages");
    Responder responder = new SerializedPageResponder();
    SimpleResponse response =
        (SimpleResponse) responder.makeResponse(new FitNesseContext(root), request);
    String xml = response.getContent();

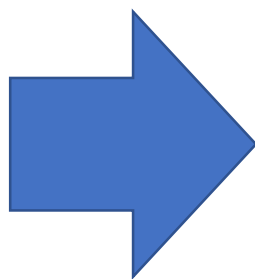
    assertEquals("text/xml", response.getContentType());
    assertSubString("<name>PageOne</name>", xml);
    assertSubString("<name>PageTwo</name>", xml);
    assertSubString("<name>ChildOne</name>", xml);
}

public void testGetPageHierarchyAsXmlDoesntContainSymbolicLinks() throws Exception {
    WikiPage pageOne = crawler.addPage(root, PathParser.parse("PageOne"));
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));
    crawler.addPage(root, PathParser.parse("PageTwo"));

    PageData data = pageOne.getData();
    WikiPageProperties properties = data.getProperties();
    WikiPageProperty symLinks = properties.set(SymbolicPage.PROPERTY_NAME);
    symLinks.set("SymPage", "PageTwo");
    pageOne.commit(data);

    request.setResource("root");
    request.addInput("type", "pages");
    Responder responder = new SerializedPageResponder();
    SimpleResponse response =
        (SimpleResponse) responder.makeResponse(new FitNesseContext(root), request);
    String xml = response.getContent();

    assertEquals("text/xml", response.getContentType());
    assertSubString("<name>PageOne</name>", xml);
    assertSubString("<name>PageTwo</name>", xml);
    assertSubString("<name>ChildOne</name>", xml);
    assertNotSubString("SymPage", xml);
}
```



```
public void testGetPageHierarchyAsXml() throws Exception {
    makePages("PageOne", "PageOne.ChildOne", "PageTwo");

    submitRequest("root", "type:pages");

    assertResponseIsXML();
    assertResponseContains(
        "<name>PageOne</name>", "<name>PageTwo</name>", "<name>ChildOne</name>");
}

public void testSymbolicLinksAreNotInXmlPageHierarchy() throws Exception {
    WikiPage page = makePage("PageOne");
    makePages("PageOne.ChildOne", "PageTwo");

    addLinkTo(page, "PageTwo", "SymPage");

    submitRequest("root", "type:pages");

    assertResponseIsXML();
    assertResponseContains(
        "<name>PageOne</name>", "<name>PageTwo</name>", "<name>ChildOne</name>");
    assertResponseDoesNotContain("SymPage");
}
```

유의사항 2. 하나의 Test Code에는 한가지 의미만.

여러가지 테스트 코드들을
하나의 테스트 함수에 넣지 말자.

```
public void testAddMonths() {  
    SerialDate d1 = SerialDate.createInstance(31, 5, 2004);  
  
    SerialDate d2 = SerialDate.addMonths(1, d1);  
    assertEquals(30, d2.getDayOfMonth());  
    assertEquals(6, d2.getMonth());  
    assertEquals(2004, d2.getYYYY());  
  
    SerialDate d3 = SerialDate.addMonths(2, d1);  
    assertEquals(31, d3.getDayOfMonth());  
    assertEquals(7, d3.getMonth());  
    assertEquals(2004, d3.getYYYY());  
  
    SerialDate d4 = SerialDate.addMonths(1, SerialDate.addMonths(1, d1));  
    assertEquals(30, d4.getDayOfMonth());  
    assertEquals(7, d4.getMonth());  
    assertEquals(2004, d4.getYYYY());  
}
```

위 코드는 3개의 테스트를 한 함수에 몰아둔 코드이다.
위 코드 보다 3개의 테스트 함수를 만드는 것이 더 좋다.

[도전] AbsoluteSum

- ✓getAbsoluteSum(ArrayList<Integer>) 제작
 - 수를 저장한 동적배열을 넣으면,
절대값으로 변경된 동적 배열을 리턴하는 메서드
 - 양수, 음수, Zero, Mix 일 때 테스트 코드 만들기

가독성 있는 Test Code 예시 1

- ✓아래 Test Code는 AAA 구조로 잘 작성했지만,
조금 더 assert문을 간결하게 작성할 수 도 있다.

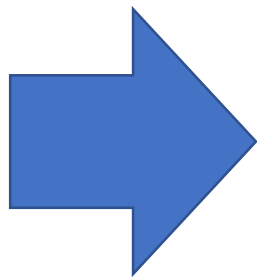
```
@Test
public void turnOnLoTempAlarmAtThreashold() throws Exception {
    hw.setTemp(WAY_TOO_COLD);
    controller.tic();
    assertTrue(hw.heaterState());
    assertTrue(hw.blowerState());
    assertFalse(hw.coolerState());
    assertFalse(hw.hiTempAlarm());
    assertTrue(hw.loTempAlarm());
}
```


가독성 있는 Test Code 예시 2

✓ 조금 더 가독성 있게 표현하기

- 왼쪽 / 오른쪽 모두 가독성이 좋은 편이지만, 가독성 있게 표현을 하고자 하는 노력의 예시이다.

```
@Test
public void turnOnLoTempAlarmAtThreashold() throws Exception {
    hw.setTemp(WAY_TOO_COLD);
    controller.tic();
    assertTrue(hw.heaterState());
    assertTrue(hw.blowerState());
    assertFalse(hw.coolerState());
    assertFalse(hw.hiTempAlarm());
    assertTrue(hw.loTempAlarm());
}
```



```
@Test
public void turnOnLoTempAlarmAtThreshold() throws Exception {
    wayTooCold();
    assertEquals("HBchL", hw.getState());
}
```

가독성 있는 Test Code 예시 3

✓ Test 함수가 많아질 경우
가독성이 좋다는 것이 잘 느껴진다.

```
@Test
public void turnOnCoolerAndBlowerIfTooHot() throws Exception {
    tooHot();
    assertEquals("hBCh1", hw.getState());
}

@Test
public void turnOnHeaterAndBlowerIfTooCold() throws Exception {
    tooCold();
    assertEquals("HBch1", hw.getState());
}

@Test
public void turnOnHiTempAlarmAtThreshold() throws Exception {
    wayTooHot();
    assertEquals("hBCh1", hw.getState());
}

@Test
public void turnOnLoTempAlarmAtThreshold() throws Exception {
    wayTooCold();
    assertEquals("HBchL", hw.getState());
}
```

깨끗한 테스트는 FIRST를 따른다. - 1

Fast

- 테스트가 느리면 자주 돌리지 못한다. 자주 돌리기 위해 빨라야한다.
- 코드를 마음껏 리팩토링 할 수 있게 빨라야한다.

Independent

- 각 테스트는 의존하면 안된다. 어떤 테스트가 먼저 수행될지 모른다.
- Fail 발생시 테스트 순서가 안맞아 Fail이 발생되었는지 의심이 없어야한다.

Repeatable

- 어떤 시간이던지, 네트워크가 안되는 등, 어떤 환경이라던지 항상 반복 가능해야한다.
- Fail 발생시, 테스트 시간에 테스트를 안돌려서 등 의심이 없어야한다.

깨끗한 테스트는 FIRST를 따른다. - 2

Self-Validating

- 테스트는 Bool 값으로 결과가 나와야한다.
- Pass 이지만, 불량이 발생할 확률이 있습니다. 가 아니라, Pass 이다.

Timely

- 테스트는 적시에 작성해야한다.
- (TDD 개발이라면) 모듈을 개발하기 전 작성한다.