



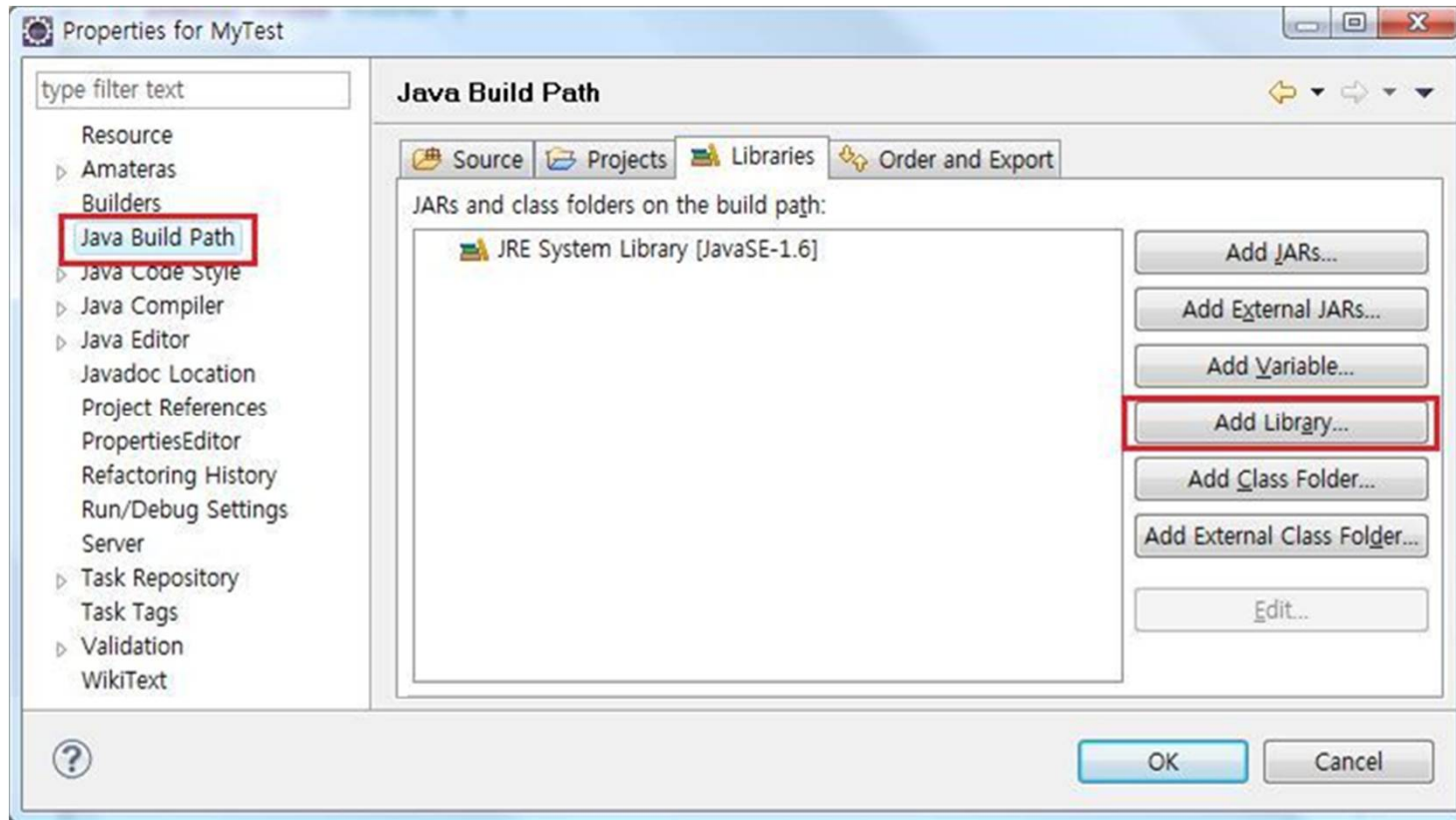
# JUnit

- JUnit이란?
  - 단위 테스트 라이브러리
- JUnit 프로젝트에 추가
  1. jar 파일로 추가.
  2. build.gradle에 추가
- JUnit Test Case 클래스 만들기
- JUnit 메서드 실행하기
- assert 메서드
  - assertEquals() + assertNotEquals()
  - assertNull() + assertNotNull()
  - assertTrue() + assertFalse()
  - Exceptions Test
  - assertSame() and assertNotSame()
  - assertEquals()
- JUnit annotations



## 프로젝트에 JUnit 추가-jar 파일로.

- Java Build Path에서 JUnit을 추가하는 방법. 비추천





## 프로젝트에 JUnit 추가-build.gradle

- build.gradle에 JUnit 추가하는 방법. 추천

// 의존성 설정

**dependencies {**

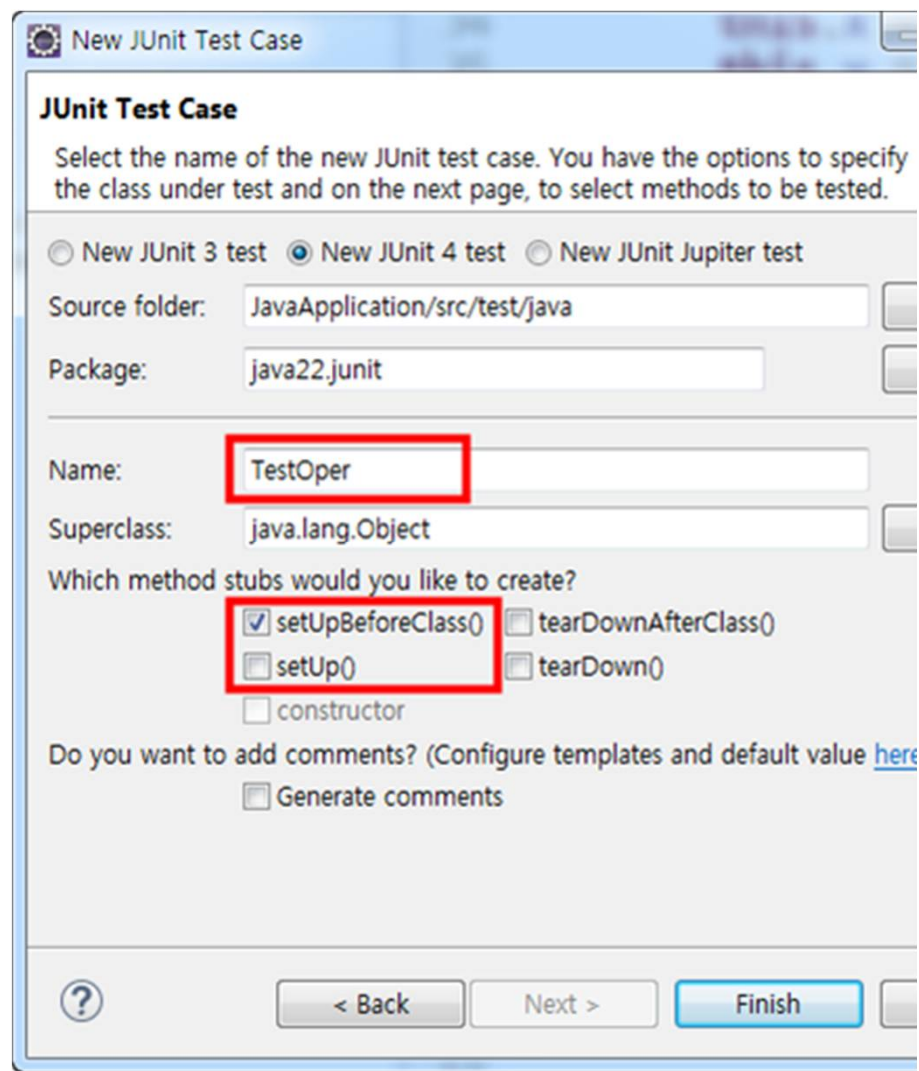
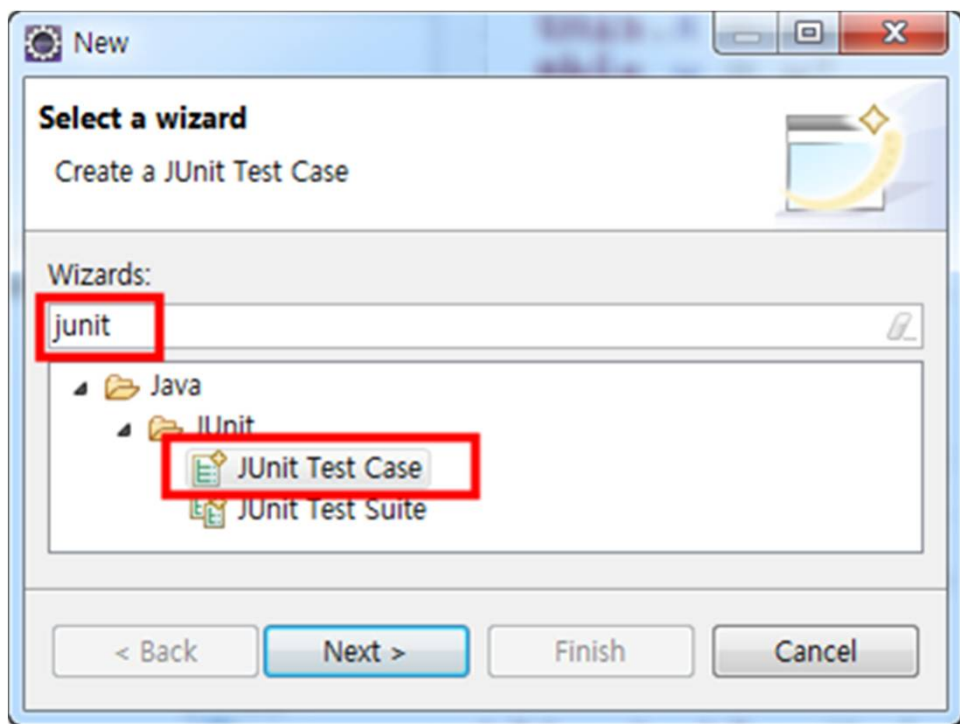
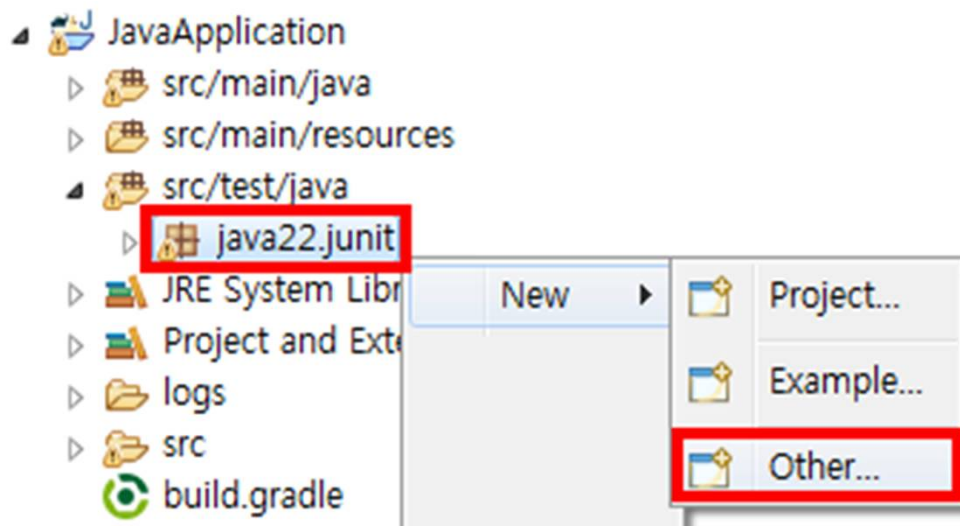
    // Use JUnit test framework

    testImplementation 'junit:junit:4.12'

**}**



# JUnit Test Case 클래스 만들기





## assert 메서드

<b>assertEquals(a,b);</b>	객체 A와 B의 값이 일치함을 확인한다. * 객체 A와 B의 equals() 메서드를 사용하여 비교
<b>assertNotEquals(a,b);</b>	
<b>assertArrayEquals(a, b);</b>	배열 A와 B가 일치함을 확인한다.
<b>assertSame(a,b);</b>	객체 A와 B가 같은 객체임을 확인한다. * 객체 A와 B의 주소값을 사용하여 비교
<b>assertNotSame(a,b);</b>	
<b>assertTrue(a);</b>	조건 A가 참인가를 확인한다.
<b>assertFalse(a);</b>	조건 A가 거짓인가를 확인한다.
<b>assertNull(a);</b>	객체 A가 null을 확인한다.
<b>assertNotNull(a);</b>	객체 A가 null이 아님을 확인한다.



## 실습: Oper 클래스 단위 테스트 코드 작성

두 정수에 대한 덧셈, 뺄셈, 곱셈, 나눗셈을 연산하는 Oper 클래스를 작성하시오

- Oper 클래스 안에 Add(), Minus(), Mul(), Div() 메서드를 만드시오.
- 나눗셈의 결과는 실수가 되도록 한다.
- JUnit을 이용하여 Oper 클래스의 메서드에 대한 단위 테스트를 작성하여 본다.

패키지명: java22.junit, 클래스명: Oper , TestOper

```
public class TestOper {  
    @Test  
    public void test_add() {}  
  
    @Test  
    public void test_minus() {}  
  
    @Test  
    public void test_mul() {}  
  
    @Test  
    public void test_div() {}  
}
```

assertEquals()  
assertNotEquals()  
  
assertTrue()  
assertFalse()  
  
assertNull()  
assertNotNull()



## assert\*() 메서드

```
import org.junit.Test;
import static org.junit.Assert.*;
```

```
public class MyUnitTest {
```

**@Test**

```
    public void testAdd() {
```

```
        // Oper 클래스의 인스턴스 생성
        Oper op = new Oper(2, 4);
        int rs = op.add();
```

```
        assertEquals(6, rs);
        assertEquals(8, rs);
```

```
        assertTrue(6 == rs);
        assertFalse(8 == rs);
```

```
    }
```

```
}
```

assertEquals()  
assertNotEquals()

assertTrue()  
assertFalse()

assertNull()  
assertNotNull()



# JUnit 메서드 실행하기

- 메서드 단위로 테스트 실행하기

@Test

```
public void test01()  
    assertNotNull()  
}
```

Debug As	▶	1 Debug on Server
Profile As	▶	JU 2 JUnit Test

- 클래스 단위로 테스트 실행하기

```
public class TestOper
```

Debug As	▶	1 Debug on Server
Profile As	▶	JU 2 JUnit Test

- 패키지 단위로 테스트 실행하기

```
JavaApplication  
├── src/main/java  
├── src/main/resources  
└── src/test/java  
    └── java22.junit
```

Debug As	▶	1 Debug on Server
Profile As	▶	JU 2 JUnit Test

- 프로젝트 단위로 테스트 실행하기

```
JavaApplication  
├── src/main/jav
```

Debug As	▶	1 Debug on Server
Profile As	▶	JU 2 JUnit Test





## 실습: Oper 클래스 단위 테스트 코드 작성

직사각형의 둘레와 면적을 구하는 Rect 클래스를 작성하고 JUnit을 이용하여 Rect 클래스의 메서드를 테스트하는 TestRect 클래스를 작성하여 보자.

### 1. Rect 클래스 작성

직사각형의 가로와 세로를 각각 **width** 와 **heigh** 라고 하고 **width** 값과 **heigh** 값은 **setter**나 **생성자**를 사용하여 설정하도록 한다.

- 필드 : width (가로) , height (세로)
- area (면적) 구하는 메서드 : `width*height`;
- perimeter (둘레) 구하는 메서드 : `2*(width+height)`;
- **src/main/java**/java22/junit/**Rect.java**

### 2. TestRect 클래스 작성

JUnit을 사용하여 `area()` 와 `perimeter()` 메서드를 테스트 하는 클래스를 작성하여 보자.

- **src/test/java**/java22/junit/**TestRect.java**



## 실습: Oper 클래스 단위 테스트 코드 작성

```
public class TestRect {
    private static Rect r = null;

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        // Rect 클래스의 인스턴스를 만들고
        // 인스턴스의 width =2, height = 30 설정
        r = new Rect();
        r.setWidth( 2 );
        r.setHeight( 30 );
    }

    @Test
    public void test_area() {
        int a = r.area() ;
        assertEquals(60, a);
        assertEquals(80, a);

        assertTrue( 60 == a );
        assertFalse( 80 == a );
    }

    // 이어서...
```

```
@Test
public void test_perimeter() {
    int p = r.perimeter();
    assertEquals(120, p);
}

@Test
public void test_assertTrue() {
    int a = r.area() ;
    assertTrue( 60 == a ); // 녹색
}

@Test
public void test_assertFalse() {
    int a = r.area() ;
    assertFalse( 80 == a ); // 녹색.
}

@Test
public void test_type() {
    Rect o = r.type();
    assertNull( o ); // 녹색
}
}
```



## 실습: MyUnit 의 단위 테스트를 작성하시오.

```
public class MyUnit {  
    public String concate(String string, String string2) {  
        return string+string2;  
    }  
    public boolean getBoolean() {  
        return false;  
    }  
    public Object getSameObject() {  
        return null;  
    }  
    public Object getObject() {  
        return null;  
    }  
    public String[] getStringArray() {  
        return new String[]{"one", "two", "three"};  
    }  
    public double getException() throws ArithmeticException {  
        throw new ArithmeticException("Not Implemented Exception");  
    }  
    public ArrayList<String> getEmptyList(){  
        return new ArrayList<String>();  
    }  
}
```



## 실습: MyUnit 의 단위 테스트를 작성하시오.

MyUnit 클래스에 대한 단위 테스트를 작성할 수 있다.

```
public class TestMyUnit {  
    private static MyUnit myUnit = null;  
    @BeforeClass  
    public static void setUpBeforeClass() throws Exception {  
        myUnit = new MyUnit();  
    }  
    @Test  
    public void test_concat() { }  
    @Test  
    public void test_getBoolean() { }  
    @Test  
    public void test_getSameObject() { }  
    @Test  
    public void test_getObject() { }  
    @Test  
    public void test_getStringArray() { }  
    @Test(expected = ArithmeticException.class)  
    public void test_getException() { }  
    @Test(expected = IndexOutOfBoundsException.class)  
    public void test_getEmptyList() { }  
}
```

**Assertions:**

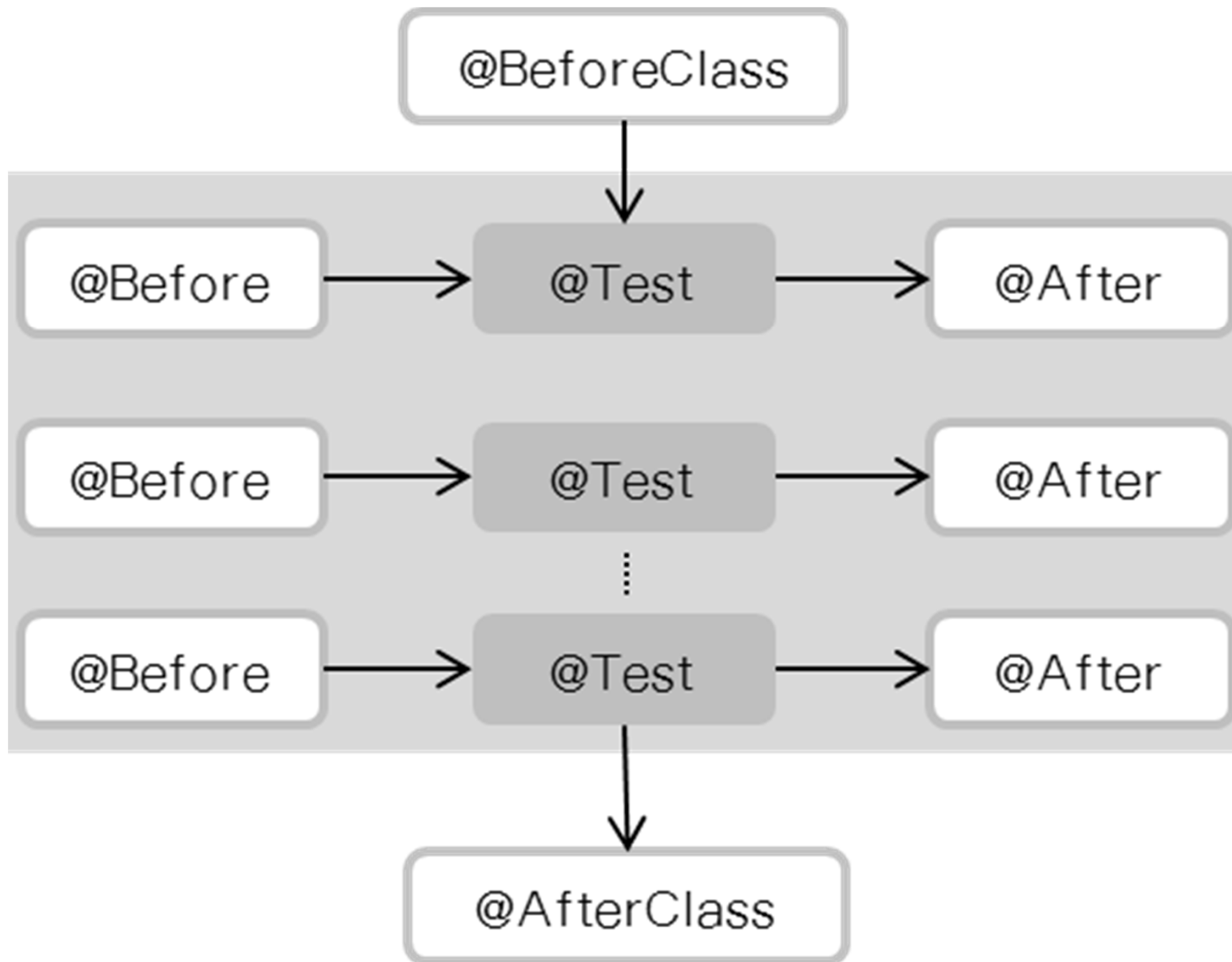
- `assertEquals()`
- `assertNotEquals()`
- `assertTrue()`
- `assertFalse()`
- `assertNull()`
- `assertNotNull()`
- `assertSame()`
- `assertNotSame()`
- `assertArrayEquals()`

**MyUnit Methods:**

- `MyUnit.concat()`
- `MyUnit.getBoolean()`
- `MyUnit.getSameObject()`
- `MyUnit.getObject()`
- `MyUnit.getStringArray()`
- `MyUnit.getException()`
- `MyUnit.getEmptyList()`



# JUnit annotations





# JUnit annotations

No.	Annotation	Description
1	@BeforeClass	Annotating a public static void method with @BeforeClass causes it to be run once before any of the test methods in the class.
2	@Before	Several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before each Test method.
3	@Test	The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.
4	@After	If you allocate external resources in a Before method, you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method.
5	@AfterClass	This will perform the method after all tests have finished. This can be used to perform clean-up activities.
6	@Ignore	The Ignore annotation is used to ignore the test and that test will not be executed.
7	@Rule	



## JUnit 실습: TestArrayList.java

```
/* 문제1. 테스트 메서드명: test01
JUnit을 이용하여 elist가 null 이 아님을
검증하는 테스트 코드들 작성하시오.
*/

/* 문제2. 테스트 메서드명: test02
JUnit을 이용하여 elist 의 갯수가 5인지를
검증하는 테스트 코드들 작성하시오.
*/

/* 문제3. 테스트 메서드명: test03
JUnit을 이용하여 elist 에 "10"이라는 값
이 존재하지 않음을 검증하는 테스트 코드들
작성하시오. */

/* 문제4. 테스트 메서드명: test04
JUnit을 이용하여 두 배열이 같음을 검증하
는 테스트 코드들 작성하시오.
String names1[] = {"y2kpooh", "hwang"};
String names2[] = {"y2kpooh", "hwang"};
*/
```

```
public class TestArrayList {

    static List<String> elist = null;

    @BeforeClass
    public static void setUpClass() {
        elist = new ArrayList<String>();
        elist.add( "0" );
        elist.add( "2" );
        elist.add( "1" );
        elist.add( "3" );
        elist.add( "4" );
    }

    /* 문제1. 테스트 메서드명: test01 */
    /* 문제2. 테스트 메서드명: test02 */
    /* 문제3. 테스트 메서드명: test03 */
    /* 문제4. 테스트 메서드명: test04 */
}
```



## JUnit 실습:

아래의 조건을 만족하는 Student 클래스를 만들고 JUnit을 이용하여 테스트 코드를 작성하시오.

### ◆ 클래스명: `java22.junit.Student`

- 필드: name, score
- score의 입력 값의 범위는 0~100까만 가능하다.
- 메서드: `getgrade()`

90점 이상이면 문자열 "A"를 리턴,  
80점 이상이면 문자열 "B"를 리턴,  
70점 이상이면 문자열 "C"를 리턴,  
60점 이상이면 문자열 "D"를 리턴,  
나머지는 문자열 "F"를 리턴

### ◆ 클래스명: `java22.junit.TestStudent`

- JUnit을 이용하여 Student 클래스의 `getgrade` 메서드를 테스트하는 메서드를 작성하시오
  - 100 이상 일 때
  - 90 이상 일 때
  - 80 이상 일 때
  - 70 이상 일 때
  - 60 이상 일 때
  - 60 미만 일 때

\* **Scanner** 사용 금지  
사람이 없이 테스트 가능하게 할 것.

\* **test\_getgrade()** 한 개만 만들고 이 안에서 테스트 코드를 작성하시오.

