- **@BeforeClass, @AfterClass, @Before, @After**

```
public class JUnitExamMain {

    private int num = 0;

    public JUnitExamMain(int num) {
        this.num = num;
    }

    public int add(int num) {
        this.num += num;
        return this.num;
    }

}
```

```
public class JUnitExamTest {
    JUnitExamMain jem;

    @BeforeClass public static void setUpBeforeClass() throws Exception {
        System.out.println("setUpBeforeClass()");
    }
    @AfterClass public static void tearDownAfterClass() throws Exception {
        System.out.println("tearDownAfterClass()");
    }
    @Before public void setUp() throws Exception {
        jem = new JUnitExamMain(10);
        System.out.println("\tsetUp()");
    }
    @After public void tearDown() throws Exception {
        System.out.println("\ttearDown()");
    }
    @Test public void testAddWithNew() {
        System.out.println("\t\ttestAddWithNew() start...");
        jem = new JUnitExamMain(5);
        assertEquals(jem.add(5), 10);
        System.out.println("\t\ttestAddWithNew() end...");
    }
    @Test public void testAddWithoutNew() {
        System.out.println("\t\ttestAddWithoutNew() start...");
        assertEquals(jem.add(5), 10);
        System.out.println("\t\ttestAddWithoutNew() end...");
    }
}
```

- **factorial : 팩토리얼. 계승(階乘)**
  - 그 수보다 작거나 같은 모든 양의 정수의 곱
  - 음이 아닌 정수 *n*의 계승 = *n*! = 1 x 2 x ... x *n*
  - 0의 계승 = 0! = 1

- **To-do**
  - 0! = 1
  - 1! = 1
  - 2! = 1 x 2
  - 3! = 1 x 2 x 3
  - n! = 1 x 2 x 3 x ... x n
  - -1! ➔ Error

- **Red**

```
public class FactorialTest {
    @Test
    public void shouldReturnOneWhenZeroIn() {
        Factorial fac = new Factorial();      ← 정의되지 않은 클래스
        assertEquals(1, fac.calc(0));         ← 정의되지 않은 메소드
    }
}
```

- **Green**

```
public class Factorial {
  public int calc(int i) {
     return 1;                                ← 테스트를 통과하는 가장 단순한 구현
  }
}
```

- **Refactoring**

  없음

- **To-do**
  - ~~0! = 1~~
  - 1! = 1
  - 2! = 1 x 2
  - 3! = 1 x 2 x 3
  - n! = 1 x 2 x 3 x ... x n
  - -1! ➔ Error

- **Red**

```
public class FactorialTest {
        :
    @Test
    public void shouldReturnOneWhenOneIn() {
        Factorial fac = new Factorial();
        assertEquals(1, fac.calc(1));
    }
}
```

- **Green**

코드 수정 없이 테스트 통과

- **Refactoring**

➔ 테스트 코드에 중복 제거

```
public class FactorialTest {
    Factorial fac;
    @Before public void setup() { fac = new Factorial(); }
    @Test public void shouldReturnOneWhenZeroIn() { assertEquals(1, fac.calc(0)); }
    @Test public void shouldReturnOneWhenOneIn () { assertEquals(1, fac.calc(1)); }
}
```

- **To-do**
  - ~~0! = 1~~
  - ~~1! = 1~~
  - 2! = 1 x 2
  - 3! = 1 x 2 x 3
  - n! = 1 x 2 x 3 x ... x n
  - -1! ➔ Error

- **Red**

```
public class FactorialTest {
        :
    @Test
    public void shouldReturnTwoWhenTwoIn() {
        assertEquals(2, fac.calc(2));
    }
}
```

- **Green**

```
public class Factorial {
    public int calc(int i) {
        if (i < 2) return 1;
        else return 2;
    }
}
```

- **Refactoring**

  없음

- **To-do**
  - ~~0! = 1~~
  - ~~1! = 1~~
  - ~~2! = 1 x 2~~
  - 3! = 1 x 2 x 3
  - n! = 1 x 2 x 3 x ... x n
  - -1! ➔ Error

Red

```
public class FactorialTest {
        :
    @Test
    public void shouldReturnSixWhenThreeIn() {
        assertEquals(6, fac.calc(3));
    }
}
```

Green

```
public class Factorial {
    public int calc(int i) {
        if (i < 2) return 1;
        else return i * calc(i-1);
    }
}
```

Refactoring

➔ 테스트 메소드의 이름을 이해하기 쉬운 이름으로 변경

```
public class FactorialTest {
        :
    @Test
    public void 팩토리얼_3은_6() {
        assertEquals(6, fac.calc(3));
    }
}
```

➔ 불필요한 구문(else) 삭제

```
public class Factorial {
    public int calc(int i) {
        if (i < 2) return 1;
        return i * calc(i-1);
    }
}
```

To-do

- ~~0! = 1~~
- ~~1! = 1~~
- ~~2! = 1 x 2~~
- ~~3! = 1 x 2 x 3~~
- n! = 1 x 2 x 3 x ... x n
- -1! ➔ Error

- **Red**

```
public class FactorialTest {
        :
    @Test
    public void 팩토리얼_10은_3628800() {
        assertEquals(3628800, fac.calc(10));
    }
}
```

- **Green**

  코드 수정 없이 테스트 통과

- **Refactoring**

  없음

- **To-do**
  - ~~0! = 1~~
  - ~~1! = 1~~
  - ~~2! = 1 x 2~~
  - ~~3! = 1 x 2 x 3~~
  - ~~n! = 1 x 2 x 3 x ... x n~~
  - -1! ➔ Error

Red

```
public class FactorialTest {
                :
    @Test(expected=IllegalArgumentException.class)
    public void 팩토리얼_음수는_예외발생() {
        fac.calc(-1);
    }}
```

Green

```
public class Factorial {
    public int calc(int i) {
        if (i < 0)
            throw new IllegalArgumentException("잘못된 입력입니다.");
                :
    }
}
```

Refactoring

➔ 테스트 함수 단순화

```
public class FactorialTest {
    @Test(expected = IllegalArgumentException.class)
    public void 팩토리얼_음수는_예외발생() {
        fac.calc(-1);
    }
    @Test
    public void 팩토리얼_0과_양수() {
        int values[][] = {{0,1},{1,1},{2,2},{3,6},{10,3628800}};
        for (int[] value : values) {
            assertEquals(value[1], fac.calc(value[0]));
        }
    }
}
```

To-do

- ~~0! = 1~~
- ~~1! = 1~~
- ~~2! = 1 x 2~~
- ~~3! = 1 x 2 x 3~~
- ~~n! = 1 x 2 x 3 x ... x n~~
- ~~-1! ➔ Error~~

String Calculator

- int add(string numbers)
- 콤마와 숫자로 구성된 문자열을 입력 받아, 콤마를 기준으로 분리한 숫자들의 합을 반환

  `add("2") = 2`

  `add("2,3") = 5`

  `add("2,3,5") = 10`
- 빈 문자열의 경우 0을 반환

  `add("") = 0`
- 구분 문자로 개행 문자(₩n)도 허용

  `add("1\2,3") = 6`
- 다양한 구분 문자(delimiter)를 "//[delimiter]₩n[numbers...]" 형식으로 허용

  `add("//;\n1;2") = 3` ➔ `";" is delimiter`
- 음수가 포함된 경우 "Negatives not allowed : [음수, ...]" 형식의 메시지를 포함한 RuntimeException을 발생
- 1000 이상인 숫자는 계산에서 제외

  `add("2,3,1001") = 5`

- **int add(string numbers)**

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

public class StringCalculatorTest {
    @Test
    public void testStringCalculator() {
        StringCalculator.add("");
    }
}
```

```
public class StringCalculator {
    public static int add(String number) {
        return 0;
    }
}
```

- **콤마와 숫자로 구성된 문자열을 입력 받아, 콤마를 기준으로 분리한 숫자들의 합을 반환**

```
public class StringCalculatorTest {
    @Test
    public void testStringCalculator() {
        assertEquals(2+3   , StringCalculator.add("2,3"));
        assertEquals(2+3+6, StringCalculator.add("2,3,6"));
    }
}
```

```
public class StringCalculator {
    public static int add(String number) {
        String[] numbers = number.split(",");
        int sum = 0;
        for(String no : numbers) {
            sum += Integer.parseInt(no);
        }
        return sum;
    }
}
```

▪ 빈 문자열의 경우 0을 반환

```
public class StringCalculatorTest {
    @Test
    public void testStringCalculator() {
        assertEquals(2+3  , StringCalculator.add("2,3"));
        assertEquals(2+3+6, StringCalculator.add("2,3,6"));
    }
    @Test
    public void test_빈문자열은_0을_반환() {
        assertEquals(0, StringCalculator.add(""));
        assertEquals(0, StringCalculator.add(" "));    // Space
        assertEquals(0, StringCalculator.add("    ")); // Tab
    }
}
```

```
public class StringCalculator {
    public static int add(String number) {
        number = number.trim();
        if ("".equals(number))
            return 0;

        String[] numbers = number.split(",");
        int sum = 0;
        for(String no : numbers) {
            sum += Integer.parseInt(no);
        }
        return sum;
    }
}
```

- **숫자 하나인 경우 해당 숫자를 반환**

```
public class StringCalculatorTest {
    @Test
    public void testStringCalculator() {
        assertEquals(0    , StringCalculator.add("0"));  // 테스트 추가
        assertEquals(2    , StringCalculator.add("2"));
        assertEquals(2+3  , StringCalculator.add("2,3"));
        assertEquals(2+3+6, StringCalculator.add("2,3,6"));
    }
    @Test
    public void test_빈문자열은_0을_반환() {
        assertEquals(0, StringCalculator.add(""));
        assertEquals(0, StringCalculator.add(" "));
        assertEquals(0, StringCalculator.add("    "));
    }
}
```

▪ 구분 문자로 개행 문자(₩n)도 허용

```
public class StringCalculatorTest {
    @Test
    public void testStringCalculator() {
        assertEquals(0    , StringCalculator.add("0"));
        assertEquals(2    , StringCalculator.add("2"));
        assertEquals(2+3  , StringCalculator.add("2,3"));
        assertEquals(2+3+6, StringCalculator.add("2,3,6"));
    }
    @Test
    public void test_빈문자열은_0을_반환() {
        assertEquals(0, StringCalculator.add(""));
        assertEquals(0, StringCalculator.add(" "));
        assertEquals(0, StringCalculator.add("    "));
    }
    @Test
    public void test_구문자로_개행문자도_허용() {
        assertEquals(2+3  , StringCalculator.add("2\n3"));
        assertEquals(2+3+6, StringCalculator.add("2\n3,6"));
    }
}
```

```
public class StringCalculator {
  public static int add(String number) {
      number = number.trim();
      if ("".equals(number))
          return 0;

      String[] numbers = number.split(",|\n");
      int sum = 0;
      for(String no : numbers) {
          sum += Integer.parseInt(no);
      }
      return sum;
  }
}
```

- **다양한 구분 문자(delimiter)를 "//[delimiter]₩n[numbers...]" 형식으로 허용**

```
public class StringCalculatorTest {
    @Test
    public void testStringCalculator() {
            :
    }
    @Test
    public void test_빈문자열은_0을_반환() {
            :
    }
    @Test
    public void test_구분자로_개행문자도_허용() {
            :
    }
    @Test
    public void test_다양한_구분문자_허용() {
        assertEquals(2+3+6, StringCalculator.add("//;\n2;3;6"));
        assertEquals(2+3+6, StringCalculator.add("//#:#\n2#:#3#:#6"));
    }
}
```

- **다양한 구분 문자(delimiter)를 "//[delimiter]₩n[numbers...]" 형식으로 허용**

```
public class StringCalculator {
    public static int add(String number) {
        number = number.trim();
        if ("".equals(number))
            return 0;

        String delimeter = ",|\n";
        String numberWithoutDelimeter = number;
        if (number.startsWith("//")) {
            delimeter = number.substring(2, number.indexOf("\n"));
            numberWithoutDelimeter = number.substring(number.indexOf("\n")+1);
        }

        String[] numbers = numberWithoutDelimeter.split(delimeter);
        int sum = 0;
        for(String no : numbers) {
            sum += Integer.parseInt(no);
        }
        return sum;
    }
}
```

- **음수가 포함된 경우 "Negatives not allowed : [음수, ...]" 형식의 메시지를 포함한 RuntimeException을 발생**

```
public class StringCalculatorTest {
            :
    @Test(expected = RuntimeException.class)
    public void test_음수가_포함된_경우_예외발생() {
        StringCalculator.add("3,-6,15,-18,46,33");
    }
    @Test
    public void test_음수가_포함된_경우_예외메시지_제공() {
        RuntimeException exception = null;
        try {
            StringCalculator.add("3,-6,15,-18,46,33");
        } catch (RuntimeException e) {
            exception = e;
        }
        assertNotNull(exception);
        assertEquals("Negatives not allowed: [-6, -18]", exception.getMessage());
    }
}
```

- **음수가 포함된 경우 "Negatives not allowed : [음수, ...]" 형식의 메시지를 포함한 RuntimeException을 발생**

```
public class StringCalculator {
    public static int add(String number) {
                    :
        List negativeNumbers = new ArrayList();
        int sum = 0;
        for(String no : numbers) {
            int i = Integer.parseInt(no);
            if (i < 0) {
                negativeNumbers.add(i);
                continue;
            }
            sum += i;
        }

        if (negativeNumbers.size() > 0) {
            System.out.println("Negatives not allowed: " + negativeNumbers.toString());
            throw new RuntimeException("Negatives not allowed: " + negativeNumbers.toString());
        }
        return sum;
    }
}
```

- **1000 이상인 숫자는 계산에서 제외**

```
public class StringCalculatorTest {
        :
    @Test
    public void test_1000이상인_숫자는_계산에서_제외() {
        assertEquals(0    , StringCalculator.add("1000,1001,1002"));
        assertEquals(2    , StringCalculator.add("2,1000,10001"));
        assertEquals(2+999, StringCalculator.add("2,999,1000"));
    }
}
```

- **1000 이상인 숫자는 계산에서 제외**

```
public class StringCalculator {
    public static int add(String number) {
                    :
        List negativeNumbers = new ArrayList();
        int sum = 0;
        for(String no : numbers) {
            int i = Integer.parseInt(no);
            if (i < 0) {
                negativeNumbers.add(i);
                continue;
            }
            if (i >= 1000) continue;
            sum += i;
        }
                    :
    }
}
```