dongguk UNIVERSITY

S/W 품질관리 및 테스팅

설계 #1: Junit

CSE4061

2023

설계 목표

- JUnit 소개
- JUnit 설치 및 사용법
- JUnit 실습
- CoffeeMaker 실습

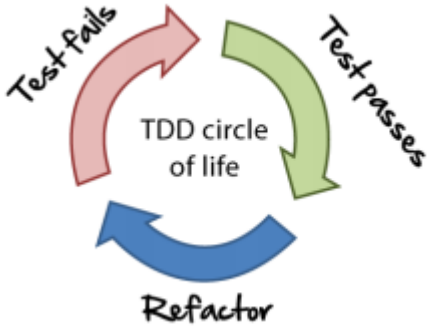
JUnit 소개

- JUnit
  - Java 언어를 위한 단위 테스트 프레임워크
  - JUnit은 테스트 주도 개발을 위한 프레임워크로 자동화된 테스트 가능
  - 오픈 소스이기 때문에 GitHub를 통해 소스 코드를 확인할 수 있음

- 테스트 주도 개발(TDD : Test-Driven Development)
  - 테스트를 먼저 한 뒤 코드를 작성하는 방법

- 단위 테스트
  - 전체 프로그램을 구성하고 있는 기본 단위 프로그램이 정상적으로 동작하는지 테스트하는 것

JUnit



JUnit 소개

- 테스트 케이스 판결
  - 판결(verdict)은 단일 테스트를 실행한 결과

- 성공(Pass): 테스트 케이스가 의도한 목적을 이루고 테스트 대상이 예상되는 결과를 수행함

- 실패(Fail): 테스트 케이스가 의도한 목적을 성취하였으나 테스트 대상이 예상된 대로 수행하지 못함

- 오류(Error): 테스트 케이스가 의도한 목적을 성취하지 못함
  - 이유:
    - 테스트 케이스 수행 중 예상하지 못한 이벤트 발생
    - 테스트 케이스가 적절히 셋업되지 않음

JUnit 소개

- **Junit 4 테스트 케이스**

```
/** Test of setName() method, of class Value  */

@Test
public void createAndSetName()
{
    Value v1 = new Value( );

    v1.setName( "Y" );

    String expected = "Y";
    String actual = v1.getName( );

    Assert.assertEquals( expected, actual );
}
```

JUnit 소개

- **Junit 4 테스트 케이스**

```
/** Test of setName() method, of class Value  */

@Test
public void
{
    Value v1 = new Value( );

    v1.setName( "Y" );

    String expected = "Y";
    String actual = v1.getName( );

    Assert.assertEquals( expected, actual );
}
```

이 Java 메소드는 테스트 케이스임을
테스트 실행자에게 표시

JUnit 소개

- **Junit 4 테스트 케이스**

```
/** Test of setName() method, of class Value  */

@Test
public void createAndSetName()
{
    Value v1 = new Value( );

    v1.setName( "Y" );

    String expected = "Y";
    String actual = v1.getName( );

    Assert.assertEquals( expected, actual );
}
```

목적:
setName이 Value 객체에 특정 이름을 저장하였는지 확인

JUnit 소개

- **Junit 4 테스트 케이스**

```
/** Test of setName() method, of class Value  */

@Test
public void createAndSetName()
{
    Value v1 = new Value( );

    v1.setName( "Y" );

    String expected = "Y"
    String actual = v1.getName( );

    Assert.assertEquals( expected, actual );
}
```

Value 객체가 정말 이름을 저장하였는지 확인하기 위하여 호출

JUnit 소개

- **Junit 4 테스트 케이스**

```
/** Test of setName() method, of class Value  */

@Test
public void createAndSetName()
{
    Value v1 = new Value( );

    v1.setName( "Y" );

    String expected = "Y";
    String actual = v1.getName( );

    Assert.assertEquals( expected, actual );
}
```

expected와
actual 는 같기를 희망

같지 않다면 테스트 케이스는
fail

JUnit 소개

- Assert(가설, 단정) 메소드
  - JUnit에서 assert는 테스트에 넣을 수 있는 정적 메소드 호출
- Assert가 참이면 메소드의 실행을 계속 진행
- Assert가 거짓이면 메소드의 실행은 그 자리에서 중지되고 테스트 케이스의 결과는 실패(fail)
- 메소드 수행 중 예외가 발생하면 테스트 케이스의 결과는 오류(error)
- 모든 메소드에서 Assert가 위배되지 않았다면 테스트 케이스는 성공(pass)

JUnit 소개

- Assert(가설, 단정) 메소드

메소드	설명
assertFalse(condition)	객체 condition이 False(거짓) 인지 확인
assertTrue(condition)	객체 condition이 True(참) 인지 확인
assertTrue(message, condition)	객체 condition이 true(참)이면 message표시
assertNull(object)	객체 object가 null인지 확인
assertNotNull(object)	객체 object가 null이 아닌지 확인

JUnit 소개

- Assert(가설, 단정) 메소드

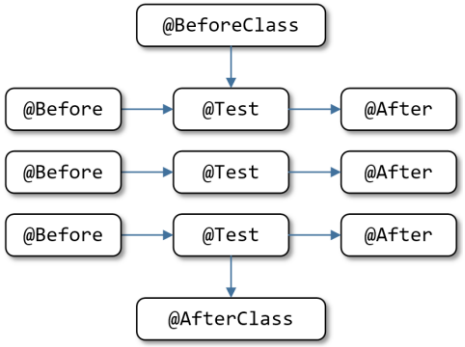
메소드	설명
assertEquals(x, y)	객체 x와 y가 일치함을 확인
assertArrayEquals(A, B)	배열 A와 B가 일치함을 확인
assertSame(Ox, Oy)	객체 Ox와 Oy가 같은 객체임을 확인
assertNotSame(Ox, Oy)	객체 Ox와 Oy가 같은 객체가 아닌지 확인
assertfail()	테스트를 바로 실패 처리

- assertEquals()메소드는 두 객체의 값이 같은지 확인하고, assertSame()메소드는 두 객체의 레퍼런스가 동일한지를 확인

JUnit 소개

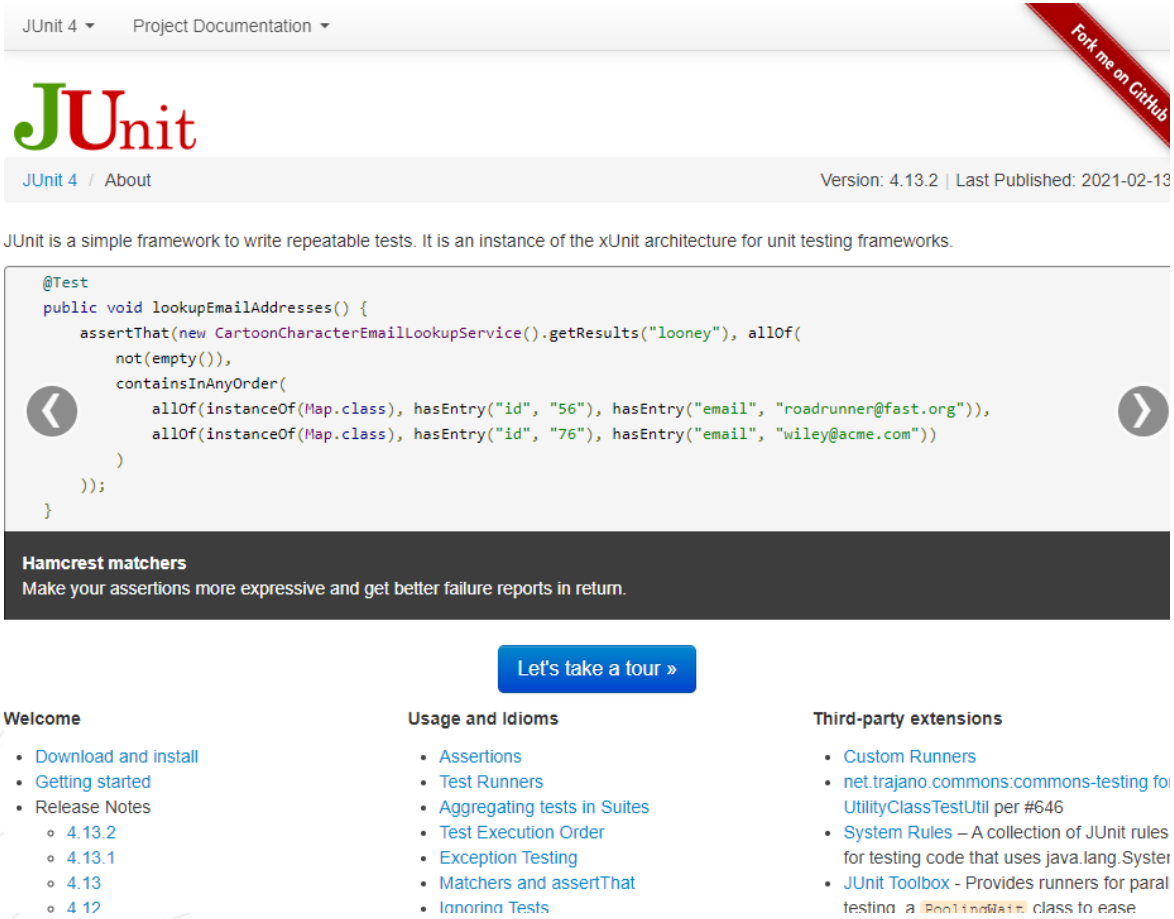
- JUnit Annotation(주석)
  - 코드 가독성과 구조를 위한 구문 메타 테이터

주석	설명
@Test	테스트를 수행하는 Annotation
@Ignore	테스트를 실행하지 않는 Annotation
@Before	@Test Annotation이 실행되기 전에 반드시 실행 @Test에서 공통으로 사용하는 코드를 @Before에 선언하여 사용
@BeforeClass	@Test Annotation보다 먼저 한번만 수행되어야 할 경우에 사용
@After	@Test Annotation이 실행된 후 실행 @Test 테스트 케이스가 실패가 되도 실행
@AfterClass	@Test Annotation보다 나중에 한번만 수행되어야 할 경우에 사용



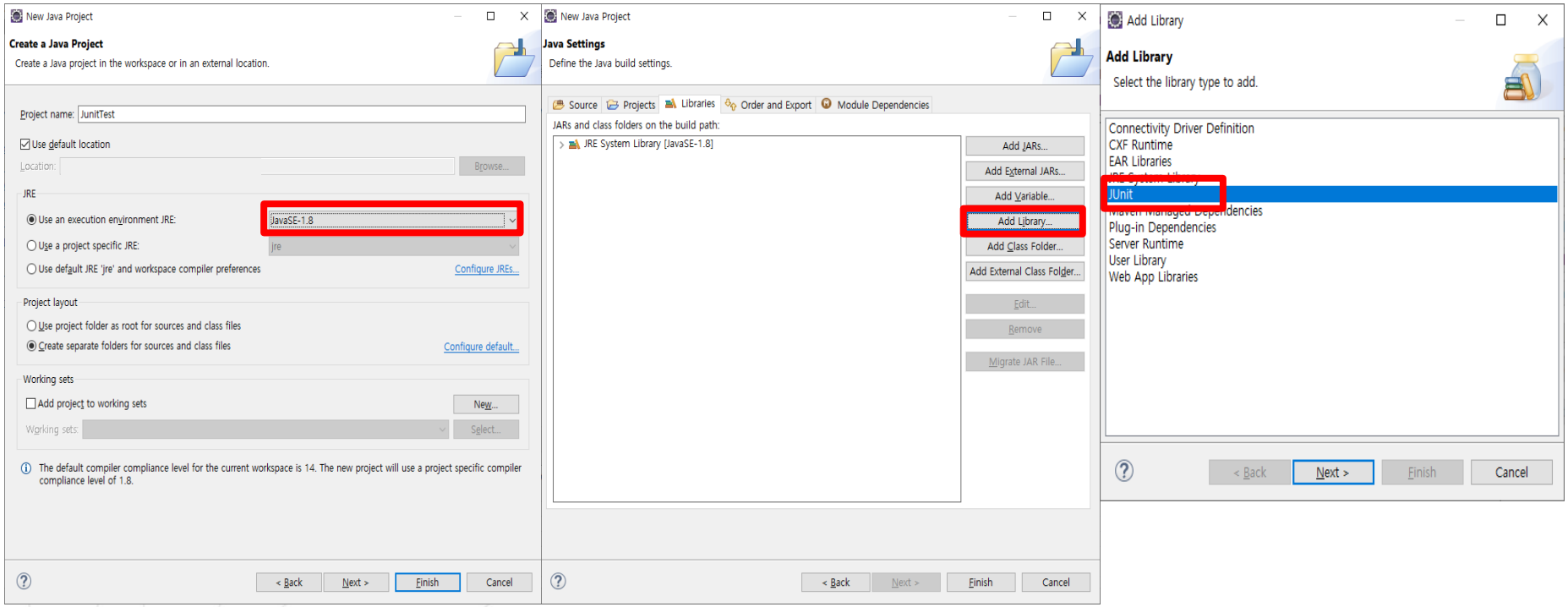
JUnit 소개

- JUnit 공식 홈페이지
  - https://junit.org/junit4/

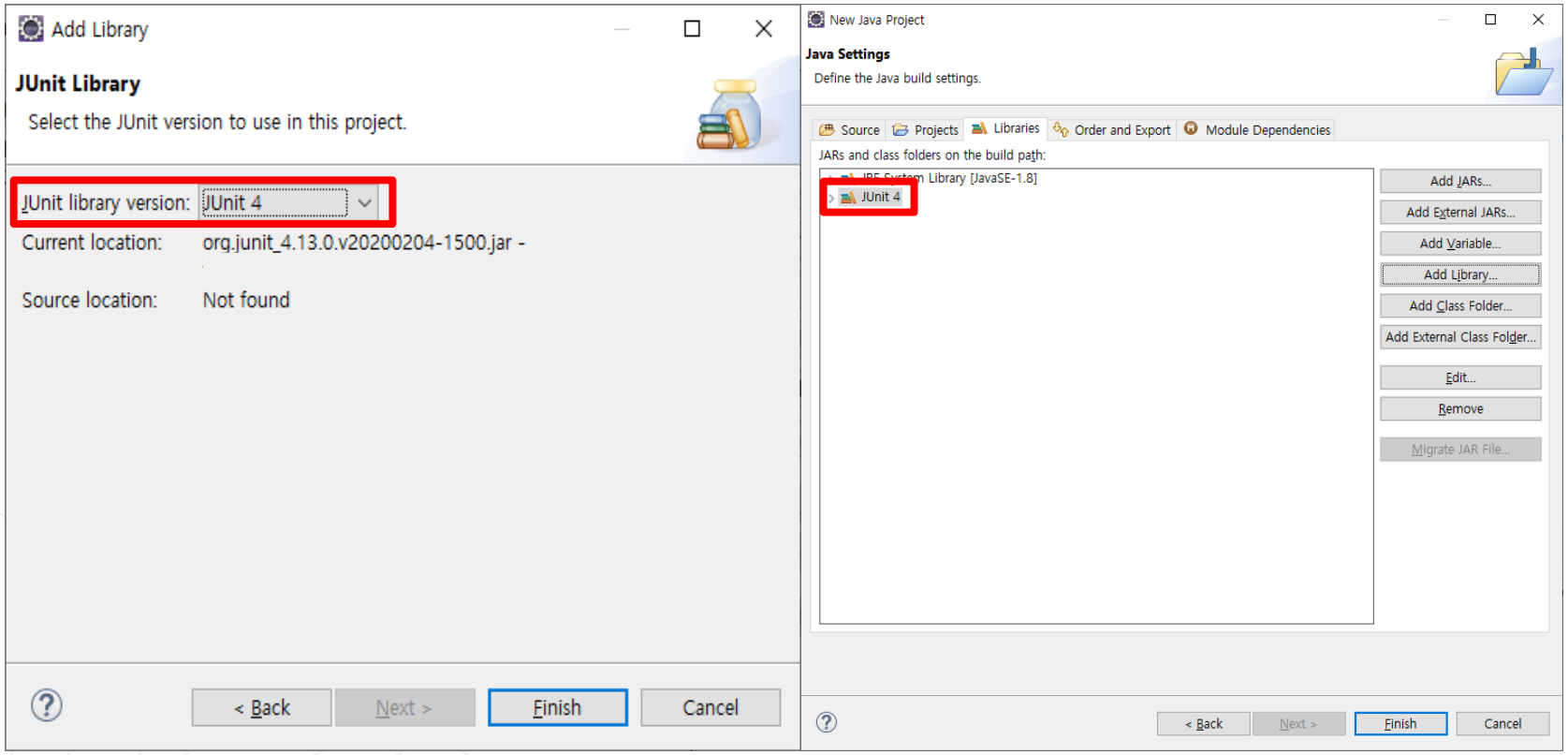


동국대학교 dongguk university

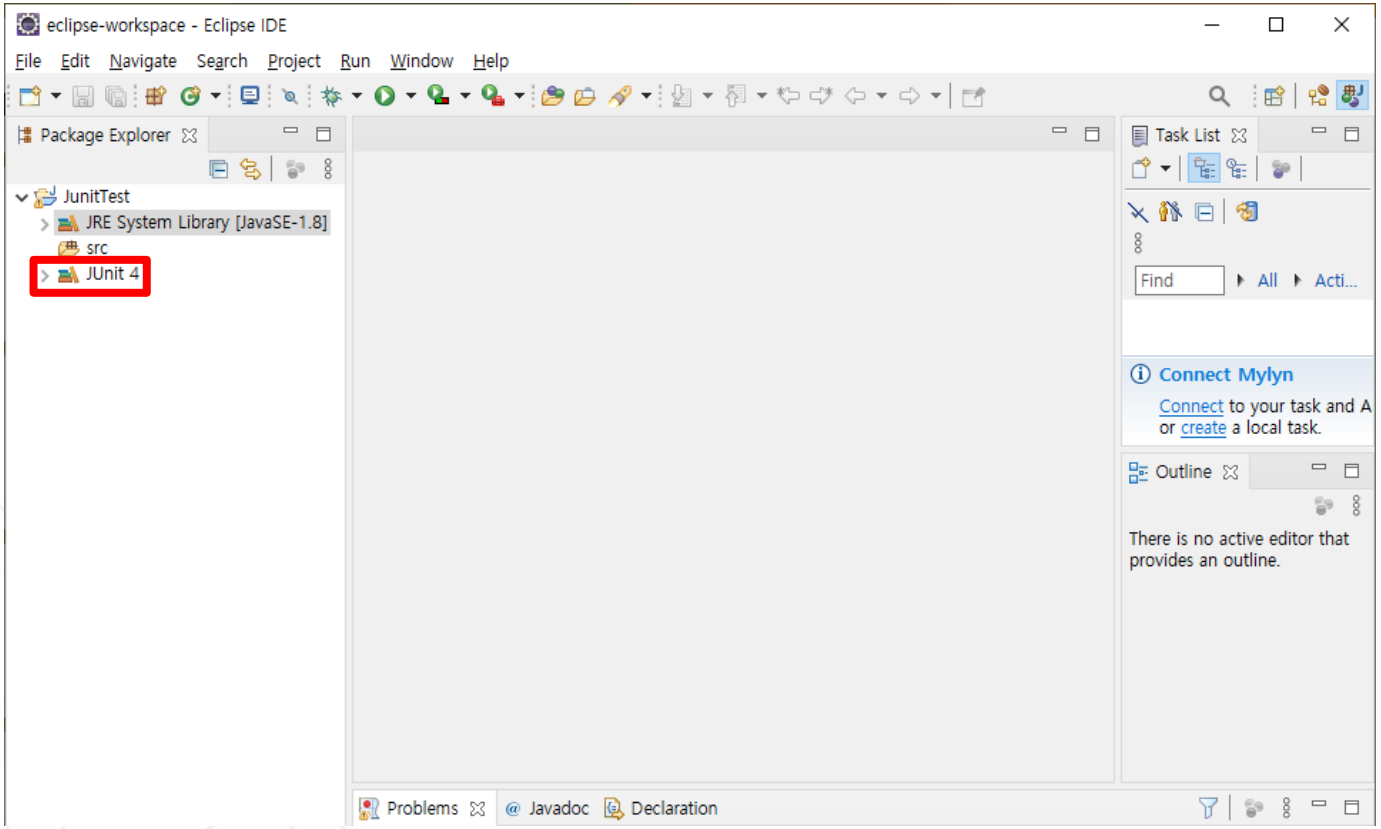
JUnit 설치 및 사용법



JUnit 설치 및 사용법



JUnit 설치 및 사용법

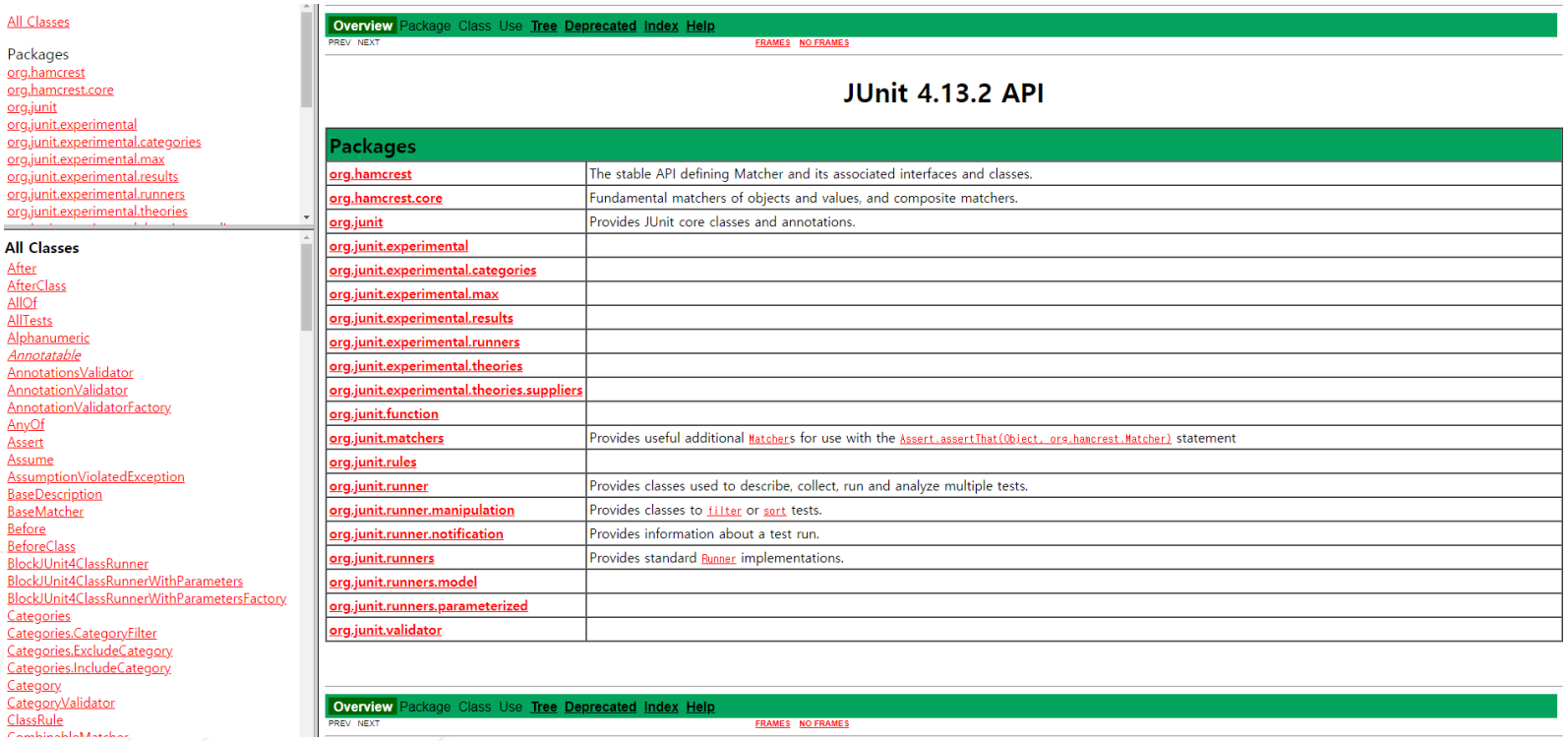


JUnit 설치 및 사용법

- Junit 기본 매뉴얼
  - @Before
    - Test 함수보다 먼저 수행되도록 정의
    - 테스트에 필요한 공통적인 내용들을 정의
  - @After
    - Test 함수가 수행된 후 실행
    - 일반적으로 정의된 함수에 할당한 자원들을 해체하는 용도로 사용
  - @BeforeClass
    - Test 함수보다 먼저 수행되며, 테스트 시 1번만 수행되도록 정의
    - 테스트의 독립성을 손상시킬 수도 있으나 공통의 내용을 뺄 수 있으므로 효율적인 방법
  - @AfterClass
    - Test 함수가 수행된 후 수행되며, 테스트 시 1번만 수행되도록 정의
    - @BeforeClass에서 예외를 발생시키더라도 @AfterClass는 반드시 수행되며, SuperClass에서 정의된 내용은 현재 클래스가 AfterClass가 수행된 이후 수행됨

JUnit 설치 및 사용법

- **Junit API - https://junit.org/junit4/javadoc/latest/index.html**



All Classes

Packages
org.hamcrest
org.hamcrest.core
org.junit
org.junit.experimental
org.junit.experimental.categories
org.junit.experimental.max
org.junit.experimental.results
org.junit.experimental.runners
org.junit.experimental.theories

All Classes
After
AfterClass
AllOf
AllTests
Alphanumeric
Annotatable
AnnotationsValidator
AnnotationValidator
AnnotationValidatorFactory
AnyOf
Assert
Assume
AssumptionViolatedException
BaseDescription
BaseMatcher
Before
BeforeClass
BlockJUnit4ClassRunner
BlockJUnit4ClassRunnerWithParameters
BlockJUnit4ClassRunnerWithParametersFactory
Categories
Categories.CategoryFilter
Categories.ExcludeCategory
Categories.IncludeCategory
Category
CategoryValidator
ClassRule

Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

JUnit 4.13.2 API

Packages	
org.hamcrest	The stable API defining Matcher and its associated interfaces and classes.
org.hamcrest.core	Fundamental matchers of objects and values, and composite matchers.
org.junit	Provides JUnit core classes and annotations.
org.junit.experimental	
org.junit.experimental.categories	
org.junit.experimental.max	
org.junit.experimental.results	
org.junit.experimental.runners	
org.junit.experimental.theories	
org.junit.experimental.theories.suppliers	
org.junit.function	
org.junit.matchers	Provides useful additional Matchers for use with the Assert.assertThat(Object, org.hamcrest.Matcher) statement
org.junit.rules	
org.junit.runner	Provides classes used to describe, collect, run and analyze multiple tests.
org.junit.runner.manipulation	Provides classes to filter or sort tests.
org.junit.runner.notification	Provides information about a test run.
org.junit.runners	Provides standard Runner implementations.
org.junit.runners.model	
org.junit.runners.parameterized	
org.junit.validator	

Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

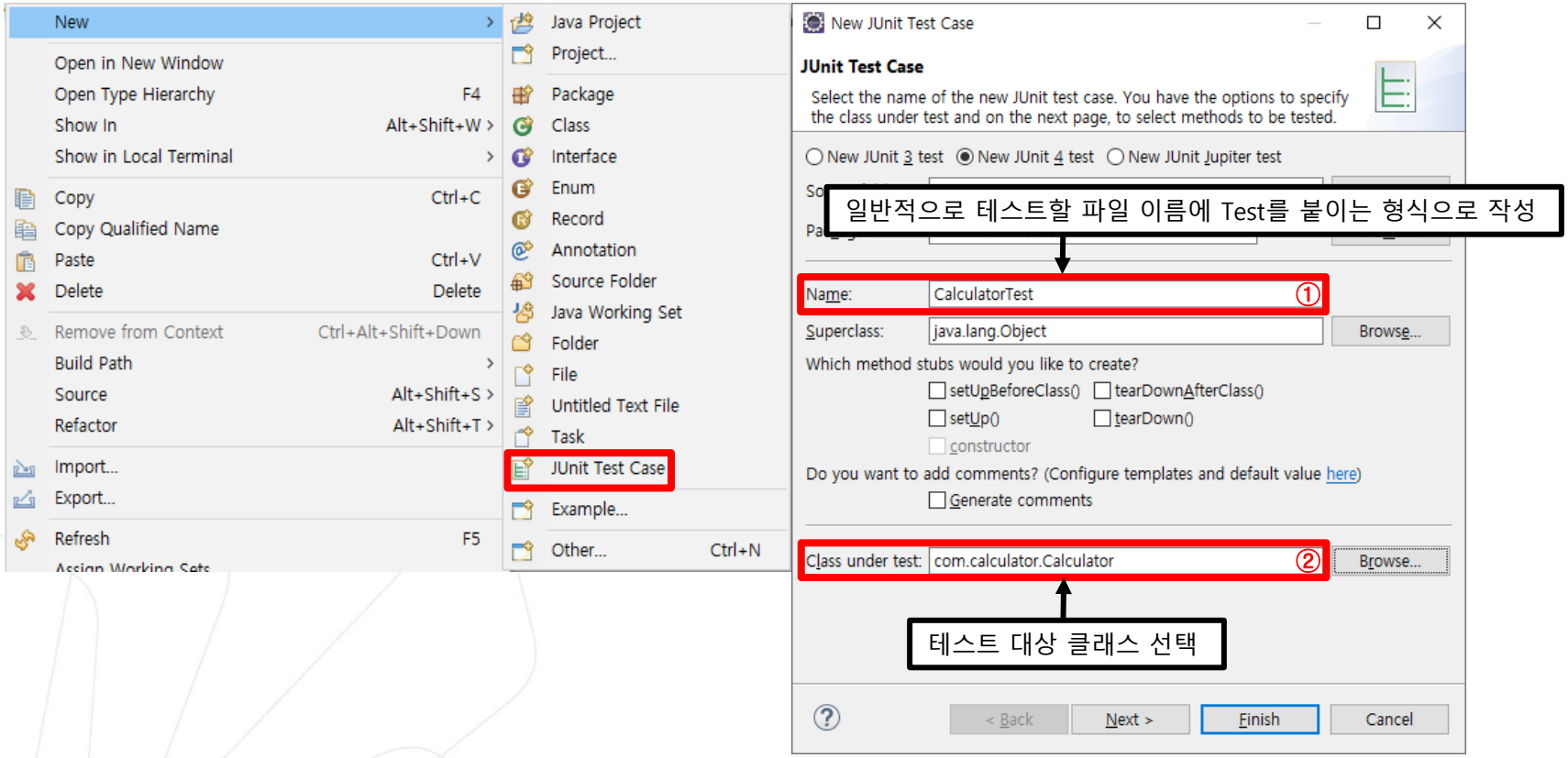
JUnit 실습

- **JUnit 실습 코드 샘플**

❖ **Calculator.java**

```
package com.calculator;

public class Calculator {
		public double add(double a, double b) {
	    return a + b;
	}

	public double subtract(double a, double b) {
	    return a - b;
	}

	public double multiply(double a, double b) {
	    return a * b;
	}

	public double divide(double a, double b) {
	    return a / b;
	}
}
```

JUnit 실습



JUnit 실습

- **JUnit 실습 코드 샘플**

  ❖ **CalculatorTest.java**

```
package com.calculator.test;
import static org.junit.Assert.*;
import com.calculator.Calculator;

public class CalculatorTest {

		@Test
		public void testAdd() {
				Calculator cal = new Calculator();
				assertEquals(15, cal.add(10, 5), 0);
		}

		@Test
		public void testSub() {
				Calculator cal = new Calculator();
				assertEquals(5, cal. subtract(10, 5), 0);
		}

		@Test
		public void testMul() {
				Calculator cal = new Calculator();
				assertEquals(50, cal.multiply(10, 5), 0);
		}

		@Test
		public void testDiv() {
				Calculator cal = new Calculator();
				assertEquals(2, cal.divide(4, 2), 0);
		}
}
```

JUnit 실습

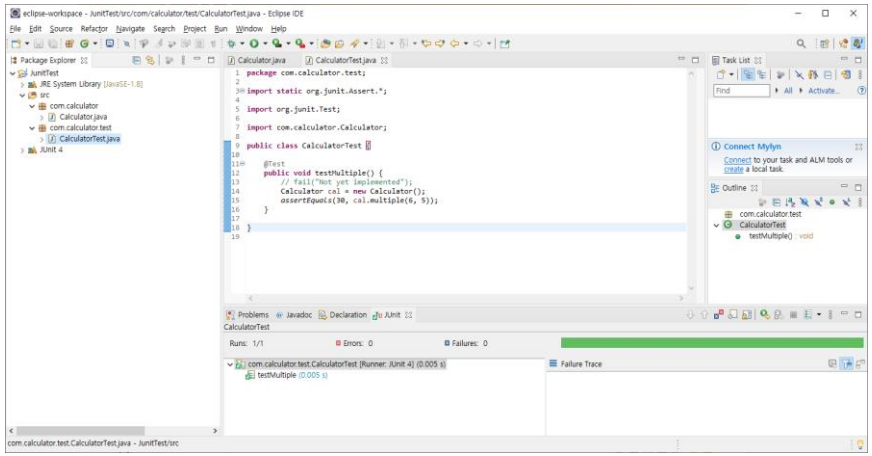
- **JUnit 실습 코드 샘플**

  ❖ **CalculatorTest.java**

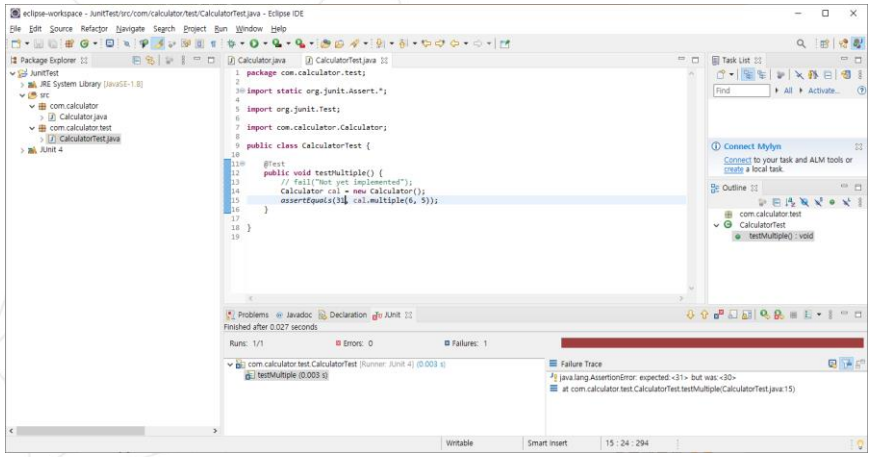
```
package com.calculator.test;
import static org.junit.Assert.*;
import com.calculator.Calculator;

public class CalculatorTest {

		@BeforeClass
		public static void beforeTest() {
				System.out.println("BeforeClass");
				}
		@Before
		public void setUp() {
				System.out.println("Before setUp");
				}

		@After
		public void tearDown() {
				System.out.println("After tearDown");
				}

		@AfterClass
		public static void afterTest() {
				System.out.println("AfterClass");
				}
}
```

JUnit 실습

- 테스트 통과 시

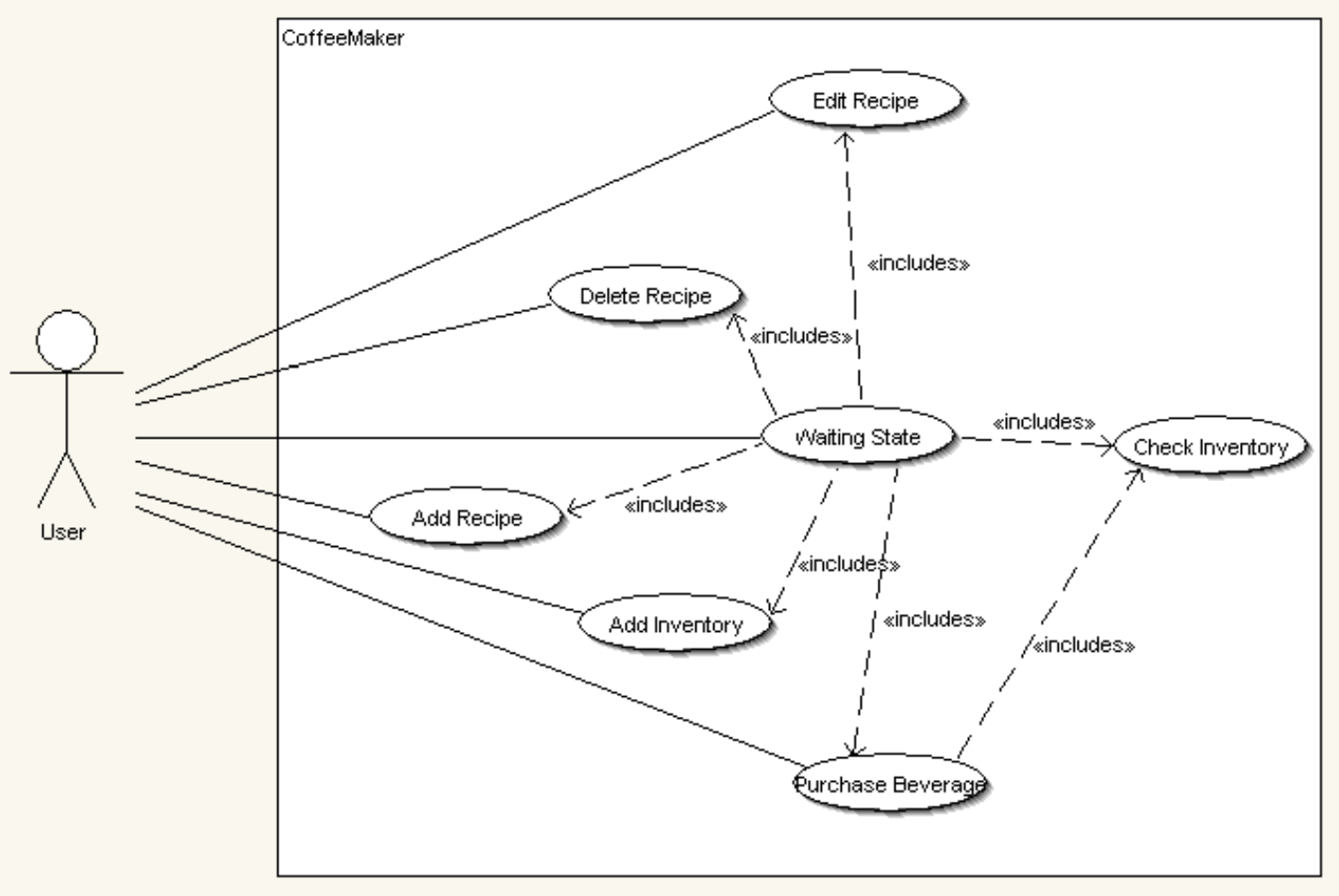


- 테스트 실패 시



CoffeeMaker 실습

- **Coffee Maker 다이어그램**



CoffeeMaker 실습

- **Coffee Maker User Story(1)**

Title: **Waiting State**		
AccTest: checkOptions0	Priority: 1	Story Points: 2
When the Coffee Maker is not in use it waits for user input. There are six different options of user input: 1) add recipe, 2) delete a recipe, 3) edit a recipe, 4) add inventory, 5) check inventory, and 6) purchase beverage.		
Title: **Add a Recipe**		
AccTest: addRecipe1	Priority: 1	Story Points: 2
Only three recipes may be added to the CoffeeMaker. A recipe consists of a name, price, units of coffee, units of milk, units of sugar, and units of chocolate. Each recipe name must be unique in the recipe list. Price must be handled as an integer. A status message is printed to specify if the recipe was successfully added or not. Upon completion, the CoffeeMaker is returned to the waiting state.		
Title: **Delete a Recipe**		
AccTest: deleteRecipe1	Priority: 2	Story Points: 1
A recipe may be deleted from the CoffeeMaker if it exists in the list of recipes in the CoffeeMaker. The recipes are listed by their name. Upon completion, a status message is printed and the Coffee Maker is returned to the waiting state.		

CoffeeMaker 실습

- **Coffee Maker User Story(2)**

Title: **Edit a Recipe**		
AccTest: editRecipe1	Priority: 2	Story Points: 1
A recipe may be edited in the CoffeeMaker if it exists in the list of recipes in the CoffeeMaker. T he reci pes are listed by their name. After selecting a recipe to edit, the user will then enter the new recipe information. A recipe name may not be changed. Upon completion, a status messag e is printed and the Coffee Maker is returned to the waiting state.		
Title: **Add Inventory**		
AccTest: addInventory 1	Priority: 1	Story Points: 2
Inventory may be added to the machine at any time from the main menu, and is added to the current inventory in the CoffeeMaker. The types of inventory in the CoffeeMaker are coffee, milk, sugar, and chocolate. The inventory is measured in integer units. Inventory may only be remove d from the Coffee Maker by purchasing a beverage. Upon completion, a status message is prin ted and the CoffeeMaker i s returned to the waiting state.		
Title: **Check Inventory**		
AccTest: checkInventory	Priority: 2	Story Points: 1
Inventory may be checked at any time from the main menu. The units of each item in the inven tory ar e displayed. Upon completion, the Coffee Maker is returned to the waiting state.		

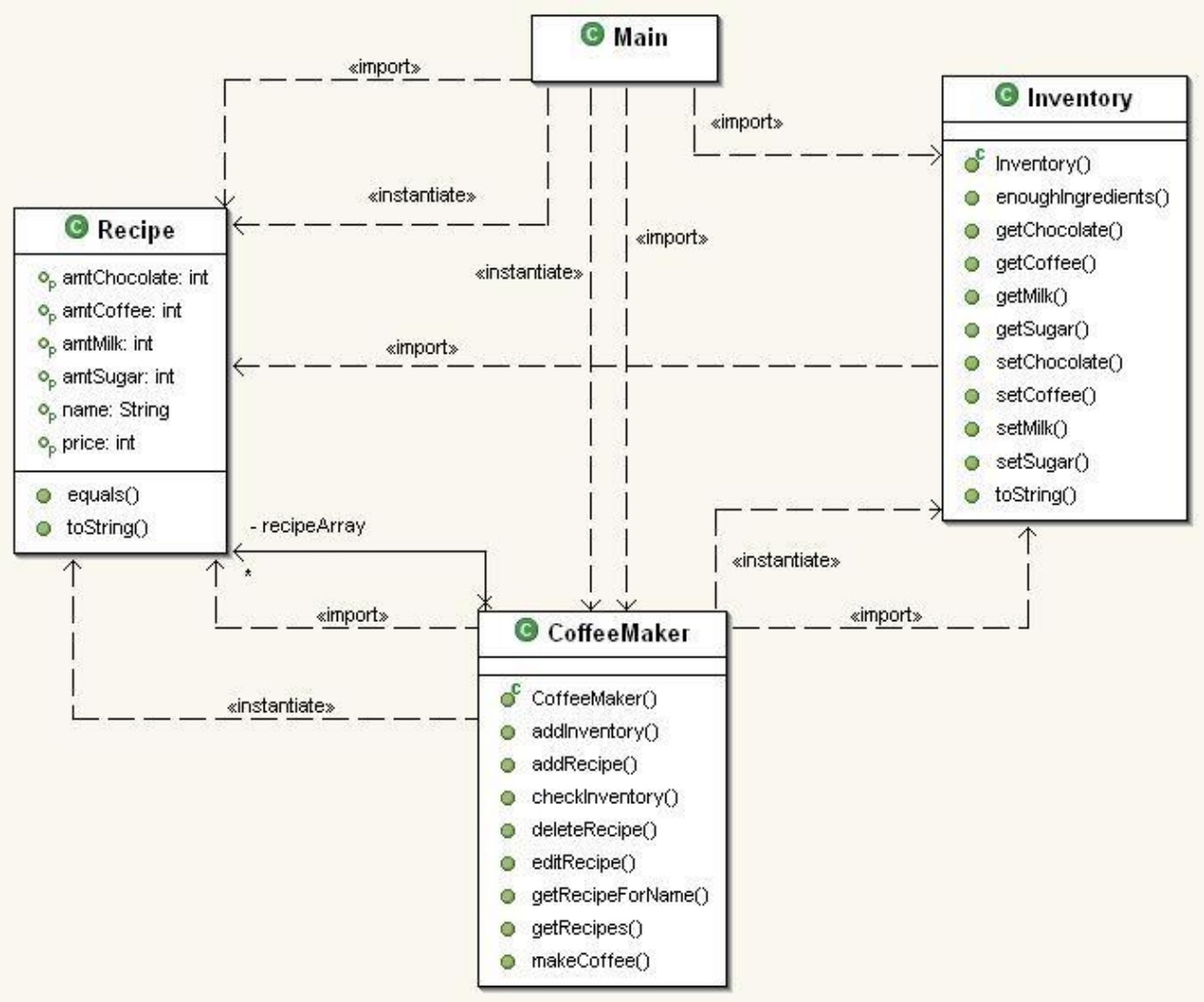
CoffeeMaker 실습

- **Coffee Maker User Story(3)**

Title: **Purchase Beverage**		
AccTest: purchaseBeverage1	Priority: 1	Story Points: 2
The user selects a beverage and inserts an amount of money. The money must be an integer. If the beverage is in the RecipeBook and the user paid enough money the beverage will be dispensed and any change will be returned. The user will not be able to purchase a beverage if they do not deposit enough money into the CoffeeMaker. A user's money will be returned if th ere is not enough inventory to make the beverage. Upon completion, the Coffee Maker displays a message about the purchase status and is returned to the main menu.		

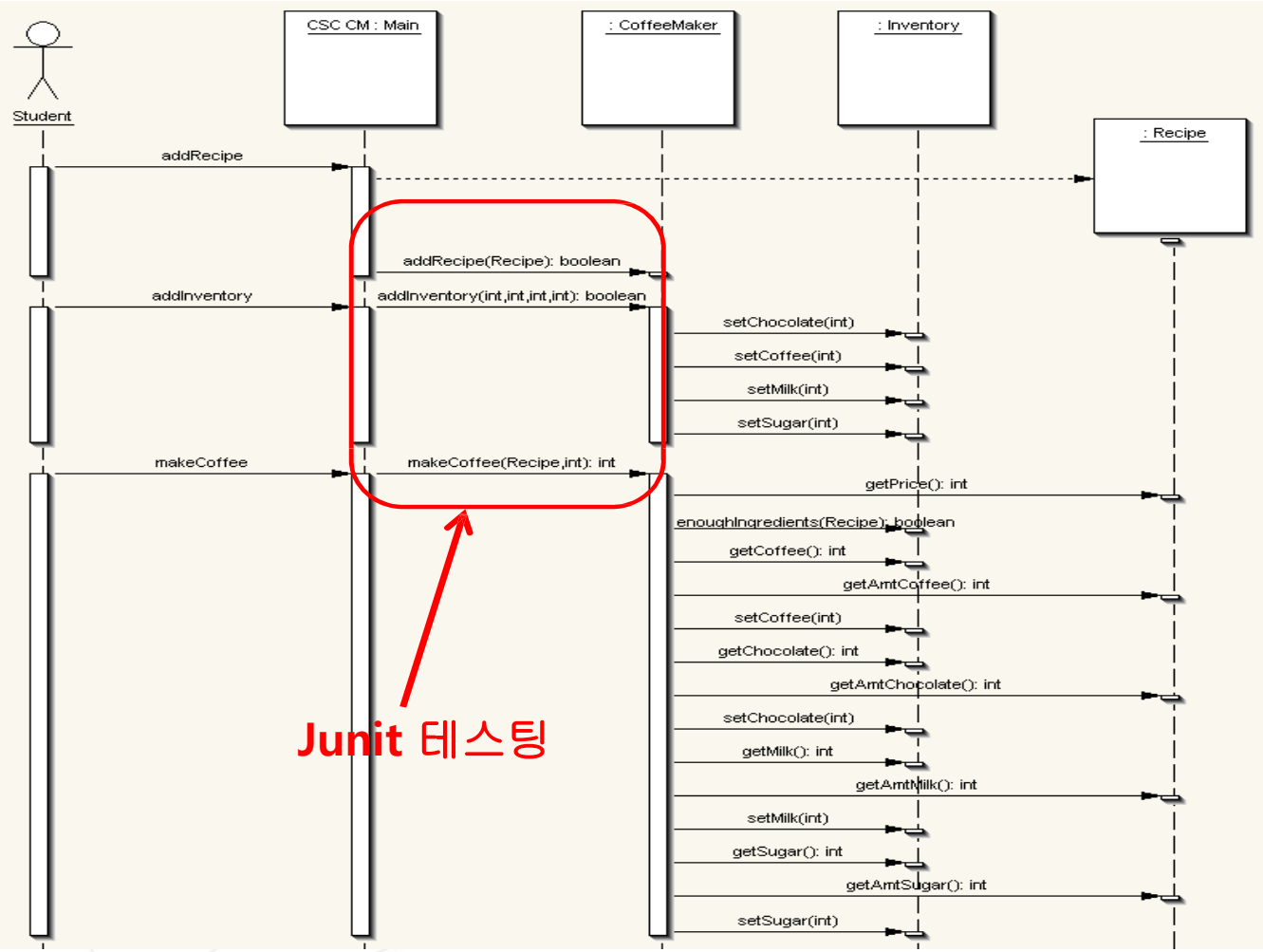
CoffeeMaker 실습

- 클래스 다이어그램



CoffeeMaker 실습

- 순서 다이어그램



CoffeeMaker 실습

```
public class CoffeeMaker {
    private static RecipeBook recipeBook;
     private static Inventory inventory;

    public CoffeeMaker() {
        recipeBook = new RecipeBook();
                inventory = new Inventory();
    }

    public boolean addRecipe(Recipe r) {
                return recipeBook.addRecipe(r);
    }

    public String deleteRecipe(int recipeToDelete) {
                return recipeBook.deleteRecipe(recipeToDelete);
    }

    public String editRecipe(int recipeToEdit, Recipe r) {
                return recipeBook.editRecipe(recipeToEdit, r);
    }
```

CoffeeMaker 실습

```
public synchronized void addInventory(String amtCoffee, String amtMilk, String amtSugar,
  String amtChocolate) throws InventoryException {
      inventory.addCoffee(amtCoffee);  inven
      tory.addMilk(amtMilk);  inventory.addS
      ugar(amtSugar);  inventory.addChocola
      te(amtChocolate);
  }
  public synchronized String checkInventory() {
      return inventory.toString();
  }
  public synchronized int makeCoffee(int recipeToPurchase, int amtPaid) {
      int change = 0;

      if (getRecipes()[recipeToPurchase] == null) {
            change = amtPaid;
      } else if (getRecipes()[recipeToPurchase].getPrice() <= amtPaid) {
            if (inventory.useIngredients(getRecipes()[recipeToPurchase])) {
                  change = amtPaid - getRecipes()[recipeToPurchase].getPrice();
            } else {
                  change = amtPaid;
            }
      } else {
            change = amtPaid;
      }

      return change;
  }
```

과제 #1: Junit을 이용한 테스트 케이스 작성

- Coffee Maker 클래스의 각 메소드를 테스트하기 위한 CoffeeMakerTest 클래스를 작성하고 테스팅하라
- Coffee Maker 와 기타 클래스의 원시코드는 e-Class의 참고자료 게시판 참고
- 제출물
  - CoffeeMakerTest 원시코드가 담긴 프로젝트 파일
  - 테스팅 실행 결과 (테스트 통과 및 실패) 캡처 파일

Sample(Capture)