

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Таныгин Максим Олегович
Должность: и.о. декана факультета фундаментальной и прикладной информатики
Дата подписания: 21.09.2023 13:09:47
Уникальный программный ключ:
65ab2aa0d384efe8480e6a4c688eddbc475e411a

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
О.А. Фоктионова
«*alt*» 12 2017 г.



**АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ПРИ КОНСТРУИРОВАНИИ
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Методические указания по выполнению лабораторной работы по
дисциплине "Конструирование программного обеспечения"
для студентов направления подготовки 09.03.04 "Программная
инженерия"

Курск 2017

УДК 004.65

Составители: В.Г. Белов, Т.М. Белова

Рецензент

Кандидат технических наук, доцент кафедры программной инженерии ЮЗГУ И.Н. Ефремова

Автоматизация тестирования при конструировании программного обеспечения: методические указания по выполнению лабораторной работы по дисциплине "Конструирование программного обеспечения" для студентов направления подготовки 09.03.04 "Программная инженерия" / Юго-Зап. гос. ун-т; сост.: В.Г. Белов, Т.М. Белова, – Курск, 2017. – 33 с.: ил. 34, табл. 2.

Изложена последовательность действий с программным обеспечением JUnit при работе в IDE Eclipse при тестирования программного обеспечения.

Материал предназначен для студентов направления подготовки бакалавров 09.03.04 «Программная инженерия», а также будет полезен студентам всех направлений подготовки, изучающим технологии тестирования пользовательского интерфейса веб-приложений.

Текст печатается в авторской редакции.

Подписано в печать *27.12.17*. Формат 60x84 1/16.
Усл. печ. л. *17*. Уч.-изд. л. *16*. Тираж 100 экз. Заказ *4391*. Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

Содержание

1	Цель лабораторной работы.....	4
2	Порядок выполнения лабораторной работы.....	5
2.1	Создание проекта в Eclipse.....	16
2.2	Подключение JUnit в Eclipse.....	23
2.3	Мастер создания тестовых наборов.....	28
3	Содержание отчета по лабораторной работе.....	32
4	Вопросы к защите лабораторной работы.....	33

1 Цель лабораторной работы

Целью лабораторной работы является приобретение знаний умений и навыков для создания в системе Eclipse проекта, подключение JUnit и осуществления модульного тестирования при работе в IDE Eclipse.

2 Порядок выполнения лабораторной работы

JUnit - это тестовая среда, в которой используются аннотации для определения методов, которые определяют тест. JUnit - проект с открытым исходным кодом, размещенный в Github. JUnit тест представляет собой метод, содержащийся в классе, который используется только для тестирования. Он называется тестовым классом. Чтобы определить, что определенный метод является тестовым методом, его требуется аннотировать @Test аннотацией. Этот метод выполняет тестируемый код. Метод assert, предоставленный JUnit или другой структурой assert используется, для проверки ожидаемого результата в сравнении с фактическим результатом. Эти вызовы методов, как правило, называют утверждениями. JUnit предполагает, что все методы тестирования могут выполняться в произвольном порядке. Хорошо написанный тестовый код не должен иметь никакого порядка, т.е. тесты не должны зависеть от других тестов. JUnit использует аннотации для обозначения методов в качестве методов тестирования и их настройки. В таблице 1 представлен обзор наиболее важных аннотаций в JUnit.

Таблица 1 - Аннотации

JUnit	Описание
<code>import org.junit.*</code>	Импорт инструкции для использования следующих аннотаций.
<code>@Test</code>	Определяет метод в качестве метода тестирования.
<code>@Before</code>	Выполняется перед каждым тестом. Он используется для подготовки тестовой среды (например, чтение входных данных, инициализация класса).

Продолжение таблицы 1

<code>@After</code>	Выполняется после каждого теста. Он используется для очистки тестовой среды (например, удаление временных данных, восстановление значений по умолчанию). Он также может экономить память, очищая дорогие структуры памяти.
<code>@BeforeClass</code>	Выполняется один раз, перед началом всех тестов. Он используется для выполнения действий, требующих много времени, например, для подключения к базе данных. Методы, отмеченные этой аннотацией, должны быть определены как <code>static</code> работа с <code>JUnit</code> .
<code>@AfterClass</code>	Выполняется один раз, после завершения всех тестов. Он используется для выполнения действий по очистке, например, для отключения от базы данных. Методы, аннотированные этой аннотацией, должны быть определены как <code>static</code> работа с <code>JUnit</code> .

Продолжение таблицы 1

@Ignore или @Ignore("Why disabled")	Отмечает, что тест должен быть отключен. Это полезно, когда базовый код был изменен, и тестовый пример еще не адаптирован. Или если время выполнения этого теста слишком велико для включения. Лучше всего предоставить необязательное описание, почему тест отключен.
@Test (expected = Exception.class)	Ошибка, если метод не выбрасывает именованное исключение.
@Test(timeout=100)	Ошибка, если метод занимает больше 100 миллисекунд.

JUnit предоставляет статические методы для проверки определенных условий через Assert класс. Эти утверждения обычно начинаются с assert. Они позволяют указать сообщение об ошибке, ожидаемый и фактический результат. Метод утверждения сравнивает фактическое значение, возвращаемое с помощью теста с ожидаемым значением. Он бросает, AssertionError если сравнение не удастся. В приведенной ниже таблице дается обзор этих методов. Параметры в скобках [] являются необязательными и имеют тип String.

Таблица 2 - Методы утверждения результатов испытаний

Утверждение	Описание
fail([message])	Пусть метод терпит неудачу. Может быть использован для проверки того, что определенная часть кода не была

Утверждение	Описание
	достигнута, или для выполнения теста на отказ до того, как будет реализован тестовый код. Параметр сообщения является необязательным.
assertTrue([message,] boolean condition)	Проверяет, что логическое условие истинно.
assertFalse([message,] boolean condition)	Проверяет, что логическое условие ложно.
assertEquals([message,] expected, actual)	Проверяет, что два значения одинаковы. Примечание: для массивов ссылка проверяется не на содержимое массивов.
assertEquals([message,] expected, actual, tolerance)	Проверьте, совпадают ли значения float или double. Допуск - это количество десятичных знаков, которые должны быть одинаковыми.
assertNull([message,] object)	Проверяет, что объект имеет значение null.
assertNotNull([message,] object)	Проверяет, что объект не равен нулю.
assertSame([message,] expected, actual)	Проверяет, что обе переменные относятся к одному и тому же объекту.
assertNotSame([message,] expected, actual)	Проверяет, что обе переменные относятся к разным объектам.

Если имеется несколько тестовых классов, то их обычно объединяют в набор тестов. Запуск тестового набора выполняет все тестовые классы в этом пакете в указанном порядке. Набор тестов также может содержать другие тестовые комплекты. Следующий пример кода на рисунке 1 демонстрирует использование набора тестов.

Он содержит два тестовых класса (MyClassTest и MySecondClassTest). Чтобы добавить еще один тестовый класс, нужно добавить его в @Suite.SuiteClasses инструкцию.

```

package com.vogella.junit.first;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({
    MyClassTest.class,
    MySecondClassTest.class })
public class AllTests {

}

```

Рисунок 1 – Использование набора тестов

Аннотация @Ignore позволяет статически игнорировать тест. В качестве альтернативы используется Assume.assumeFalse или Assume.assumeTrue определить условие для теста. Assume.assumeFalse признак теста недействительным, если его условие имеет значение true. Assume.assumeTrue оценивает тест как недействительный, если его условие оценивается как false. Например, на рисунке 2 следующее отключает проверку на Linux:

```

Assume.assumeFalse(System.getProperty("os.name").contains("Linux"));

```

Рисунок 2 – Пример аннотации Assume.assumeFalse

JUnit позволяет использовать параметры в классе тестов. Этот класс может содержать один тестовый метод, и этот метод выполняется с различными предоставленными параметрами. Отмечается тестовый класс как параметризованный тест с @RunWith(Parameterized.class) аннотацией. Такой класс тестов должен содержать статический метод, аннотированный @Parameters аннотацией. Этот метод генерирует и

возвращает набор массивов. Каждый элемент этой коллекции используется как параметр для метода тестирования. Также можно использовать `@Parameter` аннотацию в общедоступных полях, чтобы получить тестовые значения, введенные в тест. Следующий код показывает пример для параметризованного теста на рисунках 3 и 4. Он проверяет `multiply()` метод `MyClass` класса, который включен в качестве внутреннего класса для целей этого примера.

```
package testing;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import java.util.Arrays;
import java.util.Collection;
import static org.junit.Assert.assertEquals;
import static org.junit.runners.Parameterized.*;

@RunWith(Parameterized.class)
public class ParameterizedTestFields {

    // fields used together with @Parameter must be public
    @Parameter(0)
```

Рисунок 3 – Пример параметризованного теста

```

public int m1;
@Parameter(1)
public int m2;
@Parameter(2)
public int result;
// creates the test data
@Parameters
public static Collection<Object[]> data() {
    Object[][] data = new Object[][] { { 1, 2, 2 }, { 5, 3, 15 }, { 121, 4, 484 } };
    return Arrays.asList(data);
}
@Test
public void testMultiplyException() {
    MyClass tester = new MyClass();
    assertEquals("Result", result, tester.multiply(m1, m2));
}
// class to be tested
class MyClass {
    public int multiply(int i, int j) {
        return i * j;
    }
}
}

```

Рисунок 4 – Продолжение примера с параметрическим тестом

В качестве альтернативы использованию `@Parameter` аннотации используется конструктор, в котором сохраняются значения для каждого теста. Количество элементов в каждом массиве, предоставленное аннотированным методом, `@Parameters` должно соответствовать количеству параметров в конструкторе класса. Класс создается для каждого параметра, а тестовые значения передаются через конструктор в класс, показанный на рисунках 5 и 6.

```
package de.vogella.junit.first;

import static org.junit.Assert.assertEquals;

import java.util.Arrays;
import java.util.Collection;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

@RunWith(Parameterized.class)
public class ParameterizedTestUsingConstructor {

    private int m1;
    private int m2;

    public ParameterizedTestUsingConstructor(int p1, int p2) {
        m1 = p1;
        m2 = p2;
    }

    // creates the test data
    @Parameters
    public static Collection<Object[]> data() {
```

Рисунок 5 – Передача тестовых значений через конструктор

```
Object[][] data = new Object[][] { { 1, 2 }, { 5, 3 }, { 121, 4 } };
return Arrays.asList(data);
}

@Test
public void testMultiplyException() {
    MyClass tester = new MyClass();
    assertEquals("Result", m1 * m2, tester.multiply(m1, m2));
}

// class to be tested
class MyClass {
    public int multiply(int i, int j) {
        return i * j;
    }
}
}
```

Рисунок 6 – Продолжение примера передачи тестовых значений через конструктор

При запуске этого класса метод тестирования выполняется с каждым определенным параметром. В приведенном выше примере тестовый метод выполняется три раза.

Через правила JUnit можно добавлять поведение к каждому тесту в тестовом классе. Также можно аннотировать поля типа TestRule с @Rule аннотацией, вдобавок, создавать объекты, которые могут быть использованы и настроены в ваших методах тестирования. Это повышает гибкость тестов. Например, на рисунке 7 показано, как можно указать, какое сообщение об исключении ожидается во время выполнения тестового кода.

```
package de.vogella.junit.first;

import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;

public class RuleExceptionTesterExample {

    @Rule
    public ExpectedException exception = ExpectedException.none();

    @Test
    public void throwsIllegalArgumentOutOfRangeExceptionIfIconIsNull() {
        exception.expect(IllegalArgumentException.class);
        exception.expectMessage("Negative value not allowed");
        ClassToBeTested t = new ClassToBeTested();
        t.methodToBeTest(-1);
    }
}
```

Рисунок 7 – Добавление сообщения об исключении

JUnit уже предоставляет несколько полезных реализаций правил. Например, `TemporaryFolder` класс позволяет настраивать файлы и папки, которые автоматически удаляются после каждого тестового прогона. Следующий код на рисунке 8 показывает пример использования `TemporaryFolder` реализации.

```
package de.vogella.junit.first;

import static org.junit.Assert.assertTrue;

import java.io.File;
import java.io.IOException;

import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TemporaryFolder;

public class RuleTester {

    @Rule
    public TemporaryFolder folder = new TemporaryFolder();

    @Test
    public void testUsingTempFolder() throws IOException {
        File createdFolder = folder.newFolder("newfolder");
        File createdFile = folder.newFile("myfilefile.txt");
        assertTrue(createdFile.exists());
    }
}
```

Рисунок 8 - Пример использования TemporaryFolder

2.1 Создание проекта в Eclipse

Чтобы создать проект в Eclipse, выполните следующие действия:

1. Запустите среду Eclipse.
2. Выберите создание нового проекта: File/New/Java Project как показано на рисунке 9.

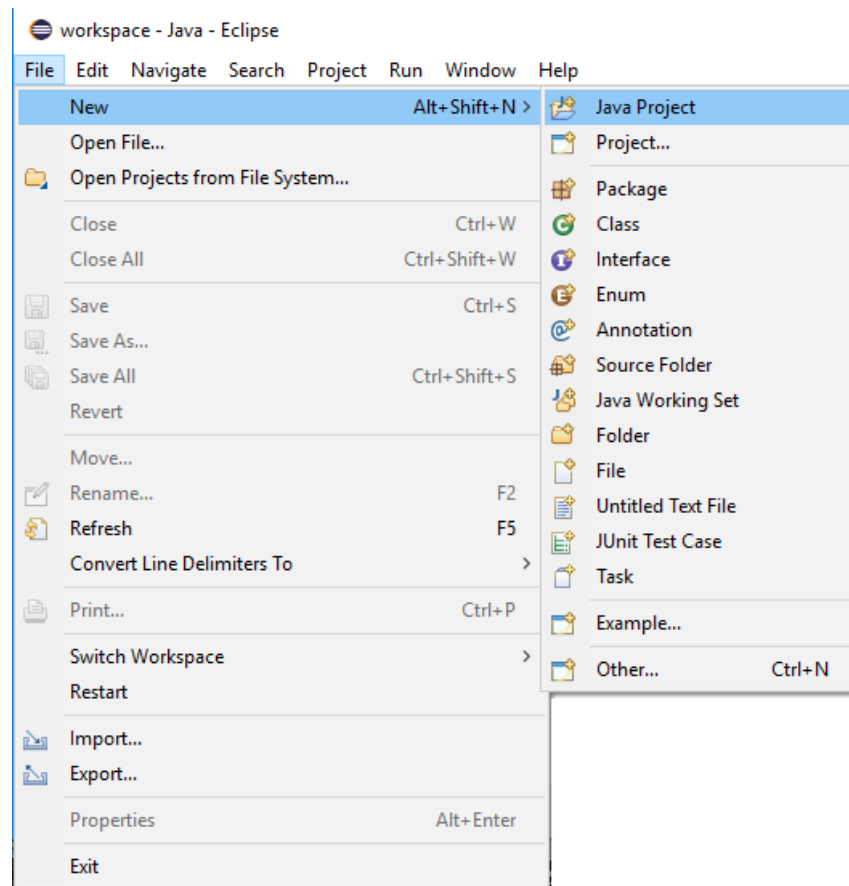


Рисунок 9 – Создание проекта

3. В открывшемся окне впишите имя проекта «JUnitTestProject». Вид окна создания проекта показано на рисунке 10.

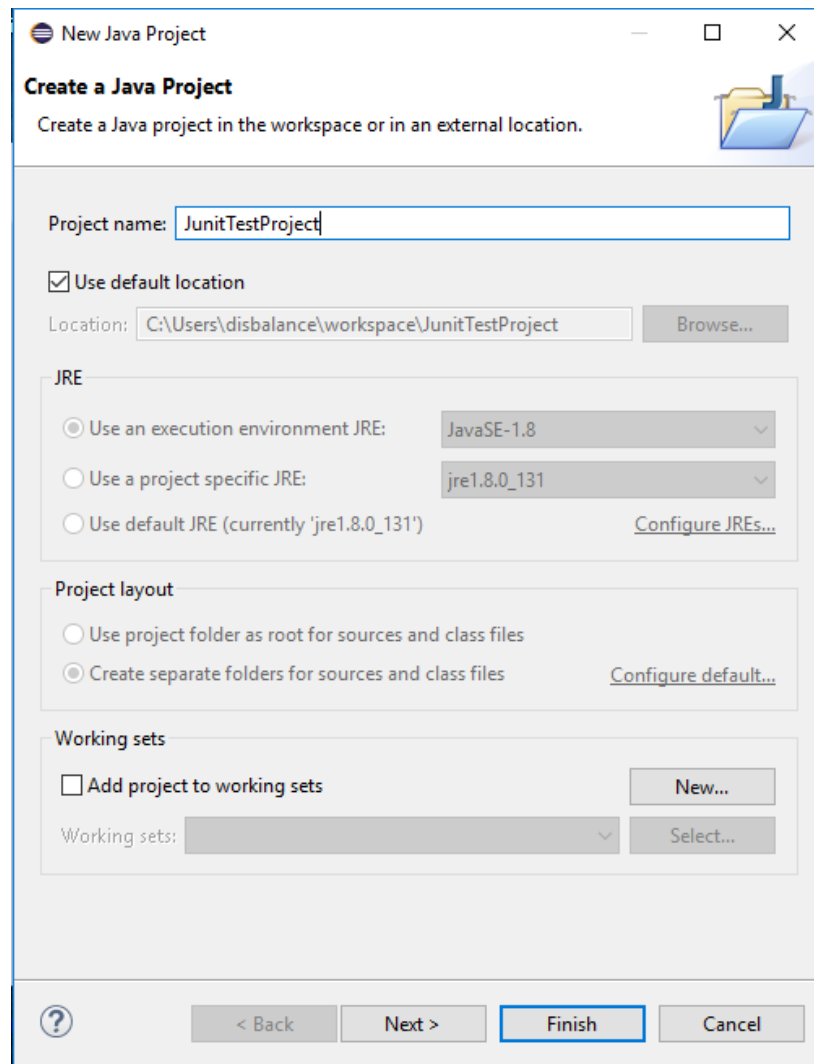


Рисунок 10 – Вид окна создания проекта

4. Нажмите Finish.
5. Создайте пакет `com.junit.sampletest`. Для этого выберите создание пакета: `File/New/Package` как показано на рисунке 11.

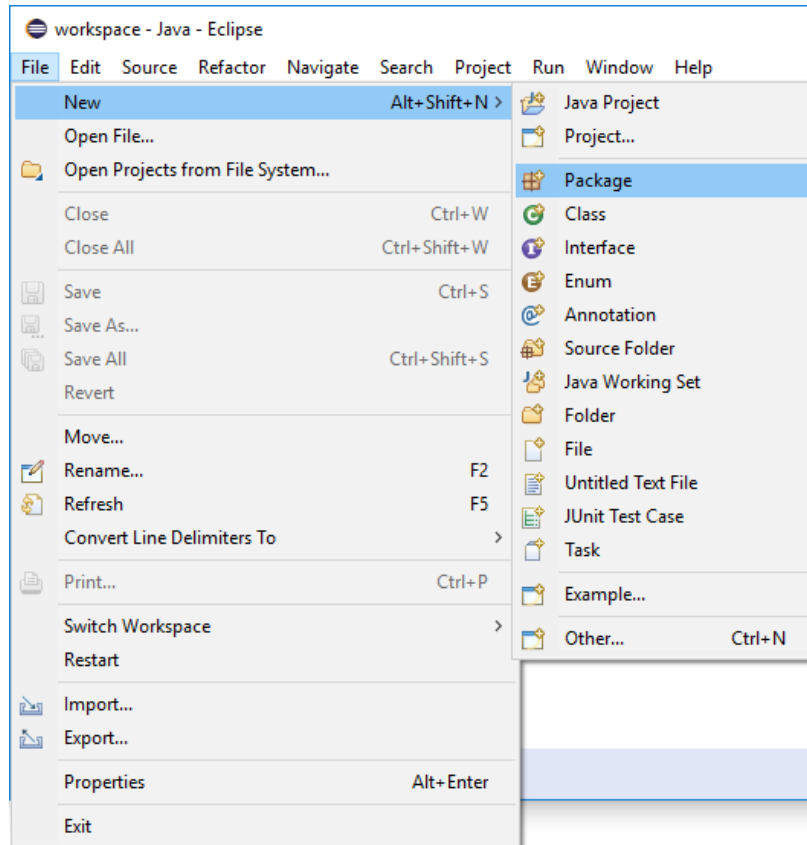


Рисунок 11 – Создание пакета

6. В открывшемся окне впишите имя «com.junit.sampletest». Вид окна создания пакета показан на рисунке 12.

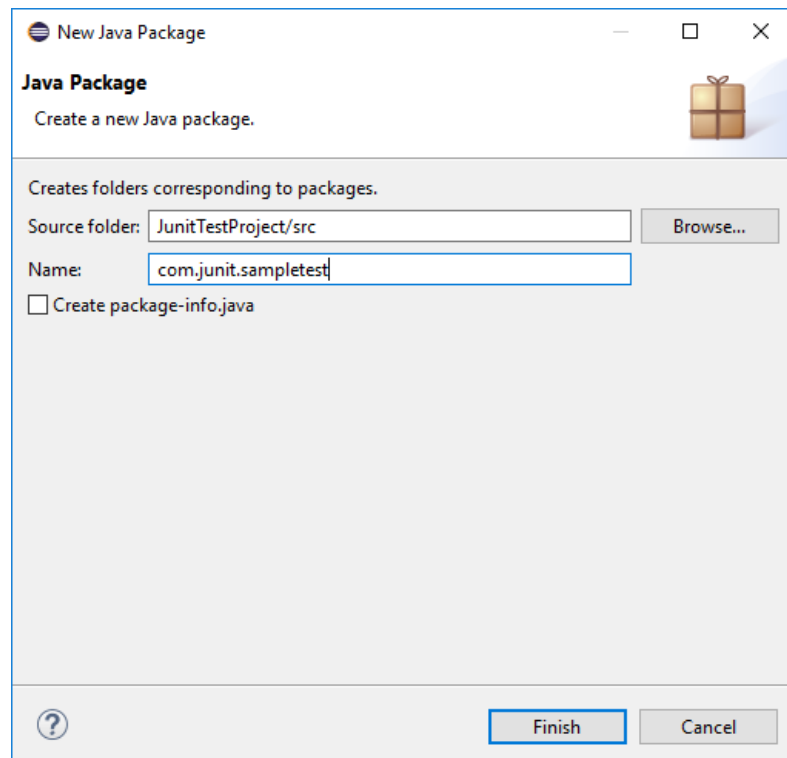


Рисунок 12 – Вид окна создания пакета

7. Нажмите Finish.
8. Вид окна Eclipse после создания пакета показан на рисунке 13.

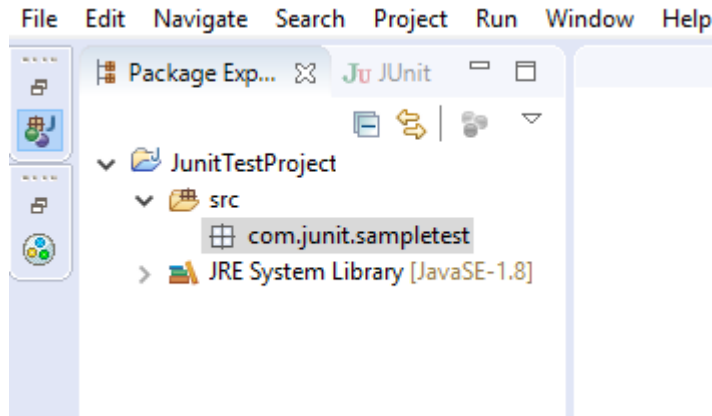


Рисунок 13 – Результат создания пакета

9. Создайте главный класс ClassToTest. Для этого кликните правой кнопкой мыши на пакете com.junit.sampletest, выберите New/Class как показано на рисунке 14.

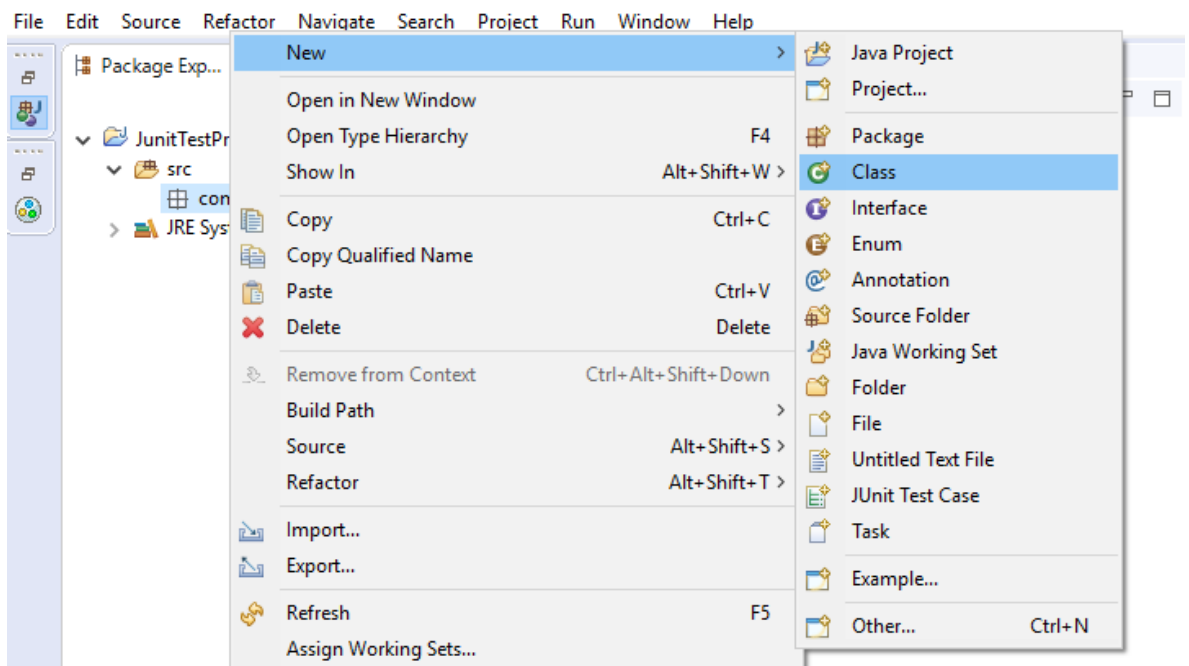


Рисунок 14 – Создание класса в пакете com.junit.sampletest

10. В открывшемся окне впишите имя класса «ClassToTest». Вид окна создания нового класса показан на рисунке 15.

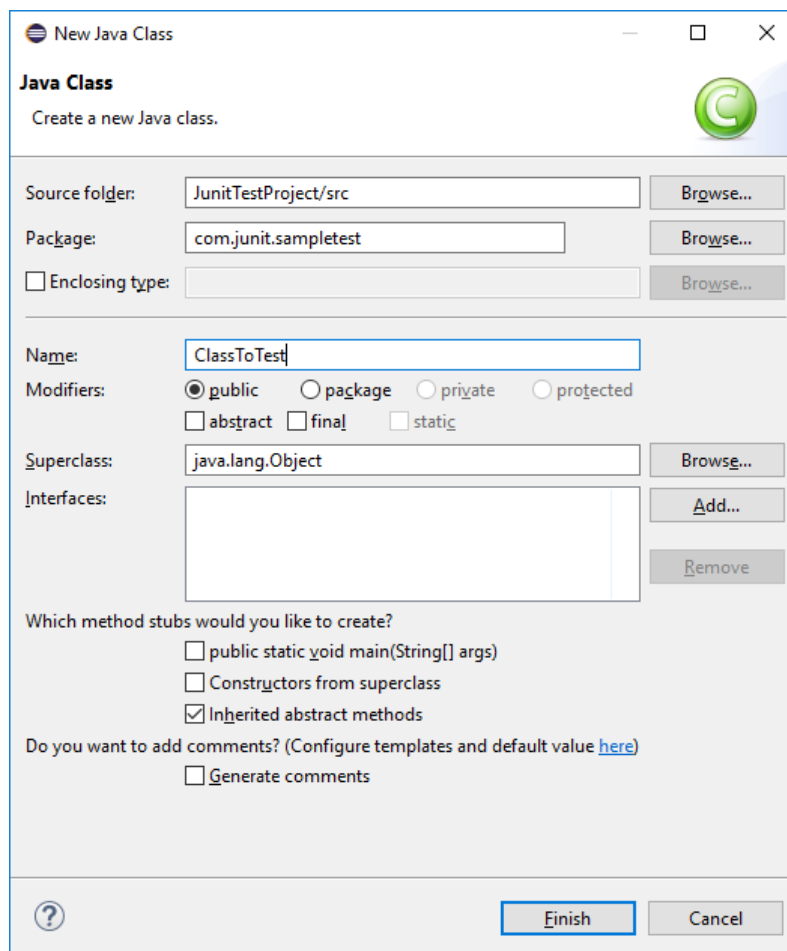


Рисунок 15 – Окно создания класса

11. Нажмите Finish.
12. Вид окна Eclipse после создания класса показано на рисунке 16.

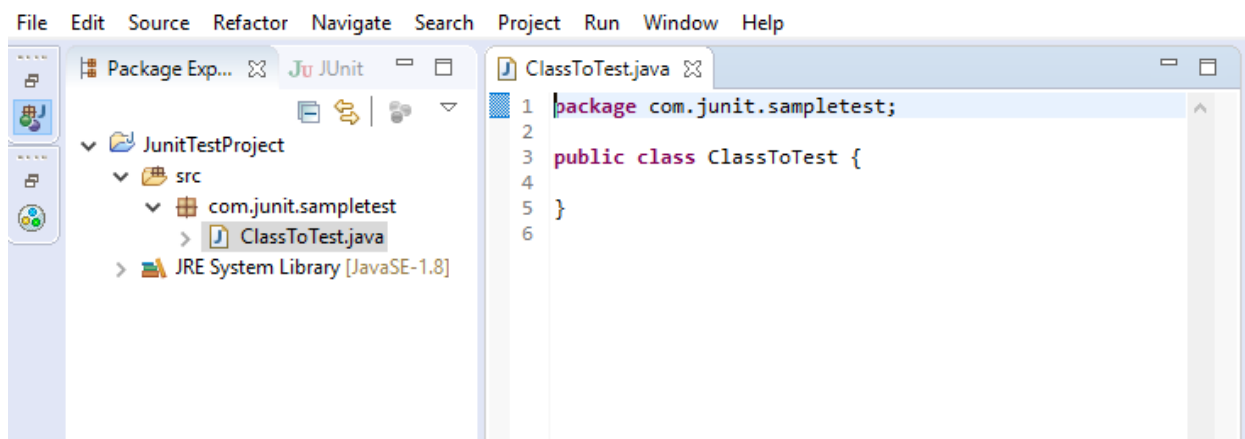


Рисунок 16 – Результат создания класса

13. Напишите простую программу в классе ClassToTest, которая показана на рисунке 17.

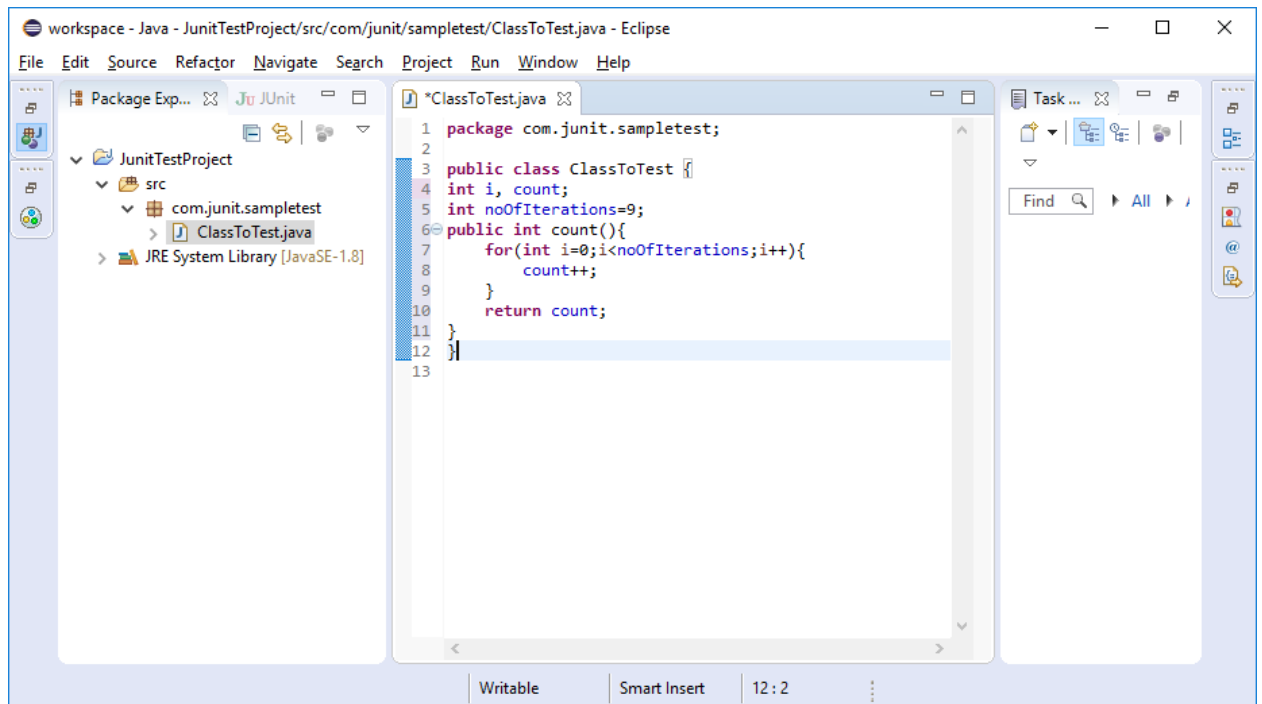


Рисунок 17 – Текст программы

14. Создайте тестовый класс TestClass. Для этого кликните правой кнопкой мыши на пакете com.junit.sampletest, выберите New/Class как на рисунке 18.

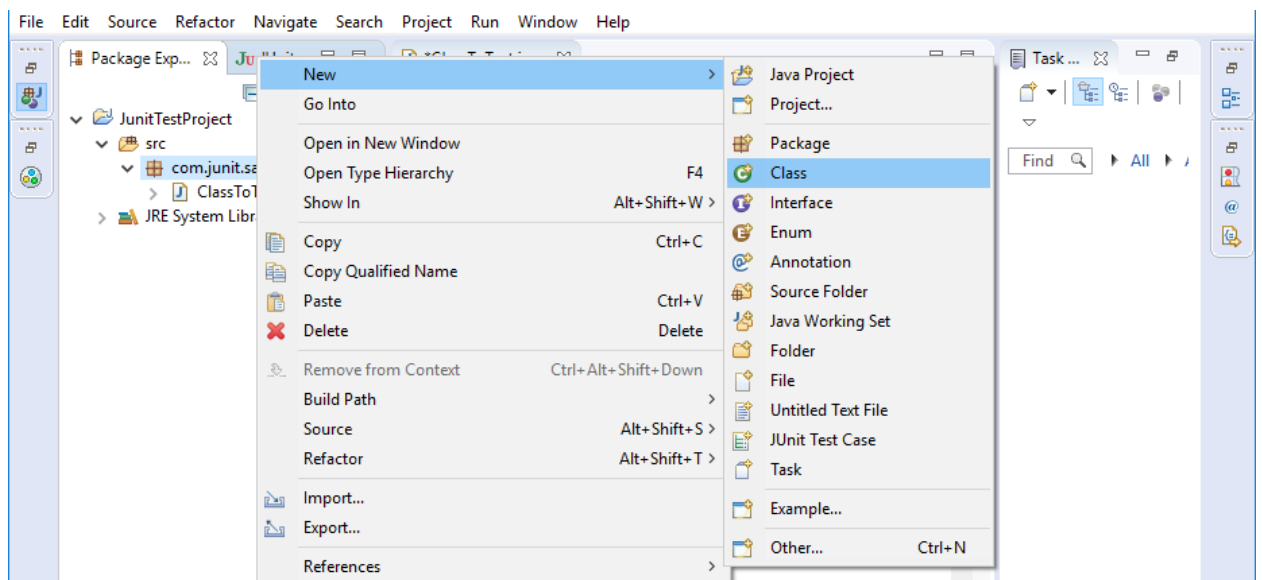


Рисунок 18 – Создание класса в пакете com.junit.sampletest

15. В открывшемся окне впишите имя класса «TestClass». Вид окна создания нового класса показано на рисунке 19.

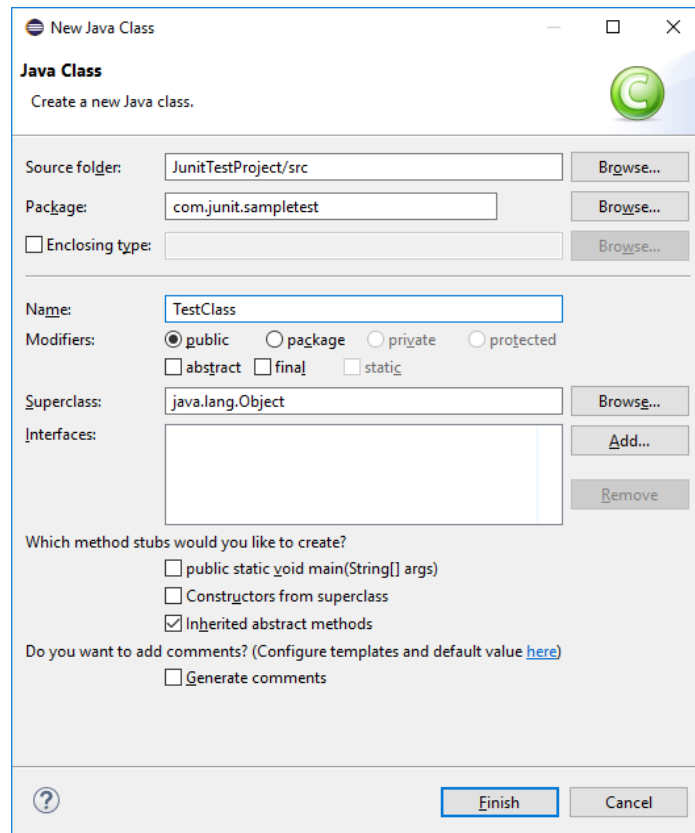


Рисунок 19 – Окно создания класса

16. Нажмите Finish.

17. Вид окна Eclipse после создания класса показано на рисунке 20.

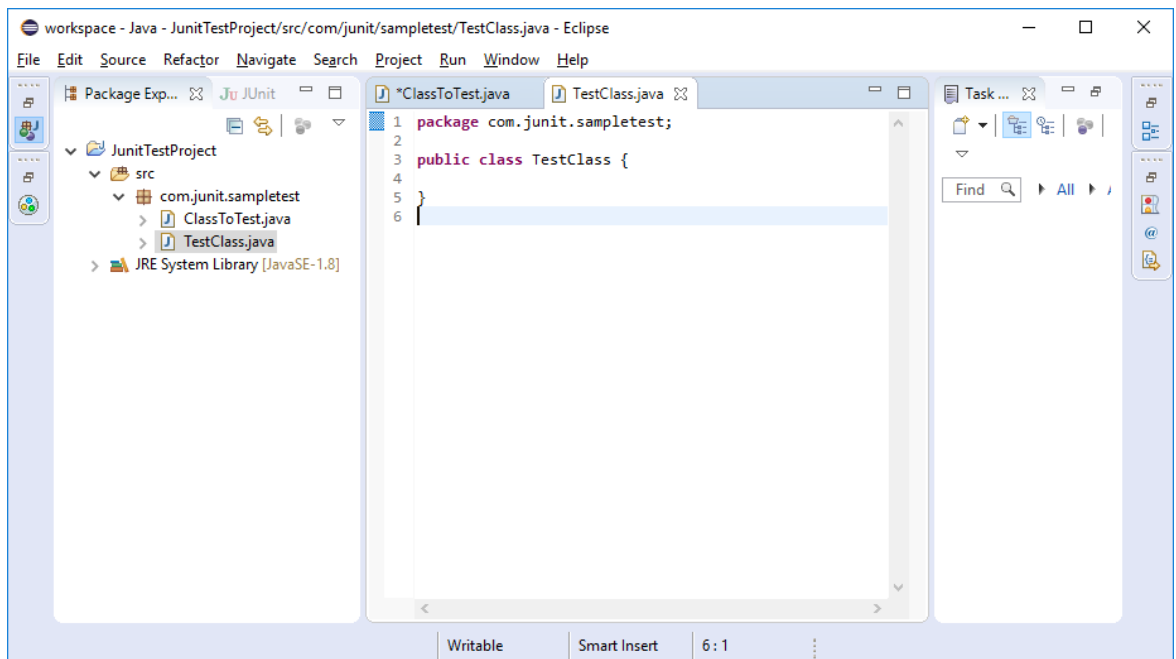


Рисунок 20 – Результат создания класса

18. Напишите код теста для основного класса ClassToTest в классе TestClass как на рисунке 21.

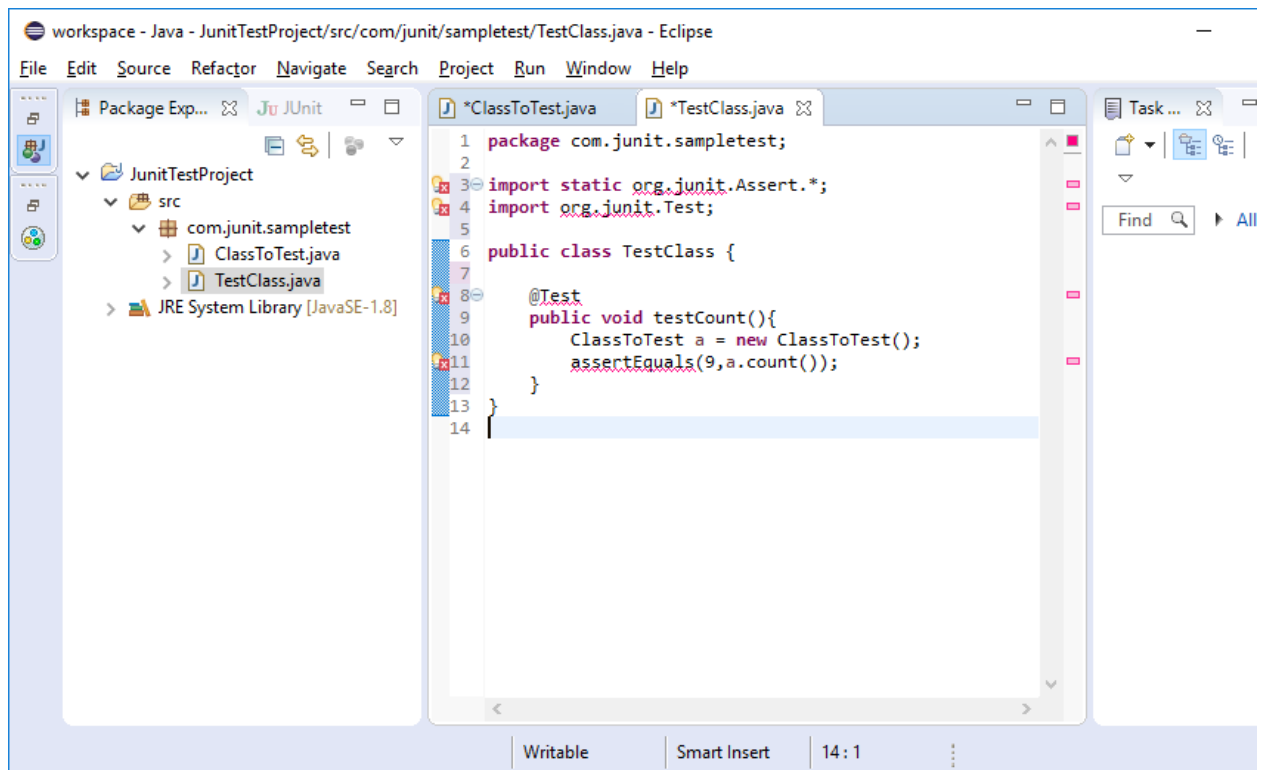


Рисунок 21 – Текст программы теста для класса ClassToTest

2.2 Подключение JUnit в Eclipse

Чтобы настроить JUnit с помощью Eclipse, выполните следующие действия:

1. Скачайте JUnit по ссылке:

<http://search.maven.org/remotecontent?filepath=junit/junit/4.10/junit-4.10.jar>.

2. Зайдите в настройку пути сборки. Щелкните правой кнопкой мыши по проекту JunitTestProject и выберите Build Path/Configure Build Path как показано на рисунке 22.

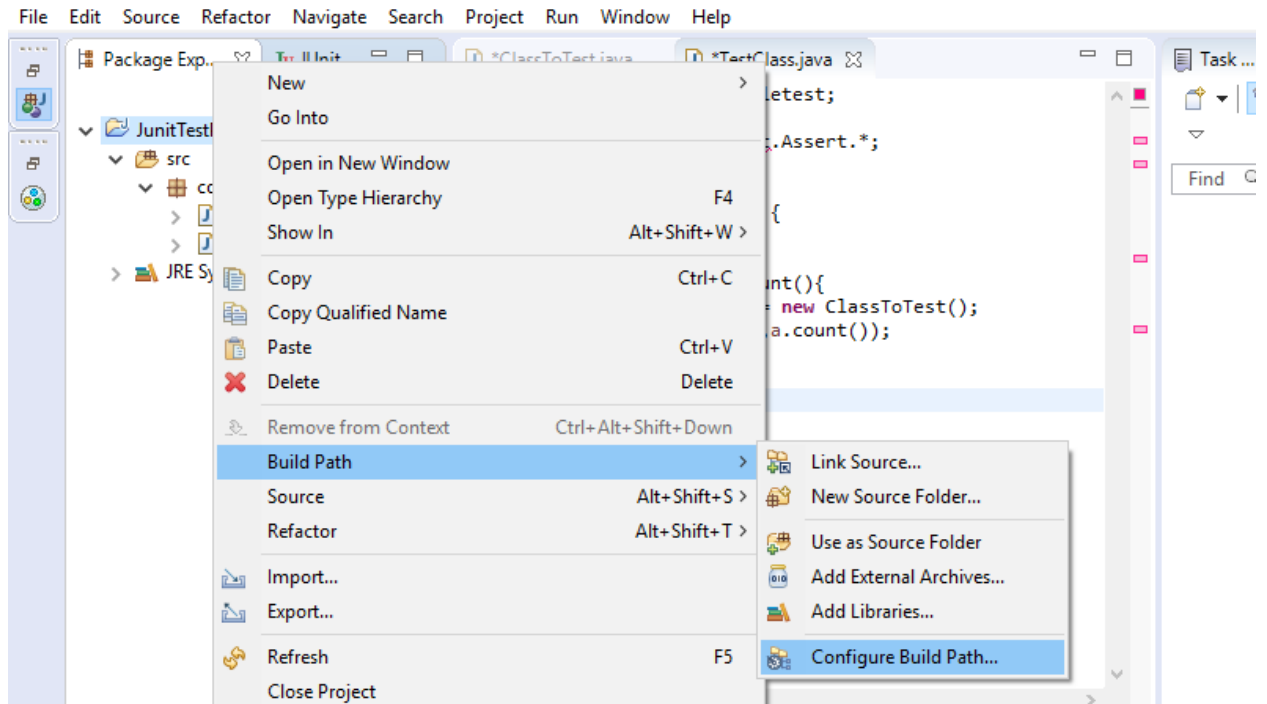


Рисунок 22 – Настройка пути сборки

3. Добавление junit-4.10.jar. Выберите слева Java Build Path/Libraries/Add External JARs... как показано на рисунке 23.

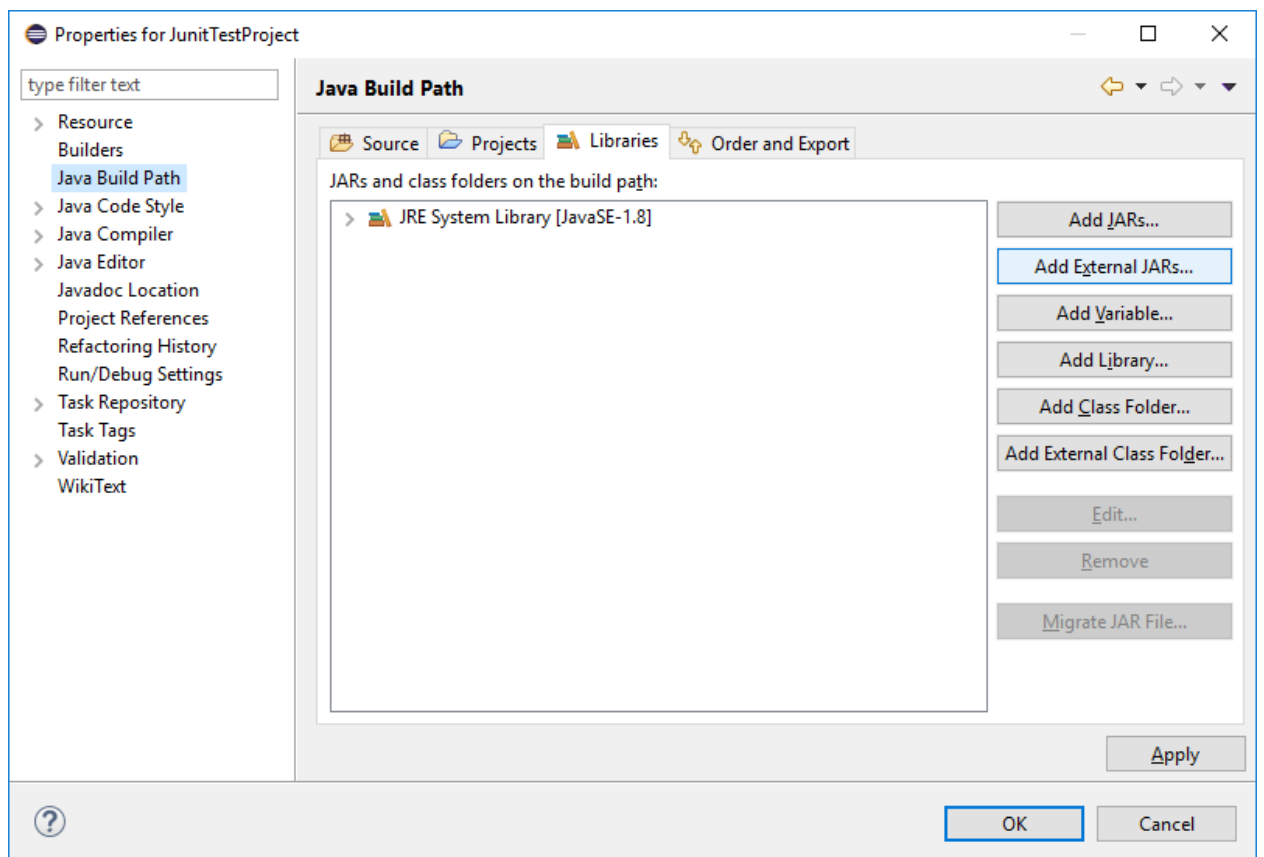


Рисунок 23 – Добавление junit-4.10.jar

4. Вид окна Eclipse после добавления junit-4.10.jar показаны на рисунках 24-25.

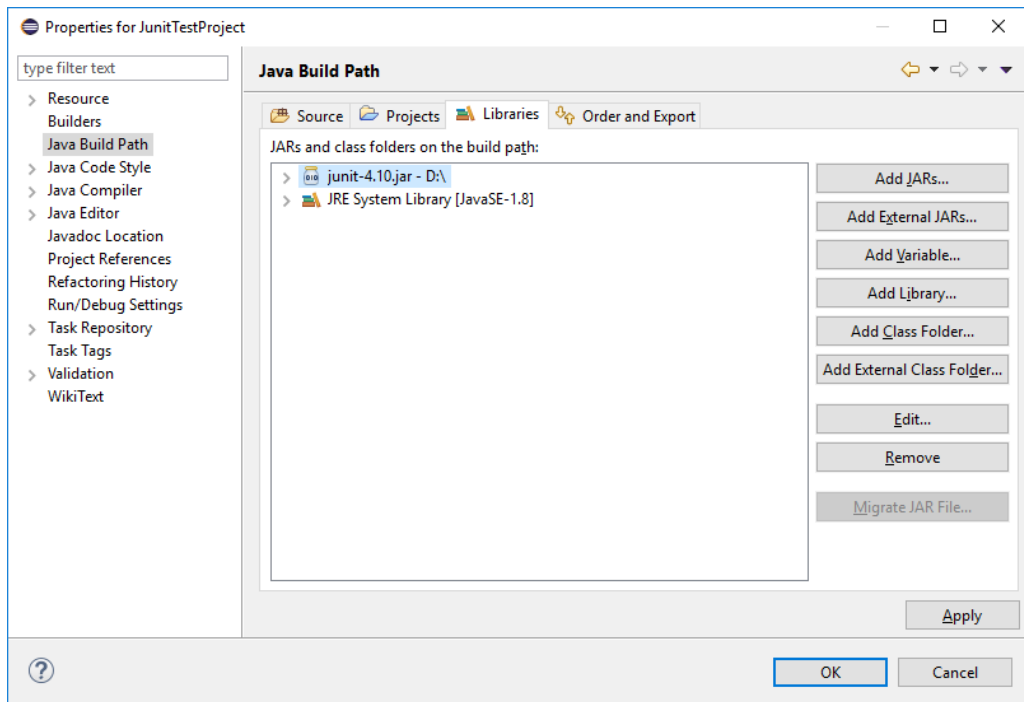


Рисунок 24 – Результат добавления junit-4.10.jar

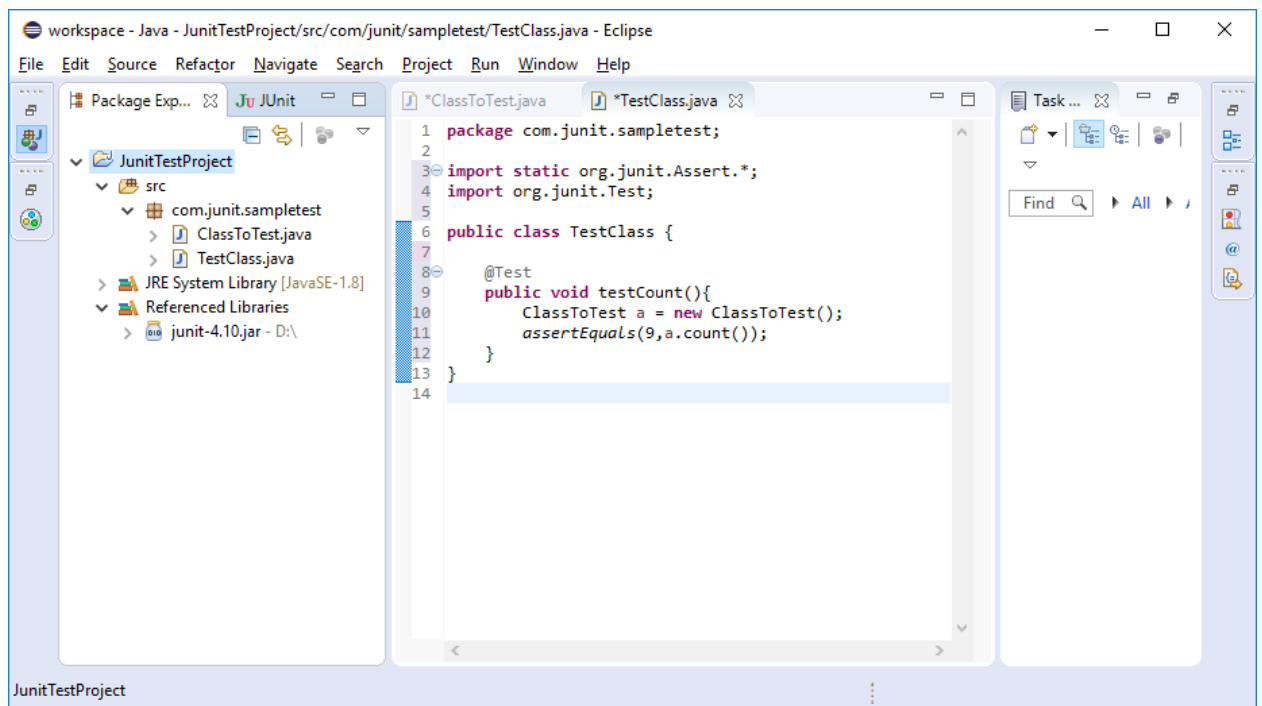


Рисунок 25 – Вид окна Eclipse после добавления junit-4.10.jar

5. Проверка теста. Щелкните правой клавишей на TestClass: Rus As/1 JUnit Test как показано на рисунке 26.

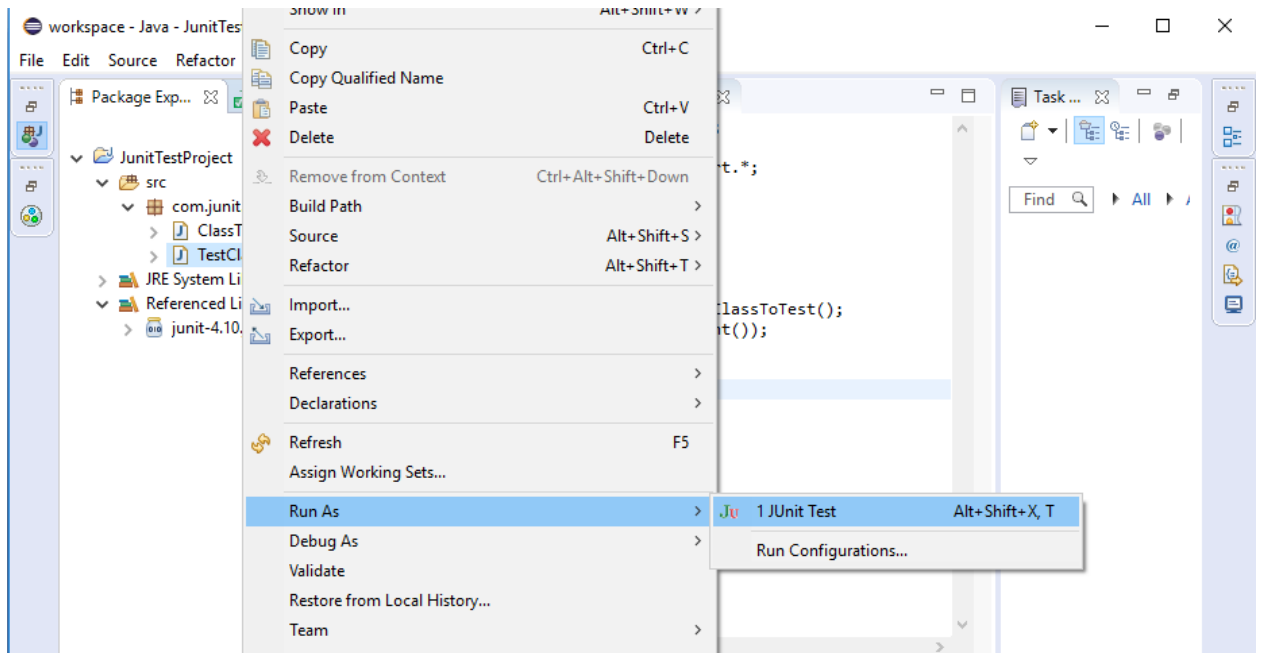


Рисунок 26 – Запуск тестируемого класса

6. Проверка результата теста. Результат проверки показан на рисунке 27.

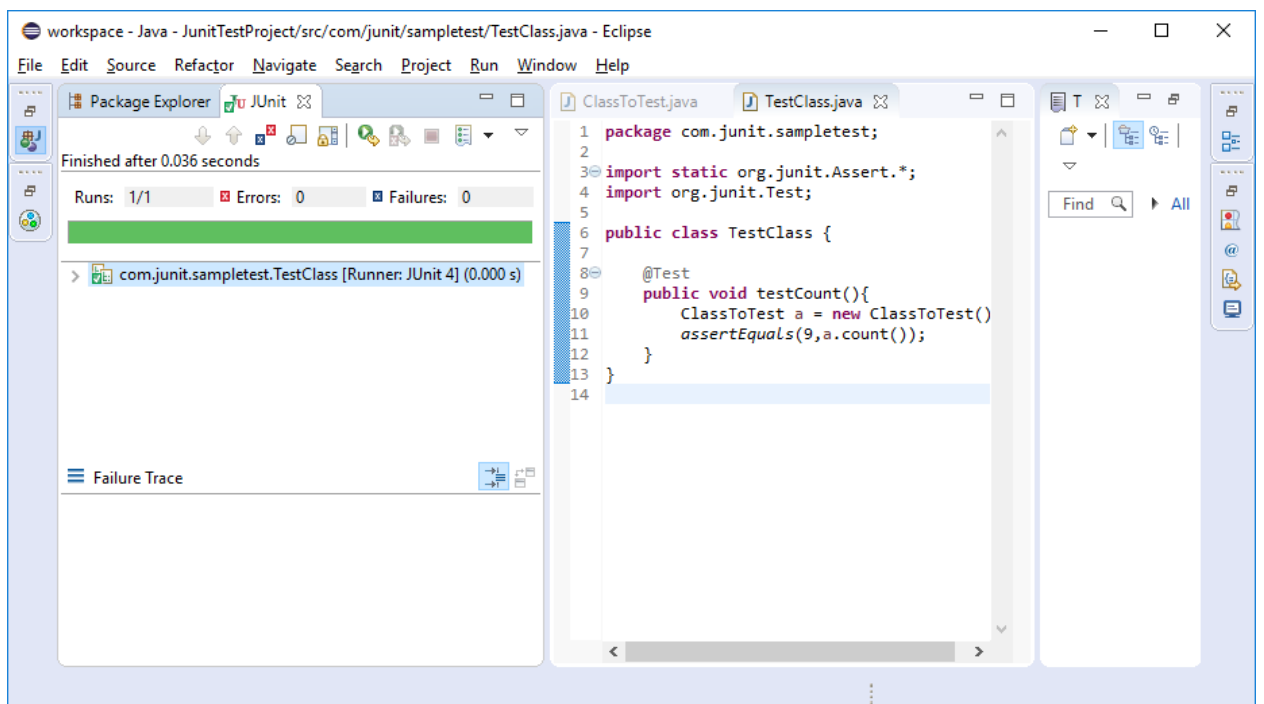



Рисунок 27 – Результат теста

7. По умолчанию это представление показывает все тесты. Чтобы показывать только неудачные тесты, нажмите на этот значок () как на рисунке 28.

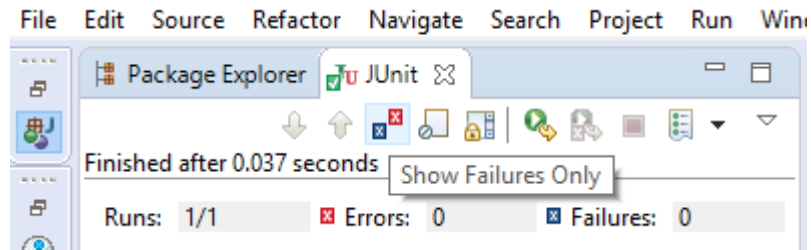



Рисунок 28 - Настройка показа тестов

8. Также можно определить, что представление активируется только в том случае, если есть тест с ошибкой. Для этого нажмите на этот значок () левой клавишей мыши и из выпадающего списка выбрать «Activate on Error/Failure Only» как на рисунке 29.

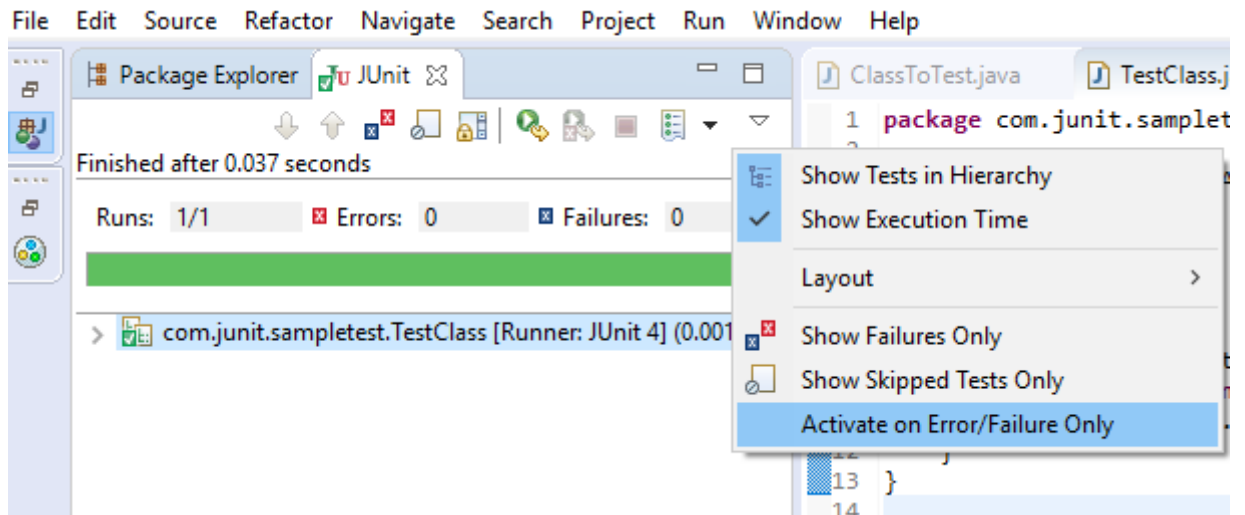


Рисунок 29 - Активация показа тестов с ошибкой

9. Чтобы получить список неудавшихся тестов, щелкните правой кнопкой мыши на результатах теста и выберите «Copy Failure List» как на рисунке 30.

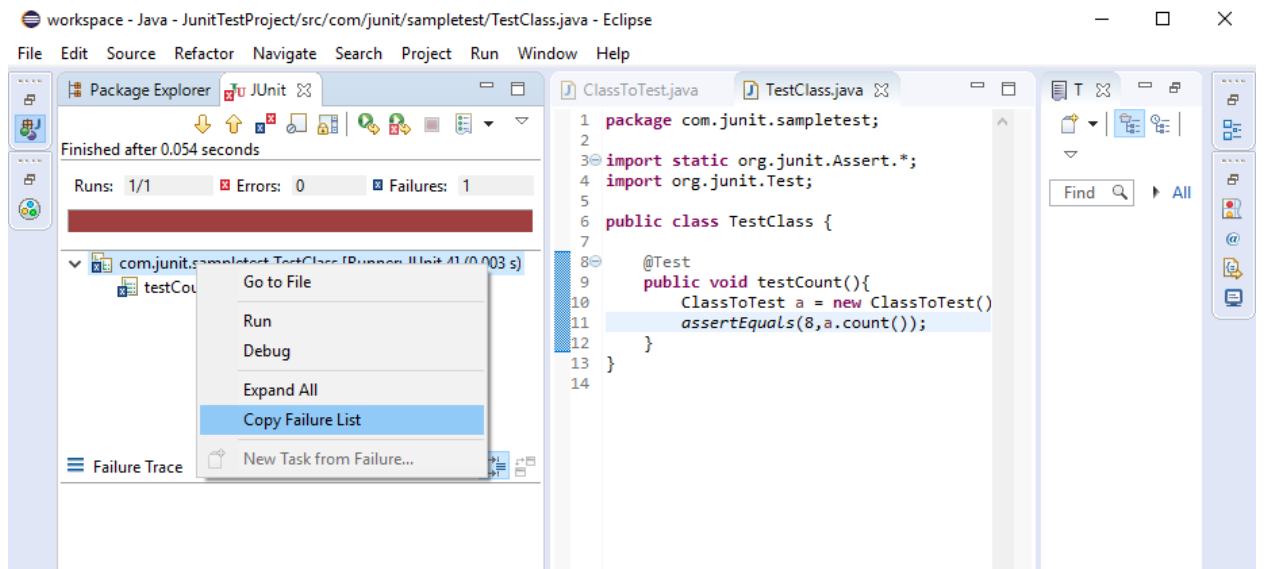


Рисунок 30 - Извлечение неудавшихся тестов и стеков

2.3 Мастер создания тестовых наборов

Чтобы создать тестовый набор через Eclipse, выполните следующие действия:

1. Откройте мастер создания. Кликните правой клавишей мыши на пакете `com.junit.sampletest`, выберите `New/Class/Other...` как показано на рисунке 31.

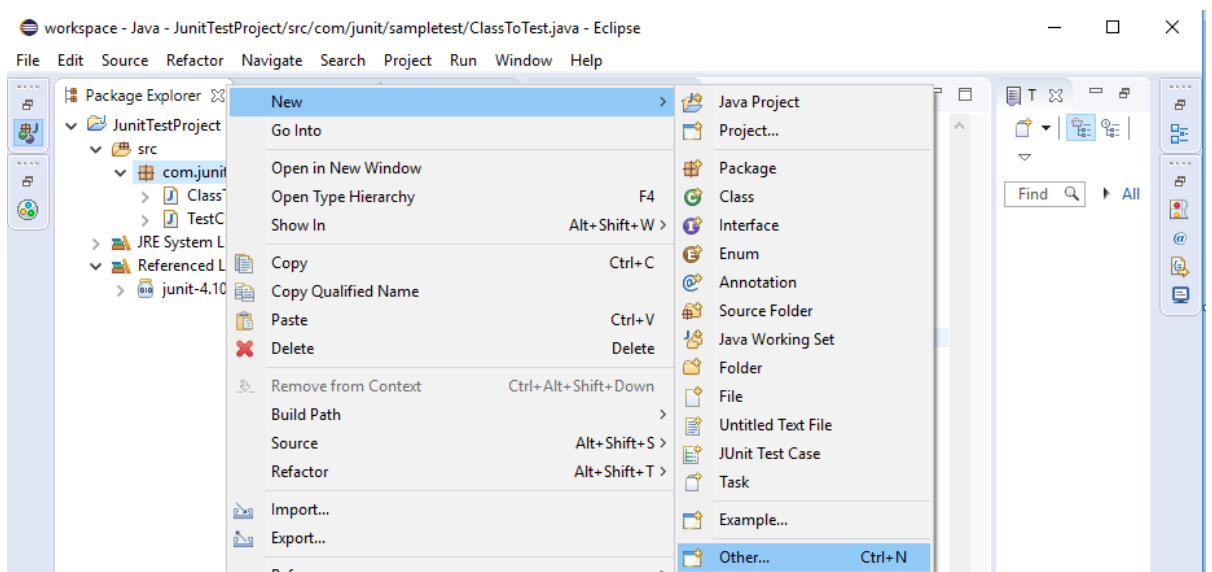


Рисунок 31 – Открытие мастера создания тестовых случаев

2. Раскройте папку Java. Далее раскройте папку JUnit и выбрать JUnit Test Suite как на рисунке 32.

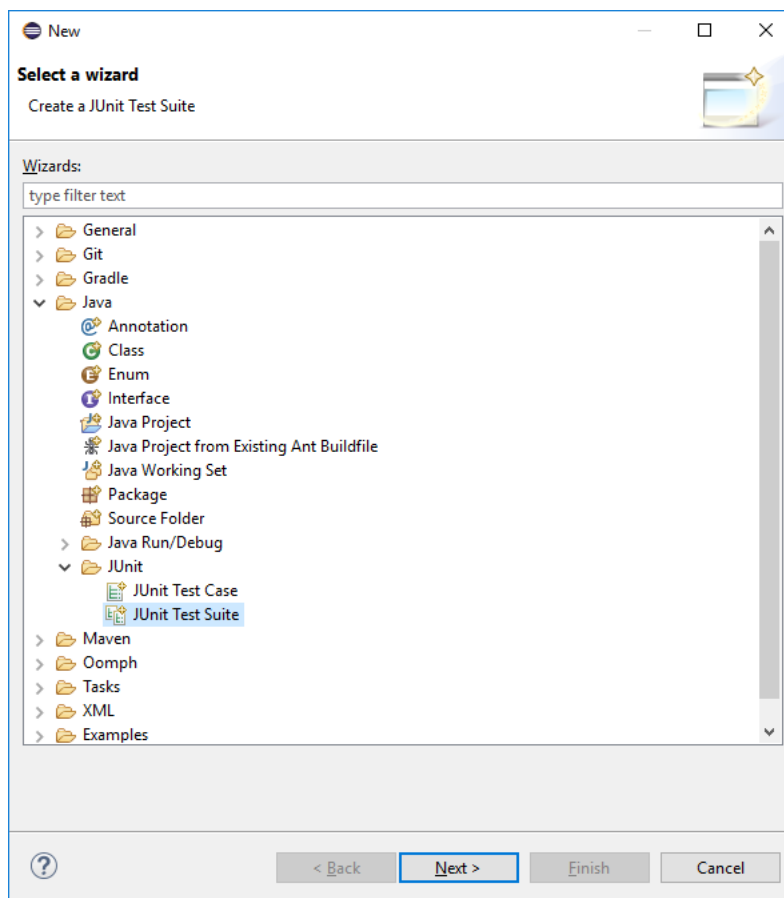


Рисунок 32 – Открытие JUnit Test Suite

3. Нажмите Next >.

4. В открывшемся окне впишите имя класса комплекта тестов «AllTest». Вид окна создания класса комплекта тестов показано на рисунке 25.

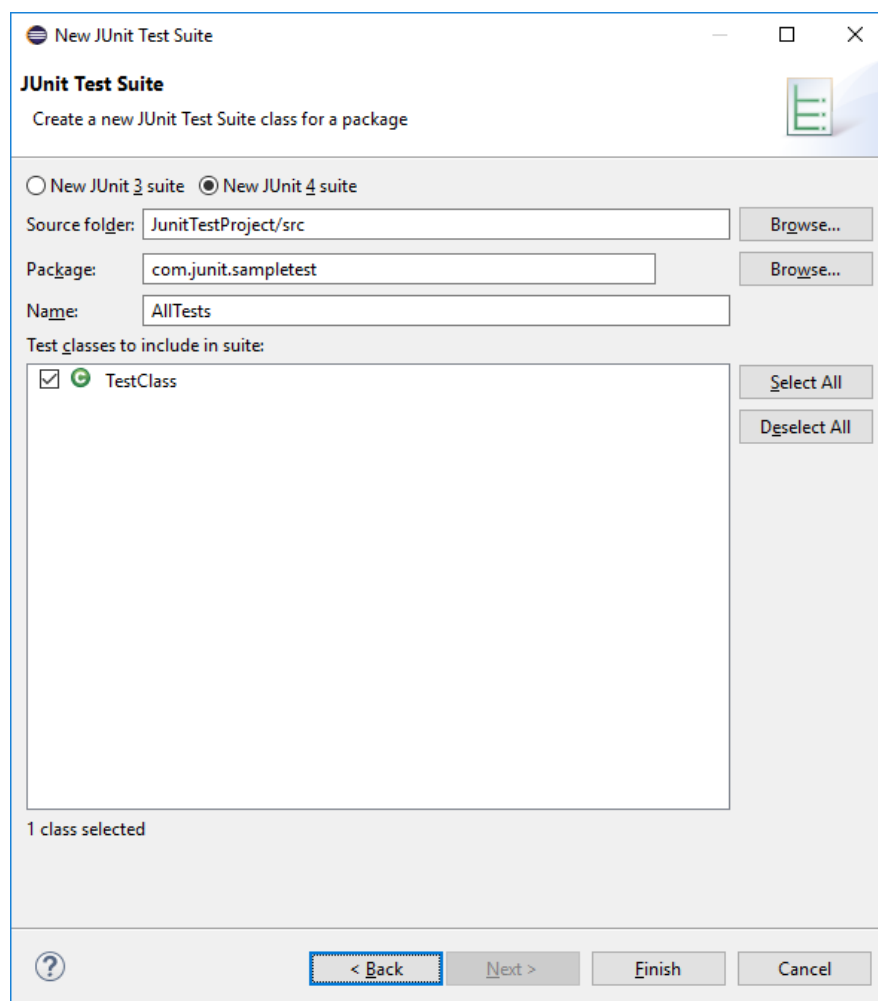


Рисунок 33 – Вид окна класса комплекта тестов

5. Выберите классы, которые следует включить в комплект. В настоящее время есть только один класс тестов, но пополнить комплект можно позже.
6. Нажмите Finish.
7. Вид окна Eclipse после создания класса комплекта тестов показан на рисунке 34.

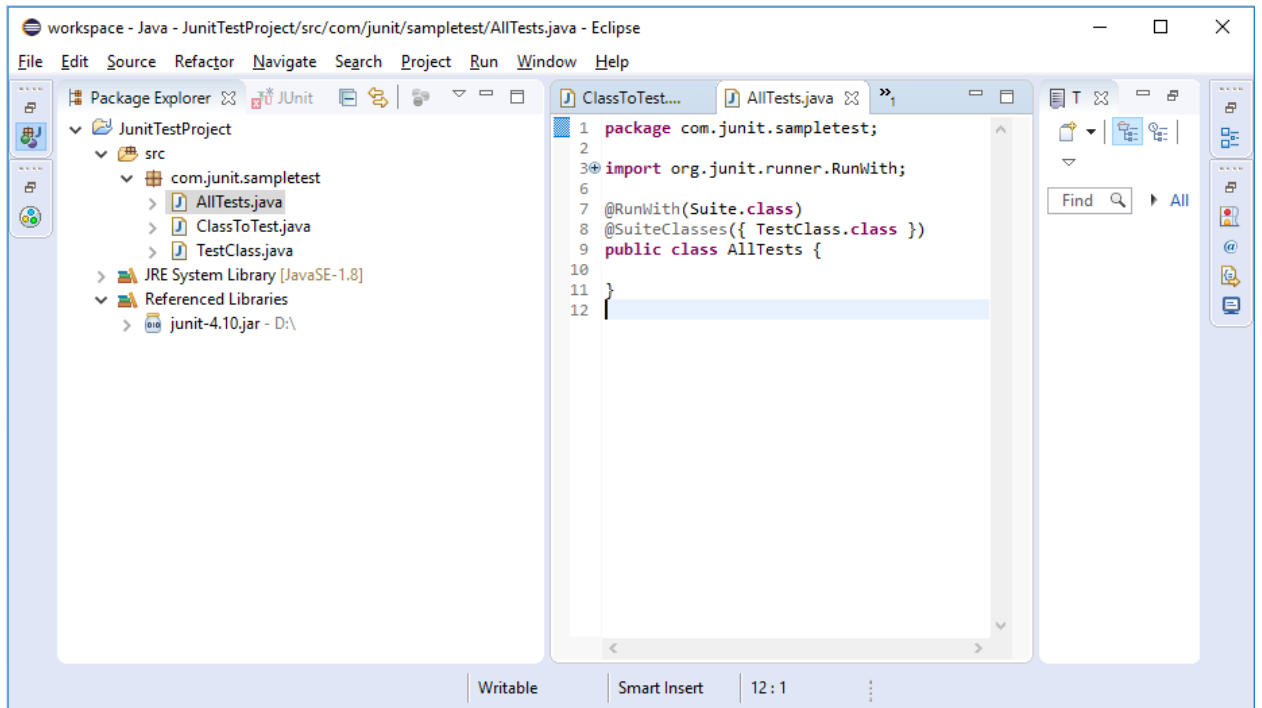


Рисунок 34 – Результат создания класса комплекта тестов

Добавлять и удалять классы тестов из комплекта тестов можно двумя способами:

1. Вручную - отредактировать файл комплекта тестов.
2. Повторно запустить мастер и выбрать новый набор классов тестов.

3 Содержание отчета по лабораторной работе

В сводный отчет по лабораторным работам в качестве одного из разделов или подразделов включается создания теста в IDE Eclipse.

4 Вопросы к защите лабораторной работы

1. Что такое JUnit?
2. Как создать класс в Eclipse?
3. Как добавить библиотеку в Eclipse?
4. Как получить список неудачных тестов в Eclipse?