Оленчикова Т.Ю.

**Методические указания и задачи к выполнению лабораторной работы по дисциплине «**Программирование на языке Java для анализа данных**»**

**Лабораторная работа 2 Модульное тестирование**

**Цель**: изучить понятие «модульное тестирование», получить практические навыки модульного тестирования в среде Eclipse/

**Теоретические материалы**

1) см. приложение 1

# 2) [Тестирование программы, JUnit](https://java-online.ru/blog-junit.xhtml)

Задание для лабораторной работы.

Используя технологию JUnit разработать Unit-тест для проверки работы классов, созданных на лабораторной работе Тест должен содержать соответствующие методы проверки правильности полученного решения.

**Требования к оформлению отчета.**

Отчет по ЛР состоит из

**а)** Титульный лист: название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания; в) UML-диаграмму классов (создаете в Umlet):

г) Исходный текст программы;

д) Скрины выполнения;

е) Выводы.

**Критерии оценивания.**

1. Задание выполнено полностью. Созданы все необходимые тесты – 2 балла; созданы не все тесты – 1 балл; Тесты не созданы – 0 баллов

2. Выполнен контрольный пример на тестирование. Отчет оформлен в соответствии с требованиями – 1 балл

3. Полные и верные ответы на контрольные вопросы – 1 балл

Итого – максимум 4 балла.

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%.

**Контрольные вопросы**

1. Тестирование как составляющий этап процесса разработки программных систем. Назначение модульного тестирования.
2. Принципы модульного тестирования.
3. Отличительные особенности библиотеки модульного тестирования JUnit.
4. Понятия модульного теста и тестируемого модуля. Последовательность действий для организации unit-тестирования.
5. Прокомментировать использованные при выполнении работы аннотации библиотеки JUnit.
6. Назначение assert-методов библиотеки JUnit. Примеры использования.
7. Прокомментировать результат тестирования ("failed" или "passed") применительно к строке кода "assertEquals(calc.getResult(), 9, 0);". Аналогичный ли будет результат в случае замены приведенной строки строкой "assertEquals(calc.getResult(), 8, 1);"?
8. Обосновать целесообразность использования аннотации "@Test(timeout=var)" вместо аннотации "@Test".
9. Назначение параметризированного тестирования. Привести пример.
10. Назначение Suite-тестирования.

**Библиографический список**

# Тестирование программы, JUnit [Электронный ресурс]. – URL: <https://java-online.ru/blog-junit.xhtml>

# Тестирование JUnit. Примеры. [Электронный ресурс]. – URL: <https://www.bestprog.net/ru/2021/09/20/java-junit-testing-example-ru/#more-34457>

**Приложение 1 Модульное тестирование Junit**

##### **1. Понятие о модульном тестировании в Java**

В системах программирования на языке Java, в том числе Java Eclipse, введены средства тестирования, которые предусматривают создание и использование автоматических тестов для разработанных программ. Правильно построенное тестирование позволяет свести количество ошибок в программе к нулю.

При разработке современных программных систем популярна методология TDD (Test Driven Development). Основная идея применения TDD заключается в том, что сначала разрабатываются тесты на желаемое поведение программы (системы), а затем пишется рабочий программный код, который с помощью средств рефакторинга проверяется уже созданными тестами. Как следствие, заранее подготовленные тесты позволяют избежать сдачи в эксплуатацию программы, содержащей ошибки.

Для проведения тестирования в Java Eclipse применяется так называемое JUnit-тестирование, которое является фактическим стандартом для тестирования Java-приложений. JUnit — это обычная среда (framework), позволяющая писать тесты в виде классов Java. Эти классы могут запускаться как единое целое с помощью средства выполнения тестов.

JUnit имеет много расширений, которые внедрены во всех современных средах разработки на языке Java. На сегодняшний день JUnit используется в таких направлениях как:

* модульное тестирование;
* тестирование веб-сайтов;
* автоматическая подстановка интерфейсов для тестирования;
* тестирование параллельного выполнения приложений;
* тестирование производительности.

JUnit-тестирование используется многими группами тестировщиков (quality assurance) на специально-разработанных веб-сайтах, обеспечивающих автоматизацию применения сквозных функциональных тестов.

В Java Eclipse использование механизма JUnit не требует работающего сервера приложений или действующей базы данных.

##### 2. Понятие тестового примера (Test Case). Класс, содержащий тестовые примеры (методы). Запуск теста

При использовании механизма JUnit, рассматриваются две основные единицы (классы):

* класс, методы которого нужно протестировать;
* класс, который представляет собой JUnit тест. Этот класс построен по специальным правилам.

Значит, JUnit-тест представляет собой специально разработанный класс, содержащий методы, которые тестируют программный код других классов. В системе Java Eclipse этот класс создается с помощью команды JUnit Test Case (смотрите примеры далее).

Класс содержит ряд методов. Каждый из методов тестирующего класса рассматривается как отдельный тестовый пример — Test Case.

Тестовые примеры (методы) JUnit-класса могут быть объявлены с использованием следующих аннотаций (рассматривается версия Java Eclipse 2018-09):

* @BeforeAll — статический метод (тестовый пример), объявленный с этой инструкцией, вызывается в начале тестирования;
* @AfterAll — статический метод, объявленный с этой инструкцией, вызывается в конце тестирования. Здесь можно задавать, например, освобождение ресурсов используемых при тестировании;
* @Test — метод, который объявлен с этой инструкцией, является тестовым примером. Он содержит непосредственно код тестирования. Этот метод использует методы класса org.junit.jupiter.api.Assertions. В этом методе используются методы сравнения, которые начинаются с префикса assert\*(). Например, для сравнения двух значений любого примитивного типа используется перегруженный метод assertEquals(). В одном тестовой классе JUnit может быть произвольное количество методов (примеров) тестирования, которые объявлены с аннотацией @Test;
* @BeforeEach — вызывается перед вызовом каждого тестового примера (метода), который объявлен с аннотацией @Test;
* @AfterEach — вызывается после завершения каждого тестового примера (метода), который объявлен с аннотацией @Test.

Допускается произвольное количество методов со всеми вышеприведенными аннотациями.

В наиболее общем случае, примерный вид класса, сгенерированный системой Java Eclipse, может быть таким

**class** TestClass {

@BeforeAll

**static** **void** setUpBeforeClass() **throws** Exception {

}

@AfterAll

**static** **void** tearDownAfterClass() **throws** Exception {

}

@BeforeEach

**void** setUp() **throws** Exception {

}

@AfterEach

**void** tearDown() **throws** Exception {

}

@Test

**void** testMethod() {

   fail("Not yet implemented");

}

}

В методе testMethod() нужно вставить собственный код работы методов другого класса (смотрите примеры ниже). После этого, можно запускать тест с помощью специальной команды Java Eclipse.

##### 3. Пример, демонстрирующий использование JUnit для проверки правильности решения квадратного уравнения. Пошаговая инструкция

##### 3.1. Условие задачи

Используя технологию JUnit разработать Unit-тест для проверки работы класса SquareEquation, который содержит средства решения квадратного уравнения.

Тест должен размещаться в классе SquareEquationTest и содержать соответствующие методы проверки правильности полученного решения.

Корни квадратного уравнения возвращаются в виде экземпляра класса Roots.

Класс SquareEquationTest содержит соответствующие средства решения квадратного уравнения.

##### 3.2. Решение

##### 3.2.1. Создание классов SquareEquation и Roots

Перед созданием классов создается проект стандартным для Java Eclipse способом. В проект добавляются два класса с именами SquareEquation и Roots и формируется код этих классов. Соответственно создается файл SquareEquation.java. После выполненных действий, окно Java Eclipse имеет вид как показано на рисунке 1.

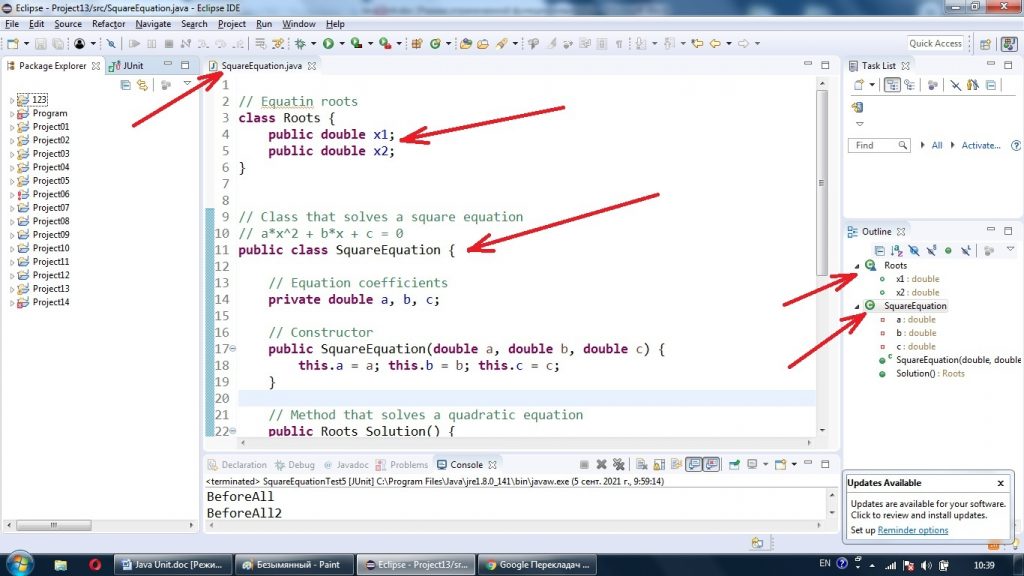
[](https://www.bestprog.net/wp-content/uploads/2021/09/01_-1.jpg)

Рисунок 1. Окно Java Eclipse. Классы Roots и SquareEquation

Классы имеют следующее назначение:

* Roots — используется для сохранения корней квадратного уравнения (x1, x2) в случае, если уравнение имеет эти корни;
* SquareEquation — реализует методы решения квадратного уравнения.
* Класс SquareEquation имеет следующие составляющие:
* внутренние переменные **a**, **b**, **c** что являются коэффициентами квадратного уравнения;
* конструктор, инициализирует значения внутренних полей **a**, **b**, **c**;
* метод Solution(), который возвращает экземпляр класса Roots в случае, если уравнение имеет корни. Если уравнение не имеет корней, то генерируется исключение типа ArithmeticException.

Полный текст классов Roots и SquareEquation выглядит так

// Корни уравнения

**class** Roots {

**public** **double** x1;

**public** **double** x2;

}

// Класс, решающий квадратное уравнение - этот класс будет протестирован другим классом.

// a\*x^2 + b\*x + c = 0

**public** **class** SquareEquation {

// Коэффициенты уравнения

**private** **double** a, b, c;

// Конструктор

**public** SquareEquation(**double** a, **double** b, **double** c) {

**this**.a = a; **this**.b = b; **this**.c = c;

}

// Метод, решающий квадратное уравнение

**public** Roots Solution() {

// Дискриминант

**double** d = b\*b - 4\*a\*c;

// Проверка, имеет ли уравнение корни

**if** (d<0)

**throw** **new** ArithmeticException("Solution has no roots.");

// Вычислить результат

Roots result = **new** Roots();

result.x1 = (-b - Math.sqrt(d)) / (2\*a);

result.x2 = (-b + Math.sqrt(d)) / (2\*a);

// Вернуть результат

**return** result;

}

}

После создания классов можно переходить к созданию Unit-теста.

##### 3.2.2. Создание класса, необходимого для реализации JUnit тестирования

Для создания Unit-теста в Java Eclipse нужно выбрать последовательность команд

File->New->JUnit Test Case

как показано на рисунке 2.

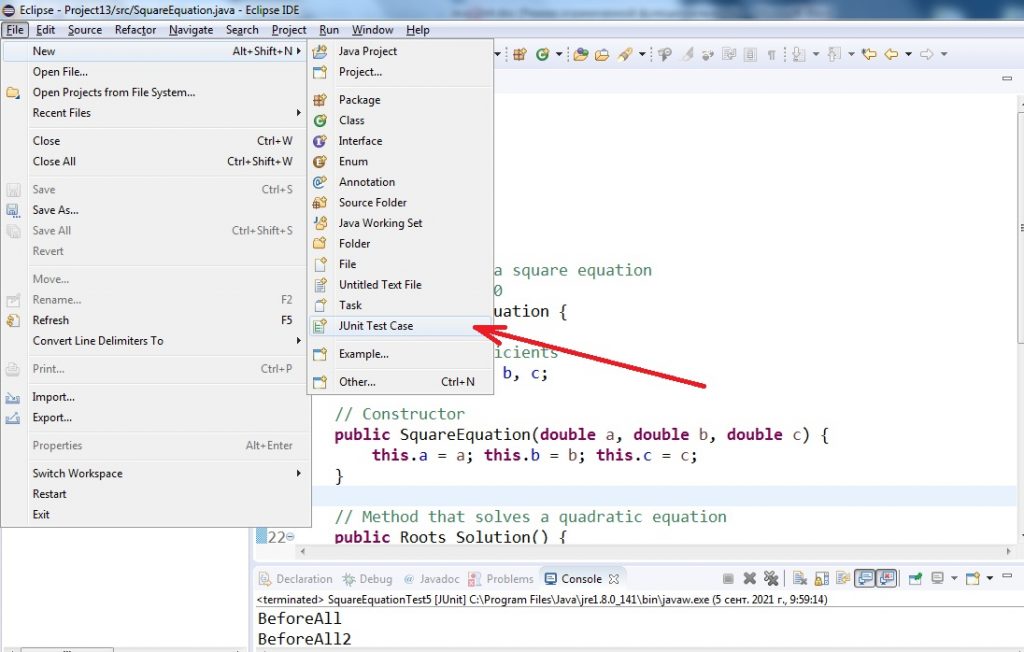
[](https://www.bestprog.net/wp-content/uploads/2021/09/02_-1.jpg)

Рисунок 2. Команда создания Unit-теста в Java

В результате откроется окно «New JUnit Test Case» (рисунок 3) в котором нужно указать необходимую информацию о тесте.

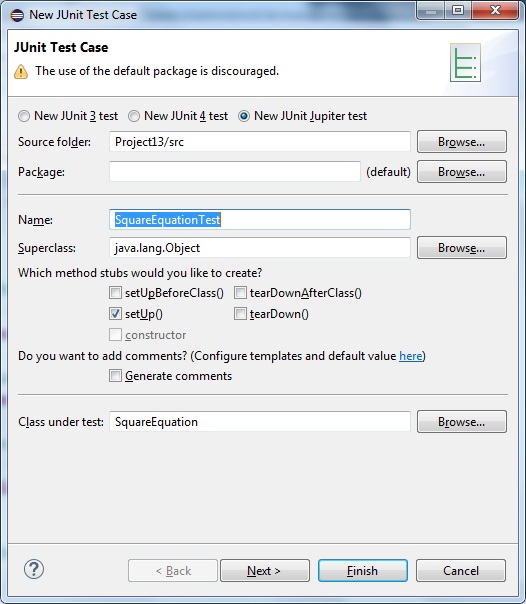
[](https://www.bestprog.net/wp-content/uploads/2021/09/03_-1.jpg)

Рисунок 3. Окно создания нового теста. Указание параметров теста

При создании окна система автоматически заполнит некоторые поля. В окне указывается следующая информация.

1. В поле Source Folder указывается имя папки с исходными файлами проекта. Система автоматически «подтягивает» имя папки с текущим проектом.
2. В поле Name — задается имя класса, в котором будут размещаться методы тестирования (с аннотациями @BeforeAll, @Test т.д.). В нашем случае предлагается SquareEquationTest (к имени класса SquareEquation добавляется окончание Test). Предложенное имя можно оставить без изменений.
3. В поле «Which method stubs would you like to create?» с помощью опций указываются методы, которые нужно сформировать в классе. Согласно выбранной опции формируются следующие методы:
   * setUpBeforeClass() — имя метода аннотации @BeforeAll (см п.2);
   * setUp() — имя метода аннотации @BeforeEach;
   * tearDownAfterClass() — имя метода аннотации @AfterAll;
   * tearDown() — имя метода аннотации @AfterEach.

Для нашего теста достаточно активировать только одно поле setUp(). Это позволит добавить метод аннотации @BeforeEach, что будет вызываться перед тестом. В этом методе будет создаваться экземпляр класса SquareEquation (смотрите коды ниже).

1. В поле «Class under test:» указывается имя класса SquareEquation, методы которого нужно протестировать.
2. Дополнительно задаются другие опции:
   * выбор модели тестирования «New JUnit Jupiter test»;
   * возможность добавления комментариев;
   * другое.

В нашем случае можно оставить все без изменений.

После выбора кнопки «Next >» откроется следующее окно «New JUnit Test Case» в котором нужно задать перечень методов, которые будут тестироваться. Вид окна показан на рисунке 4. Избранные методы будут объявляться с аннотацией @Test. Это непосредственно тестовые примеры. Все остальные опции в окне можно оставить по умолчанию и перейти далее с помощью кнопки Finish.

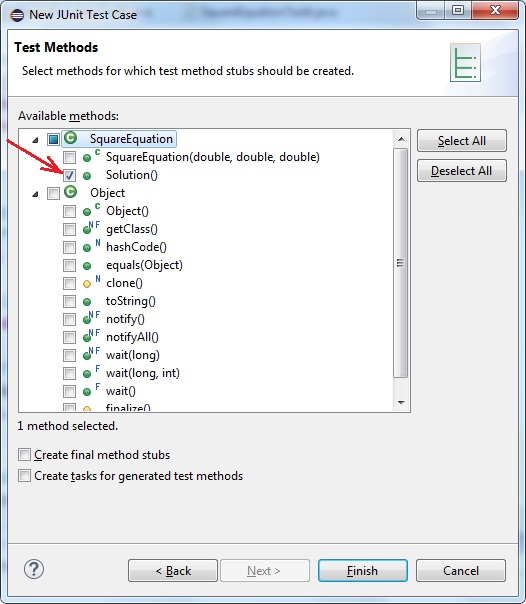
[](https://www.bestprog.net/wp-content/uploads/2021/09/04_-1.jpg)

Рисунок 4. Выбор методов которые будут тестироваться. Выбран метод Solution()

##### 3.2.3. Код класса SquareEquationTest

После создания JUnit теста с помощью оконного интерфейса Java Eclipse сгенерирует код класса SquareEquationTest

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**class** SquareEquationTest {

@BeforeEach

**void** setUp() **throws** Exception {

}

@Test

**void** testSolution() {

fail("Not yet implemented");

}

}

В этом коде сформированы два метода:

* метод setUp() с аннотацией @BeforeEach. Этот метод будет вызываться первым при тестировании;
* метод testSolution() с аннотацией @Test. В этом методе будет помещен непосредственно тест функции Solution() класса SquareEquation.

##### 3.2.4. Модификация класса SquareEquationTest. Программирование теста

Чтобы протестировать работу класса SquareEquation нужно создать экземпляр этого класса. Поэтому, в текст класса SquareEquationTest дополнительно вводится ссылка на класс SquareEquation

**class** SquareEquationTest {

**private** SquareEquation se;

...

}

Затем в методе setUp() создается экземпляр класса SquareEquation, на который указывает ссылка se

@BeforeEach

**void** setUp() **throws** Exception {

// Этот метод вызывается первым.

// Создать экземпляр класса SquareEquation.

se = **new** SquareEquation(2, 1, -3); // 2\*x^2 + x - 3 = 0

}

После этого, в методе TestSolution() вписывается код проверки правильности полученного решения для заданных коэффициентов

a = 2

b = 1

c = -3

в экземпляре se

@Test

**void** testSolution() {

// Объявить экземпляр класса Roots

Roots rt = se.Solution();

// Проверка решения x1

assertEquals(rt.x1, -1.5);

// Проверка решения x2

assertEquals(rt.x2, 1.0);

}

Проверка решения осуществляется с помощью перегруженного метода assertEquals() класса org.junit.jupiter.api.Assertions. Происходит сравнение решения в переменных rt.x1 и rt.x2 с заранее вычисленным (вручную) решением текущего варианта (a = 2, b = 1, c = -3) квадратного уравнения. Если есть совпадение, то решение правильное. О том, что решение правильное, нужна сказать система после запуска теста (следующий пункт).

В целом текст всего модуля тестирования следующий

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**class** SquareEquationTest {

// Ссылка на класс SquareEquation

**private** SquareEquation se;

@BeforeEach

**void** setUp() **throws** Exception {

// Этот метод вызывается первым.

// Создать экземпляр класса SquareEquation

se = **new** SquareEquation(2, 1, -3); // 2\*x^2 + x - 3 = 0

}

@Test

**void** testSolution() {

// Объявить экземпляр класса Roots

Roots rt = se.Solution();

// Проверка решения x1

assertEquals(rt.x1, -1.5);

// Проверка решения x2

assertEquals(rt.x2, 1.0);

}

}

##### 3.3. Запуск теста средствами Java Eclipse. Просмотр результата тестирования

После того, как сформирован код класса SquareEquationTest, разработанный тест можно запускать на выполнение. Это осуществляется командами

Run -> Run As -> JUnit Test

или вызовом последовательности комбинаций клавиш Alt + Shift + X, T как показано на рисунке 5.

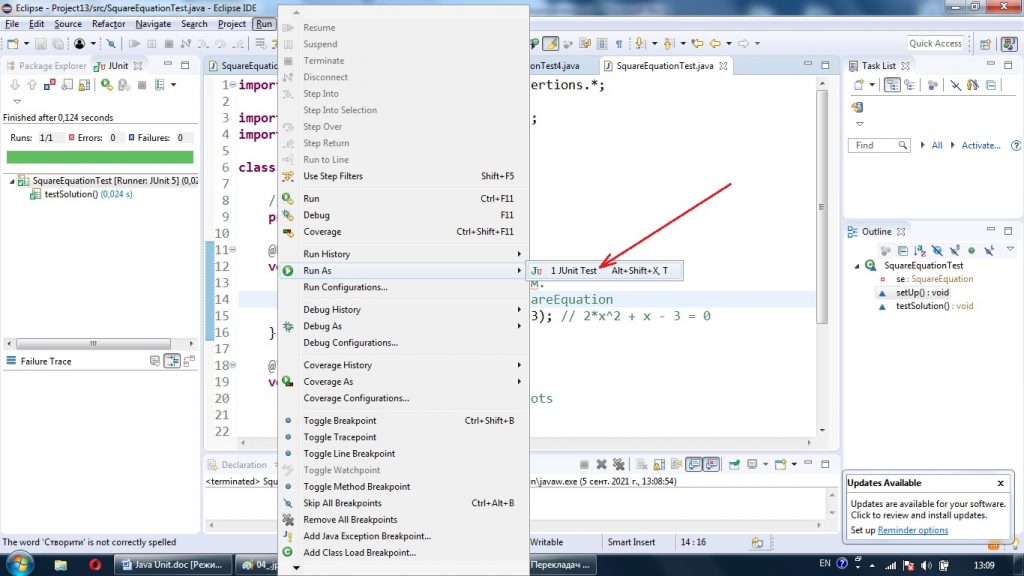
[](https://www.bestprog.net/wp-content/uploads/2021/09/05_.jpg)

Рисунок 5. Запуск JUnit-теста

Существует другой способ с помощью команды быстрого меню (рисунок 6).

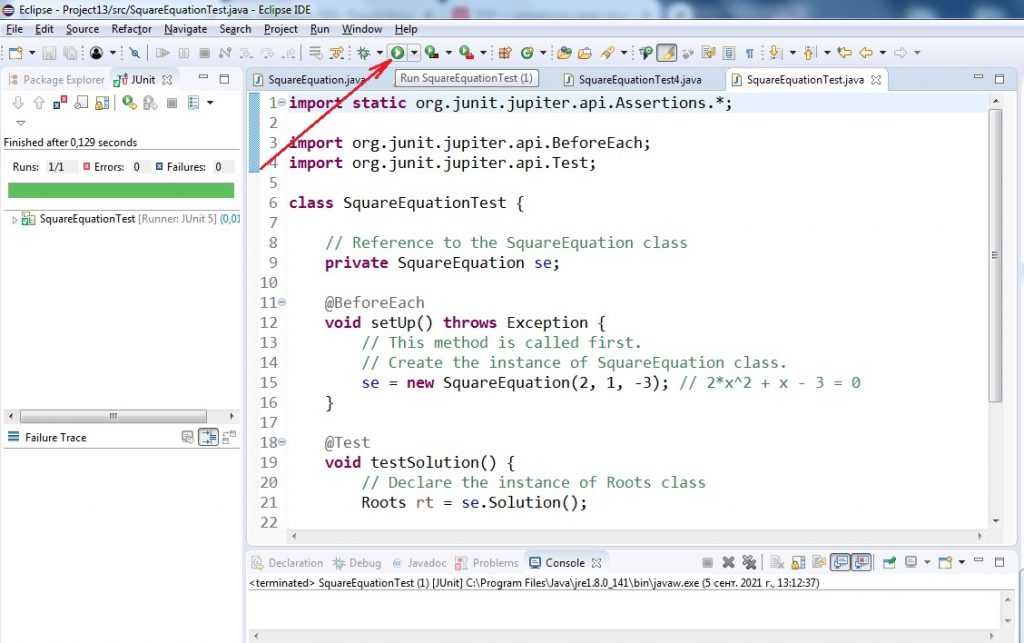
[](https://www.bestprog.net/wp-content/uploads/2021/09/06_.jpg)

Рисунок 6. Команда запуска JUnit-теста

После запуска в левой части Java Eclipse в окне JUnit отобразится результат тестирования (рисунок 7).

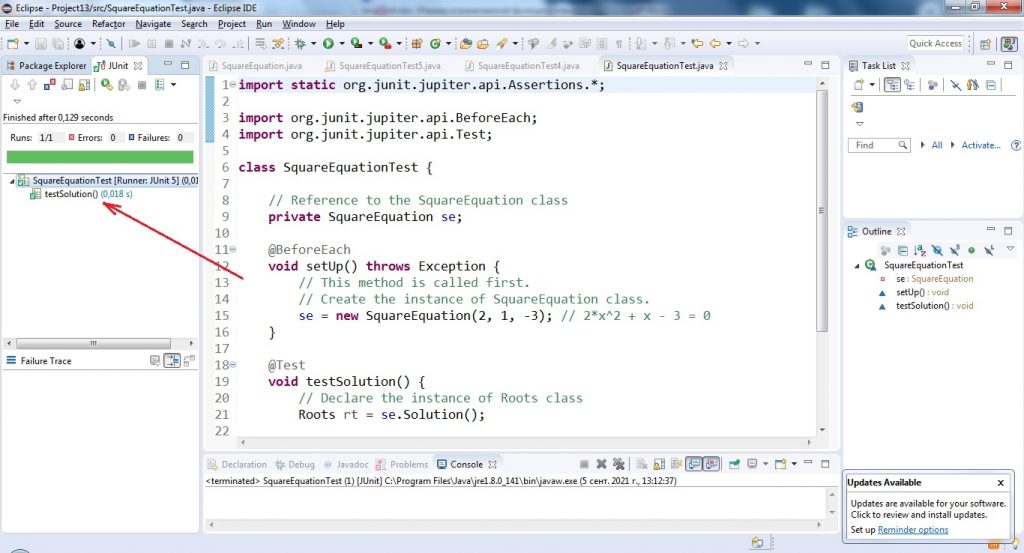
[](https://www.bestprog.net/wp-content/uploads/2021/09/07_.jpg)

Рисунок 7. Результат тестирования

Как видно из результата, получено подтверждение правильного решения, то есть тест сдан. Значит, метод Solution() класса SquareEquation дает правильный результат для случая когда a = 2, b = 1, c = -3.

Если в методе TestSolution() при вызове assertEquals() специально указать неверный ответ, например

assertEquals(rt.x2, 777.777);

то после повторного запуска теста, в окне JUnit отобразится ошибка как видно на рисунке 8. Хотя это не ошибка программы, а умышленная ошибка в тесте.

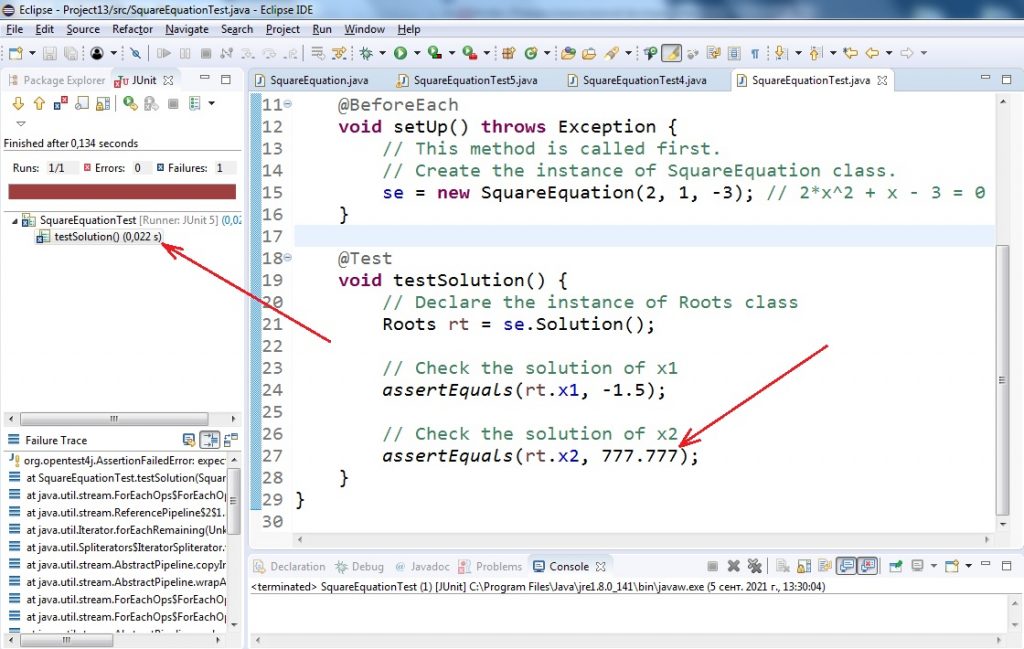
[](https://www.bestprog.net/wp-content/uploads/2021/09/08_.jpg)

Рисунок 8. Демонстрация провала теста

##### 4. Пример, демонстрирующий последовательность вызова методов в JUnit-тесте

Пусть в системе Java Eclipse создан класс с именем TestClass для проведения тестирования некоторого другого класса. В данном примере, суть тестируемого класса не имеет значения.

Программный код класса следующий

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.AfterAll;

**import** org.junit.jupiter.api.AfterEach;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**class** TestClass {

@BeforeAll

**static** **void** setUpBeforeClass() **throws** Exception {

System.**out**.println("BeforeAll");

}

@BeforeAll

**static** **void** setUpBeforeClass2() **throws** Exception {

System.**out**.println("BeforeAll2");

}

@AfterAll

**static** **void** tearDownAfterClass() **throws** Exception {

System.**out**.println("AfterAll");

}

@BeforeEach

**void** setUp() **throws** Exception {

System.**out**.println("BeforeEach");

}

@BeforeEach

**void** setUp2() **throws** Exception {

System.**out**.println("BeforeEach2");

}

@AfterEach

**void** tearDown() **throws** Exception {

System.**out**.println("AfterEach");

}

@Test

**void** testSolution() {

System.**out**.println("Test");

}

@Test

**void** testSolution2() {

System.**out**.println("Test2");

}

}

После запуска теста на выполнение, программа выдаст следующий результат

BeforeAll

BeforeAll2

BeforeEach2

BeforeEach

Test

AfterEach

BeforeEach2

BeforeEach

Test2

AfterEach

AfterAll

Проанализировав этот результат, можно прийти к выводу, что методы в тестовом классе вызываются в порядке, определяемом их аннотациями. Первыми вызываются методы аннотации @BeforeAll.

Следующими рассматриваются тестовые примеры, объявленные с аннотацией @Test. Перед каждым тестовым примером вызываются все методы с аннотацией @BeforeEach. После каждого тестового примера вызываются все методы с аннотацией @AfterEach.

Последними вызываются все методы с аннотацией @AfterAll.