

Модульное тестирование при помощи JUnit

Евгений Беркунский, НУК
eugeny.berkunsky@gmail.com
<http://www.berkut.mk.ua>



Програма

- Введение.
- Установка.
- Возможности JUnit.
- Пример тестового случая.
- Автоматизация разработки.
- Ссылки.

Введение

- Что такое модульное тестирование?
 - Тестирование отдельных функций системы.
 - Как правило выполняется разработчиком модуля.
 - Может быть легко автоматизировано.
 - Закладывает основу для регрессионного тестирования приложения.
- JUnit – это библиотека, позволяющая проводить модульное тестирование Java приложений.

Мифы тестирования

- «У меня нет времени на тесты».
- «Тестирование – скучное и не творческое занятие».
- «Мой код и так отлично работает».
- «Это работа для отдела тестирования. У них получится лучше».



Миф №1

«У меня нет времени на тесты»

Написание тестов стабилизирует код и позволяет существенно сократить время отладки.



Миф №2

«Тестирование – скучное и не творческое занятие»

- Многие книги по тестированию написаны сухо и излагают лишь некие теоретические основы.
- Однако предугадывание и предотвращение ошибок – весьма творческое и ответственное занятие.
- Далеко не каждый разработчик способен хорошо протестировать свой (и чужой) код.
 - Это очень ценный навык.
- Бесконечная отладка ошибок – куда менее творческое занятие, чем тестирование.



Миф №3

«Мой код и так отлично работает»

- Представим себе идеального разработчика. Он:
 - Прочитал все нужные книги.
 - Попал во все возможные ловушки.
 - Знает все паттерны проектирования.
 - Помнит наизусть детали реализации каждого из 363 классов своей системы.
- Как Вы думаете, изменив 35-ю строчку 276 -го класса, сможет ли он точно предсказать, как это скажется на остальных 362?
- Что говорить об остальных, не столь совершенных разработчиках?
- Не легче ли использовать набор надежных автоматических тестов в подобном случае – и не держать в голове лишние подробности?

Миф №4

«Это работа для отдела тестирования.

У них получится лучше»

- Увы, тестирование системы в целом далеко не всегда подходит для обнаружения ошибок в отдельных компонентах.
- Даже при точном воспроизведении ошибки и ее грамотном описании требуется немало времени, чтобы найти ответственного.
 - Зачастую это время других разработчиков.
- Ошибку, обнаруженную на ранней стадии разработки куда легче исправить.
- Системное тестирование очень важно, но вовсе не заменяет модульное тестирование.
 - Корректный модульный тест часто может написать лишь автор кода.

Пример программы

Пусть есть класс, реализующий несколько математических функций.

```
public class CustomMath {  
    public static int sum(int x, int y){  
        return x+y; //возвращает результат сложения двух чисел  
    }  
    public static int division(int x, int y){  
        if (y == 0) {//если делитель равен нулю  
            throw new IllegalArgumentException("divider is 0 ");  
        } //бросается исключение  
        return x/y; //возвращает результат деления  
    }  
}
```

Наблюдение

- Иногда требуется снабжать программу модульными тестами.
- Тесты неудобно хранить в самой программе.
 - Усложняет чтение кода.
 - Такие тесты сложно запускать.
 - Тесты не относятся к бизнес-логике приложения и должны будут быть исключены из конечного продукта.
- Внешняя библиотека, подключенная к проекту может существенно облегчить разработку и поддержание модульных тестов.
 - Для каждого языка программирования реализация отличается.
 - CUnit, JUnit, ...

Вариант модульного тестирования №1

Некоторые проверки можно поместить в сам класс.

```
public class CustomMath {  
    public static void main(String[] args) {  
        if (sum(1, 3) == 4) { //проверяем, что при сложении 1 и 3  
                               //нам возвращается 4  
            System.out.println("Test1 passed.");  
        } else {System.out.println("Test1 failed.");}  
        try {  
            int z = division(1, 0);  
            System.out.println("Test3 failed.");  
        } catch (IllegalArgumentException e){  
            //тест считается успешным, если при попытке деления на 0  
            //генерируется ожидаемое исключение  
            System.out.println("Test3 passed.");  
        }  
    }  
}
```

Установка JUnit

- Сайт проекта:
 - <http://download.sourceforge.net/junit/>
- Настройка окружения:
 - <http://supportweb.cs.bham.ac.uk/docs/tutorials/docsystem/build/tutorials/winenvars/winenvarshome.html>
- Библиотека может быть подключена к любой Java программе.
- Модуль JUnit входит во многие популярные IDE.

JUnit – внешний вид приложения

Тесты можно запускать как из консоли, так и при помощи графического приложения.



Возможности JUnit

Тест, автоматически сгенерированный для метода сложения, выглядит так:

```
//подключение библиотеки
import junit.framework.*;
//Тест должен быть наследником класса TestCase
public class CustomMathTest extends TestCase {
    //тестирующие методы начинаются с префикса "test"
    public void testSum() {
        System.out.println("sum");
        int x = 0; int y = 0; int expResult = 0;
        int result = CustomMath.sum(x, y);
        //проверка условия на совпадение
        assertEquals(expResult, result);
        fail("The test case is a prototype."); }
}
```

Возможности JUnit (cont.)

В конце обычно происходит сравнение фактического результата с ожидаемым.

```
//тестирующие методы начинаются с "test"  
//Данный тест сгенерирован для метода деления  
public void testDivision() {  
    System.out.println("division");  
    int x = 0; int y = 0; int expResult = 0;  
    int result = CustomMath.division(x, y);  
    //проверка условия на совпадение  
    assertEquals(expResult, result);  
    fail("The test case is a prototype.");  
} }
```

Обработка исключений

Запуск теста для метода `division(x, y)` выдал ошибку:

```
Testcase: testDivision(simplemathapp.CustomMathTest):
```

```
Caused an ERROR divider is null
```

```
java.lang.IllegalArgumentException: divider is null
```

```
    at simplemathapp.CustomMath.division(CustomMath.java:42)
```

```
    at simplemathapp.CustomMathTest.testDivision(CustomMathTest.java:38)
```


Обработка исключений (cont.)

Причина в том, что тест некорректно обрабатывает деление на ноль:

```
public void testDivision() {  
    System.out.println("division");  
    int x = 0; int y = 0; int expResult = 0;  
    int result = CustomMath.division(x, y);  
    assertEquals(expResult, result);  
}
```

Вместо сравнения числовых результатов нужно ожидать исключение.

Обработка исключений (cont.)

- В случае исключения мы не делаем ничего.
- Если исключения нет – сообщаем об ошибке.

```
public void testDivision() {  
    int x = 0; int y = 0; int expResult = 0;  
    try {  
        expResult = CustomMath.division(x, y);  
        fail("Exception expected");  
    } catch (IllegalArgumentException e) {  
        System.out.println("pass");  
    }  
}
```

Версия JUnit 4.x доступна на сайте проекта:

- использует аннотации,
- отслеживает истечение времени,
- возможность определять методы инициализации тестовой сьюиты.

Ограничение JUnit

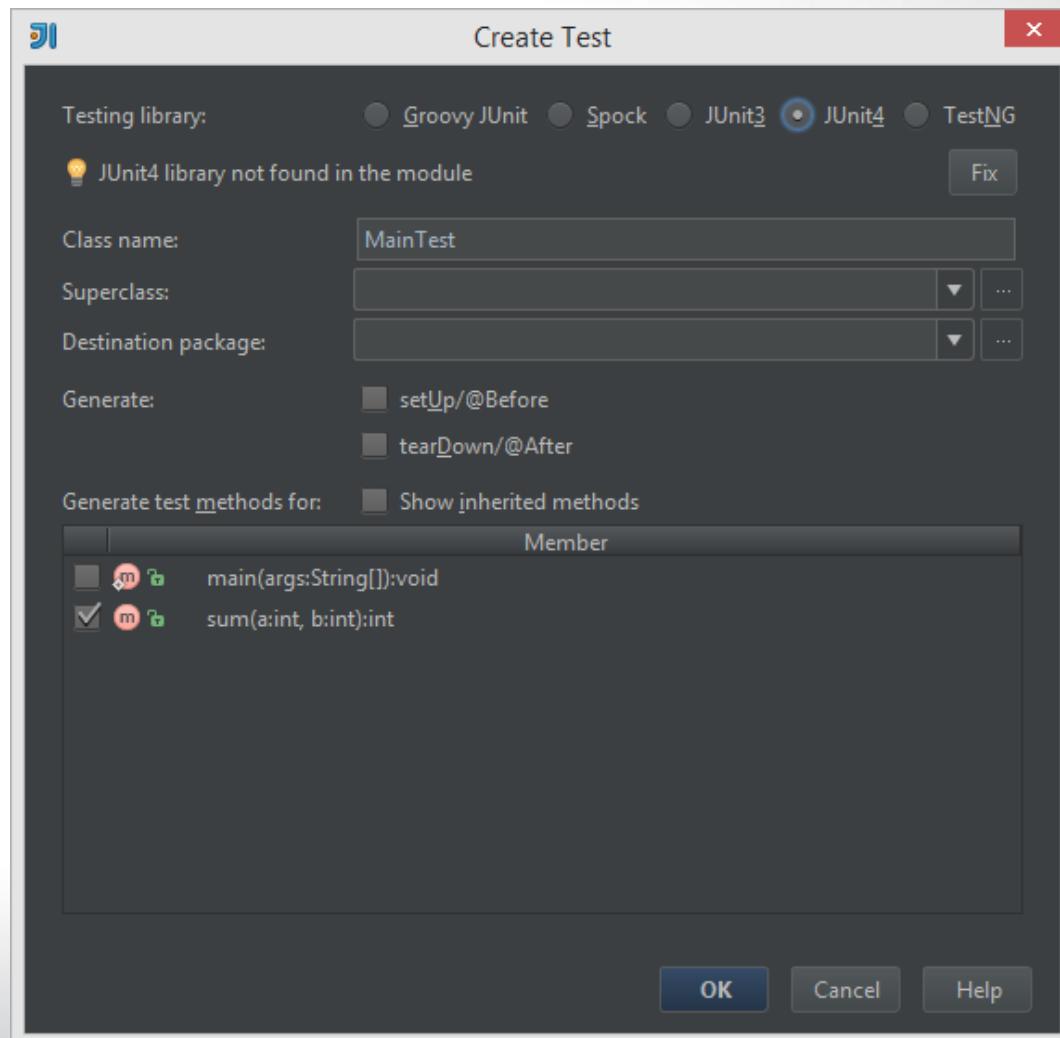
- Невозможность моделирования многопоточных ситуаций.
 - Запуск нескольких тестов одновременно не позволяет отследить момент их завершения.
 - Библиотеки JUnitPerf, GroboUtils обходят данное ограничение.

Выводы

- Наличие хорошего набора тестов важно для контроля качества программы.
- Разработка тестов может отнимать немного времени.
- Автоматизация тестов – приоритетная задача для успешного тестирования.

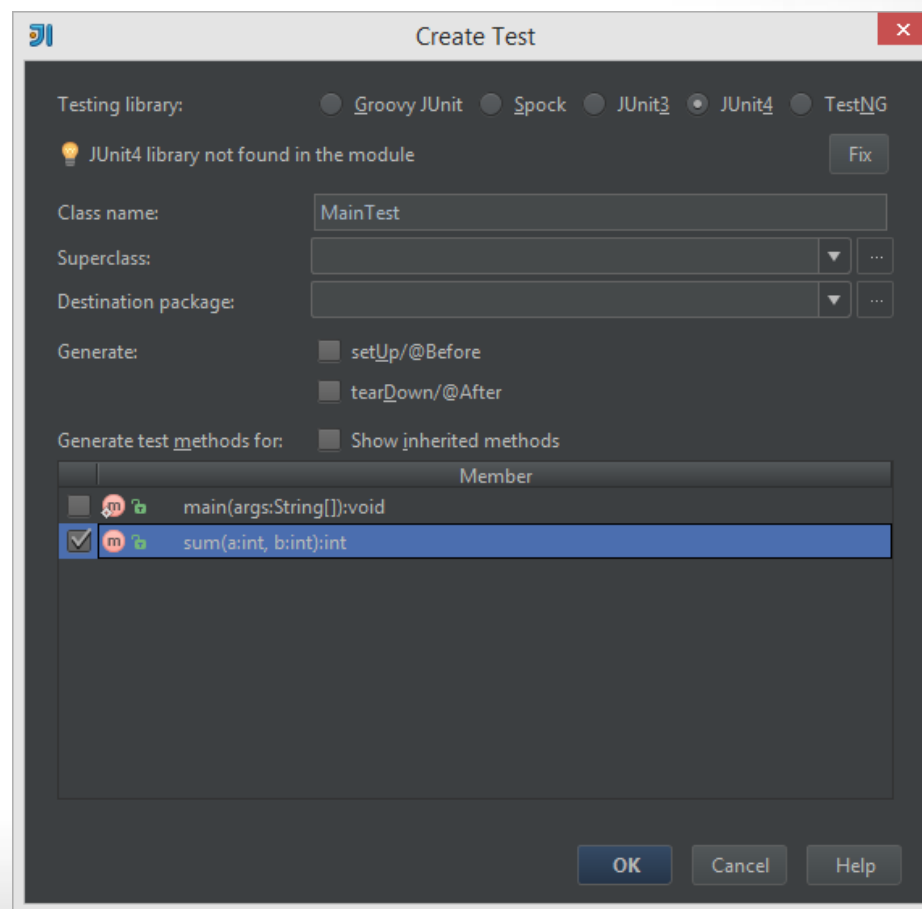
Автоматизация разработки тестов в IDEA

- Модуль JUnit встроен в среду разработки.
- Проект состоит из одного класса и двух публичных методов.
- Для создания тестового покрытия требуется несколько нажатий мыши.



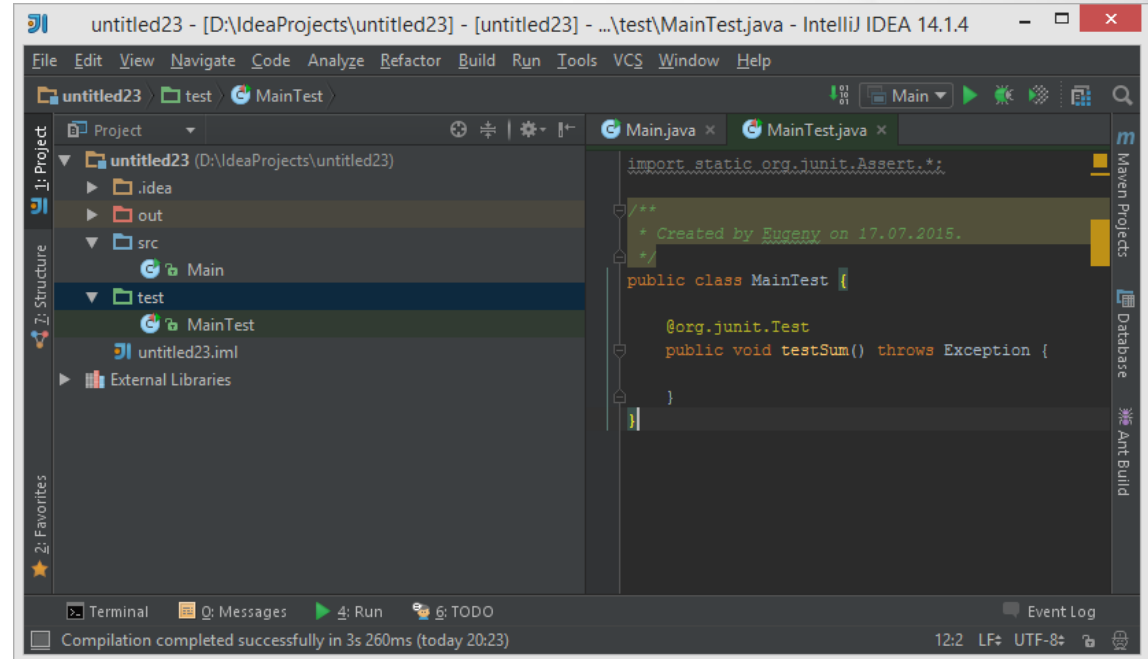
Генерація тестового покриття

Для классов проекта
автоматически создаются
тестовые случаи.



Созданные тесты

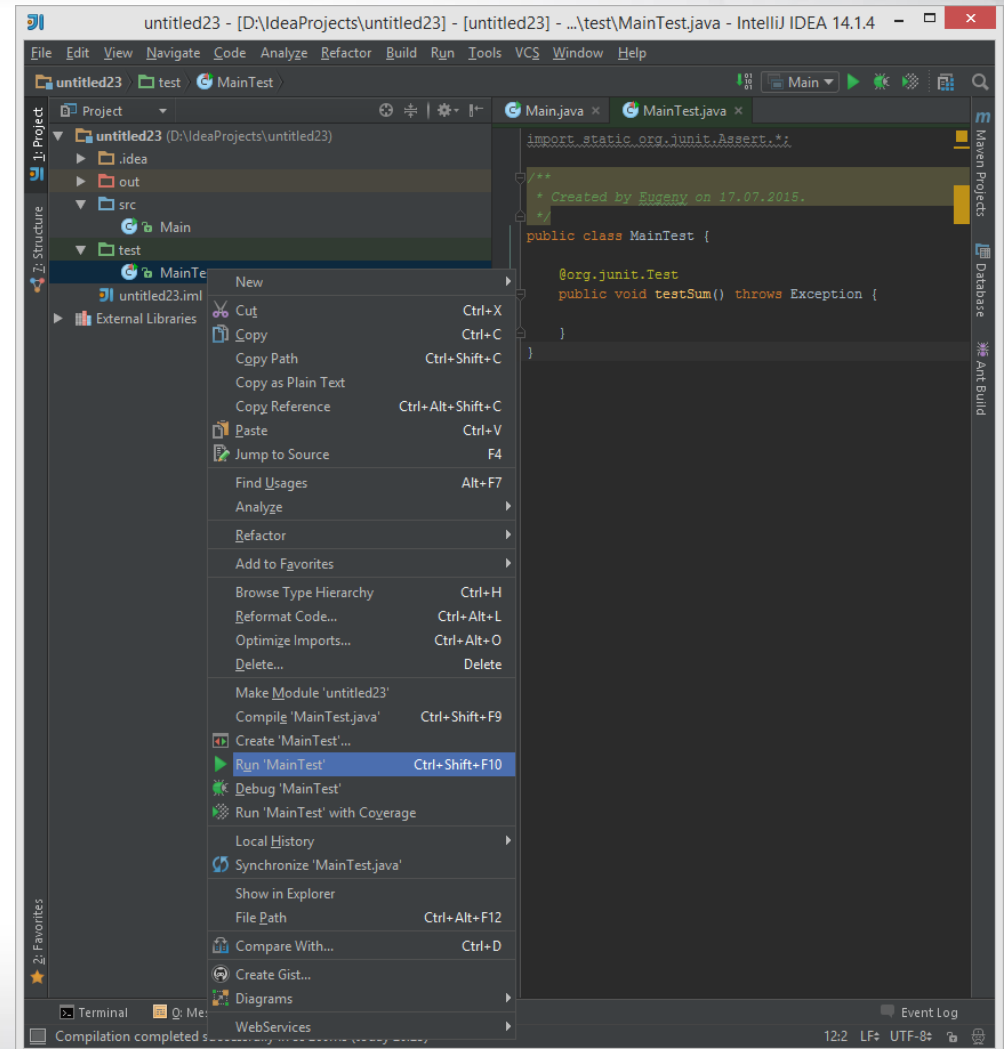
- Тесты помещаются в отдельную папку и являются частью проекта.
- Требуется некоторая доработка вновь созданных тестов.



Запуск тестов

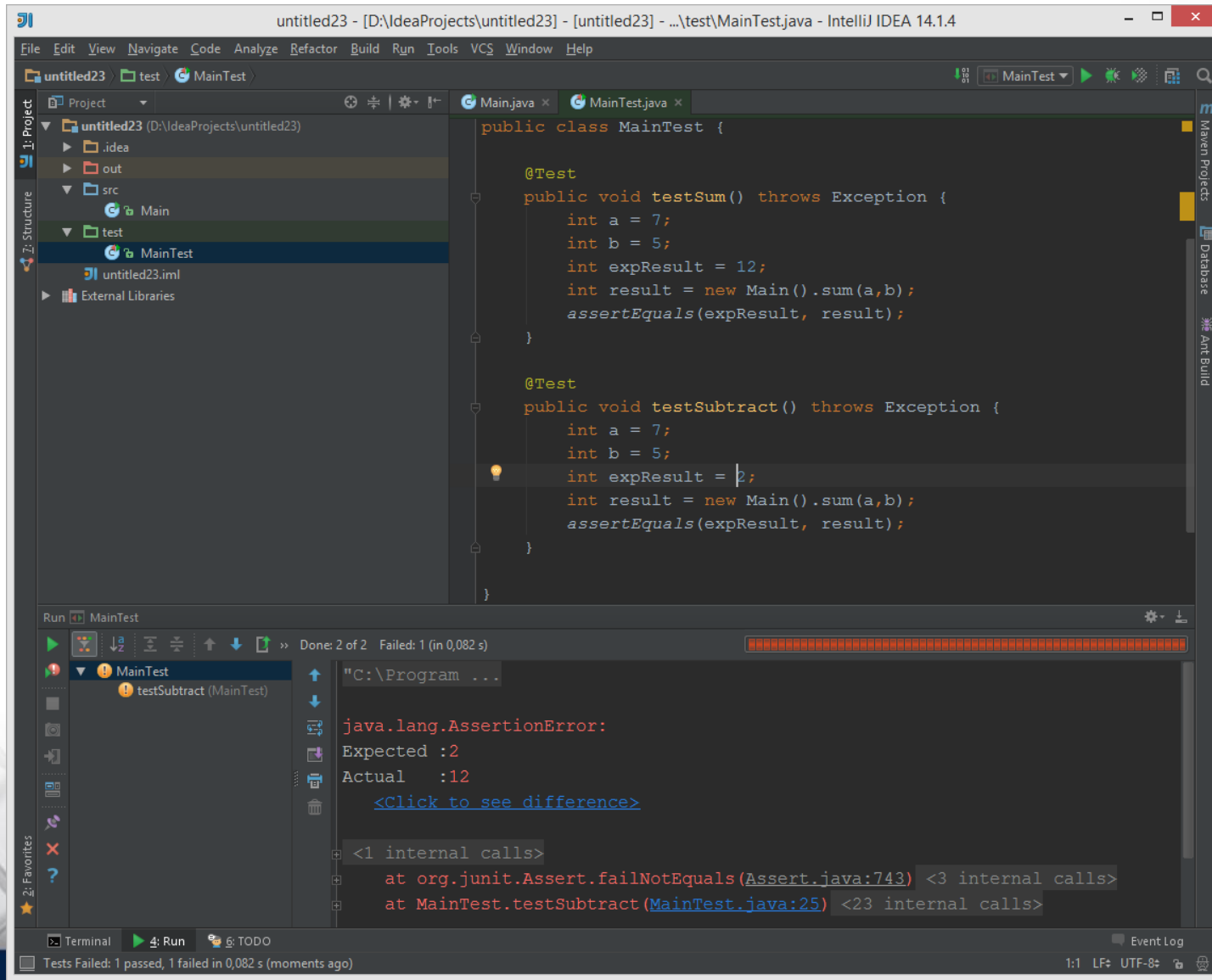
Тестовый класс
запускается так же,
как и обычный
класс с методом
`main()`.

Хоть у него и нет
такого метода



Результат работы тестов

- Представлен в виде дерева тестовых методов и классов.
- Один из двух тестов завершился с ошибкой.



The screenshot shows the IntelliJ IDEA 14.1.4 interface. The main editor displays the `MainTest.java` file with two test methods:

```
public class MainTest {  
  
    @Test  
    public void testSum() throws Exception {  
        int a = 7;  
        int b = 5;  
        int expectedResult = 12;  
        int result = new Main().sum(a,b);  
        assertEquals(expectedResult, result);  
    }  
  
    @Test  
    public void testSubtract() throws Exception {  
        int a = 7;  
        int b = 5;  
        int expectedResult = 2;  
        int result = new Main().sum(a,b);  
        assertEquals(expectedResult, result);  
    }  
}
```

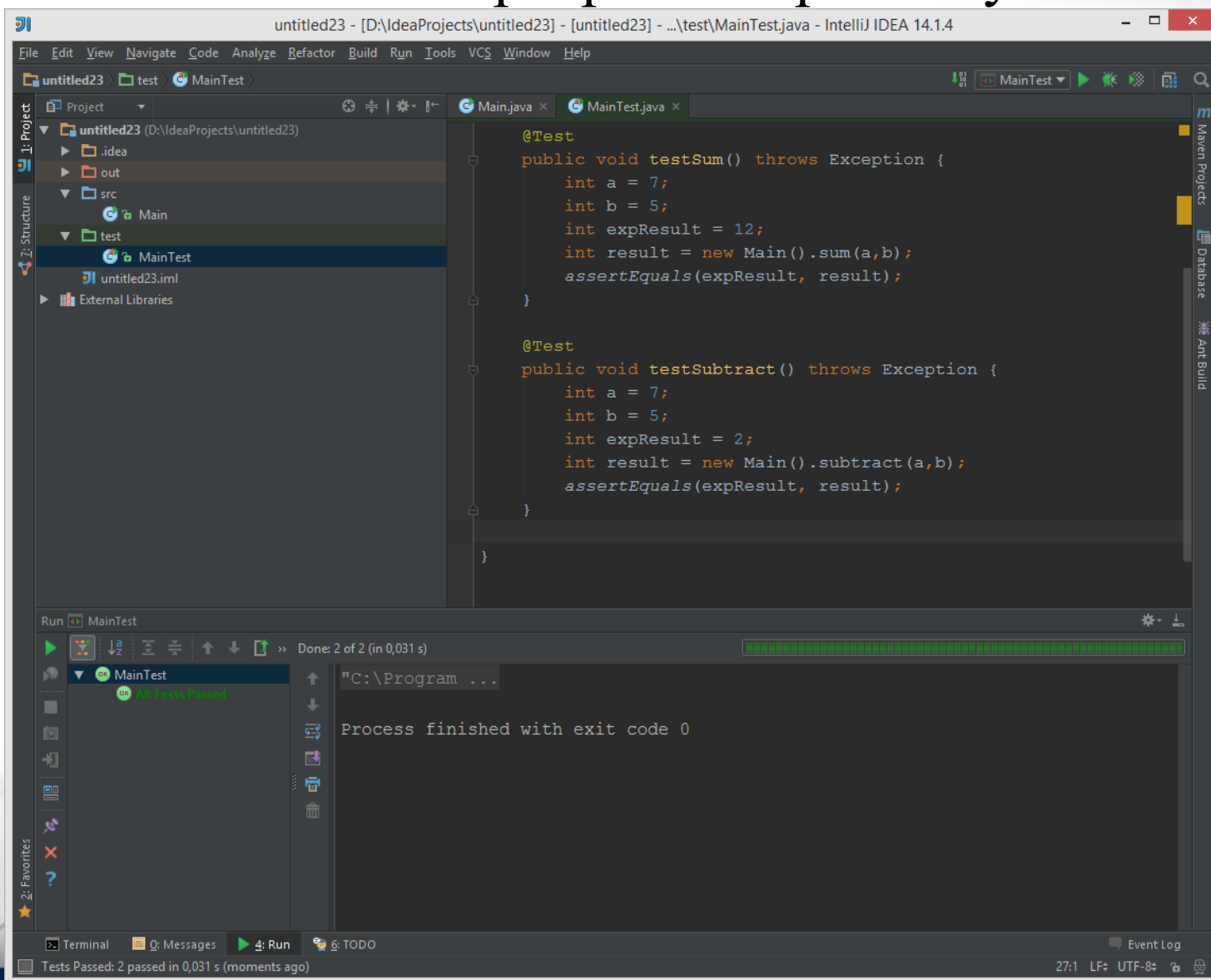
The left sidebar shows the project structure with `untitled23` and `MainTest` selected. The bottom panel shows the test results for `MainTest`:

- Done: 2 of 2
- Failed: 1 (in 0,082 s)
- Test `testSubtract (MainTest)` failed.
- Error message: `java.lang.AssertionError: Expected :2 Actual :12`
- Stack trace:
 - `<Click to see difference>`
 - `<1 internal calls>`
 - `at org.junit.Assert.failNotEquals(Assert.java:743) <3 internal calls>`
 - `at MainTest.testSubtract(MainTest.java:25) <23 internal calls>`

The status bar at the bottom indicates: `Tests Failed: 1 passed, 1 failed in 0,082 s (moments ago)`.

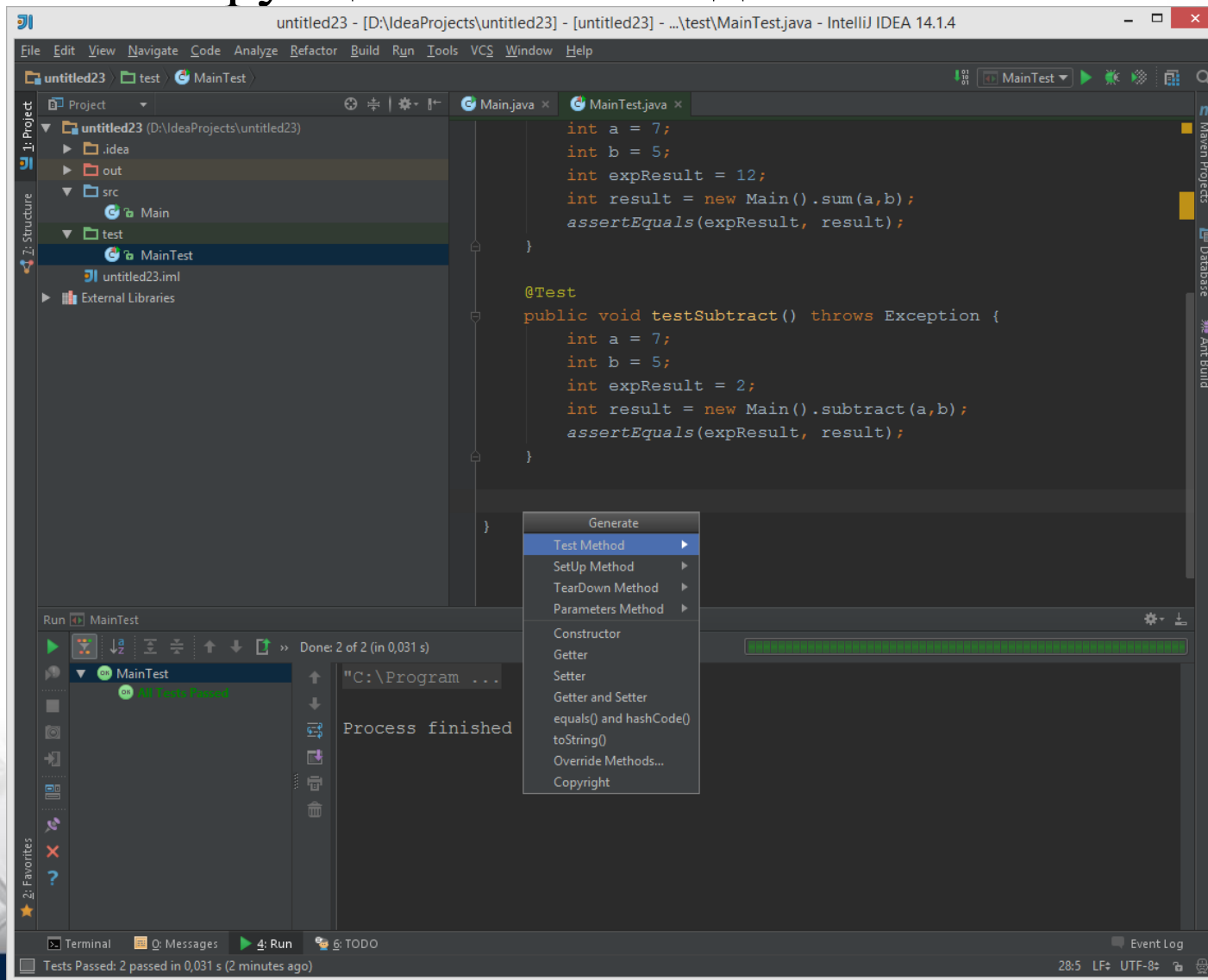
Повторный запуск тестов

Исправление ошибки в программе прошло успешно.



Расширение тестовой базы

При добавлении тестового случая нужно дополнить тест новыми инструкциями или создать новый тест.



Подход «Сначала тесты, затем код»

- Каждой строчке кода предшествует неработающий тест.
 - После написания кода тест должен пройти.
- Как только тест проходит, тестируемый код подвергается рефакторингу.
 - Затем вновь запускается тест.
- Разработка проходит маленькими итерациями, с чередованием написания кода и его тестирования.
- При интеграции кода с другими частями системы все тесты должны проходить успешно.



Преимущества подхода

- Каждая функция протестирована.
 - Если изменение «ломает» старую функциональность, падение тестов служит индикатором проблемы.
- Тесты документируют код, поскольку показывают, какие результаты и исключения следует ожидать от каждого модуля.
- Итерации разработки очень коротки (порядка 10 минут).
 - Несложно отследить какие изменения это повлекли.
- Дизайн приложения заметно упрощается.
 - Сложные методы сложно тестировать (и поддерживать).

Рекомендации к написанию тестов

- Название тестового метода.
- Размер теста.
- Ожидаемый результат.
- Тестовые данные.
- Исключения.

Название тестового метода

- Имя теста должно описывать:
 - Тестируемую функциональность.
 - Возможно, условия тестирования.

Непонятно	Понятно
test1	testAddUser
testAddUserThrowException	testAddUserWithoutPassword

Размер теста

- Тестовый метод должен быть коротким.
 - Иногда полезно выносить дополнительные проверки во вспомогательные методы, чтобы улучшить читаемость теста.
- Количество проверок (assert) должно быть минимальным.
 - Иначе по падению теста сложно будет найти причину ошибки.
- Каждый тест должен покрывать одну единицу бизнес-логики. Это может быть:
 - Простой метод.
 - Один из исходов конструкции if..else.
 - Один из случаев (case) блока switch.
 - Исключение, обрабатываемое блоком try...catch
 - Исключение, генерируемое (throw) в методе.

Ожидаемый результат

- Ожидаемый результат должен быть константой.
- Не следует в тесте повторять тестируемую логику, подсчитывая результат.

```
public void testBalance1() {  
    Account account = new account();  
    account.deposit(10);  
    account.withdraw(5);  
    account.deposit(6);  
    int expectedBalance = 11; //правильно  
    assertTrue(expectedBalance, account.getBalance());  
}  
public void testBalance2() {  
    Account account = new Account();  
    account.deposit(10);  
    account.withdraw(5);  
    account.deposit(6);  
    int expectedBalance = 10 - 5 + 6; //неправильно  
    assertTrue(expectedBalance, account.getBalance());  
}
```

Тестовые данные

- Тестовые данные и ожидаемый результат должны быть рядом.
- Если в приведенном примере объект `user` удобнее создавать вне тестового метода (н-р чтобы избежать дублирования кода), следует использовать именованные константы.

```
public void testIsPasswordValid() {  
    assertTrue(user.isPasswordValid("abcdef"));  
    // понять, правильно ли написан тест,  
    // можно лишь отыскав где создается объект user  
    assertFalse(user.isPasswordValid("123456"));  
}
```

Тестовые данные (cont.)

```
public void testIsPasswordValid() {  
    User user = new User("Name", "abcdef");  
    assertTrue(user.isPasswordValid("abcdef"));  
    //здесь все понятно  
    assertFalse(user.isPasswordValid("123456"));  
}
```

Тестовые данные (cont.)

```
public void testIsPasswordValid() {  
    assertTrue(user.isPasswordValid(CORRECT_PASSWORD));  
    // альтернативный вариант с использованием  
    // именованных констант  
    assertFalse(user.isPasswordValid(WRONG_PASSWORD));  
}
```

Исключения

- Если получение исключения не обозначает удачное завершение теста, рекомендуется пробрасывать его из тестового метода, а не обрабатывать блоком try...catch. Само исключение более информативно, чем сообщение.
- Не рекомендуется использовать общий класс (Exception), так как это затрудняет чтение и поддержку теста.

```
public void testRetrieveUser() {  
    try {  
        assertNotNull(manager.retrieveUser("name", "password"));  
    } catch (WrongPasswordException e) {  
        fail("Exception occurred: " + e.getMessage());  
        e.printStackTrace(); } }  
  
public void testRetrieveUser() throws WrongPasswordException {  
    assertNotNull(manager.retrieveUser("name", "password"));  
} //оптимальный вариант  
  
public void testRetrieveUser() throws Exception {  
    assertNotNull(manager.retrieveUser("name", "password")); } }
```

Ссылки

- Сайт проекта JUnit:
 - <http://junit.sourceforge.net>
- Использование фреймворков для тестирования
 - <https://www.jetbrains.com/idea/help/testing.html>

Спасибо!

Модульное тестирование при помощи JUnit

Евгений Беркунский, НУК
eugeny.berkunsky@gmail.com
<http://www.berkut.mk.ua>

