<epam> Delivering Excellence in Software Engineering

JUnit

Краткий обзор JUnit / JUnit 4

Определения

- **Юнит-тестирование (*unit testing*)** — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволит достаточно быстро проверить, не привело ли очередное изменение кода к *регрессии*, то есть к появлению ошибок в уже написанных и оттестированных местах программы, а также облегчает локализацию и устранение таких ошибок.

http://ru.wikipedia.org/wiki/Юнит-тестирование

- **Разработка через тестирование** - процесс разработки программного обеспечения, который предусматривает написание и автоматизацию модульных тестов еще до момента написания соответствующих классов или модулей. Это гарантирует, что все обязанности любого элемента программного обеспечения определяются еще до того, как они будут закодированы.

Цели

- **Поощрение изменений**
  - Юнит-тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

- **Упрощение интеграции**
  - Юнит-тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируются отдельные части программы, затем программа в целом.

- **Документирование кода**
  - Юнит-тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

Цели

- **Отделение интерфейса от реализации**
  - Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных. В ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с базой данных и реализует этот интерфейс, используя свой собственный mock-объект. Это приводит к менее связанному коду, минимизируя зависимости в системе.

- **Баг-трэкинг**
  - В случае обнаружения бага для него можно (даже рекомендуется) создать тест для выявления повторения подобной ошибочной ситуации при последующем изменении кода.

JUnit версии

- **JUnit**

Java < 1.5.0

Наследуем и расширяем классы

- **JUnit 4**

Java ≥ 1.5.0

Используем аннотации

JUnit – два соглашения

Первое соглашение заключается в том, что в JUnit неявным образом требуется, чтобы имя любого метода, предназначенного для функционирования в качестве логического теста, начиналось с префикса ***test***. Любой метод, имя которого начиналось с этого префикса, например, **testUserCreate**, исполняется в соответствии с хорошо описанным процессом тестирования, который гарантирует исполнение соответствующей фикстуры (fixture) как до, так и после этого тестового метода.

Второе соглашение, позволяющее среде JUnit распознавать нужный объект – класс, содержащий тесты – заключается в том, что сам этот класс должен являться расширением класса **TestCase** среды JUnit (или некоторым производным от него).

JUnit - пример

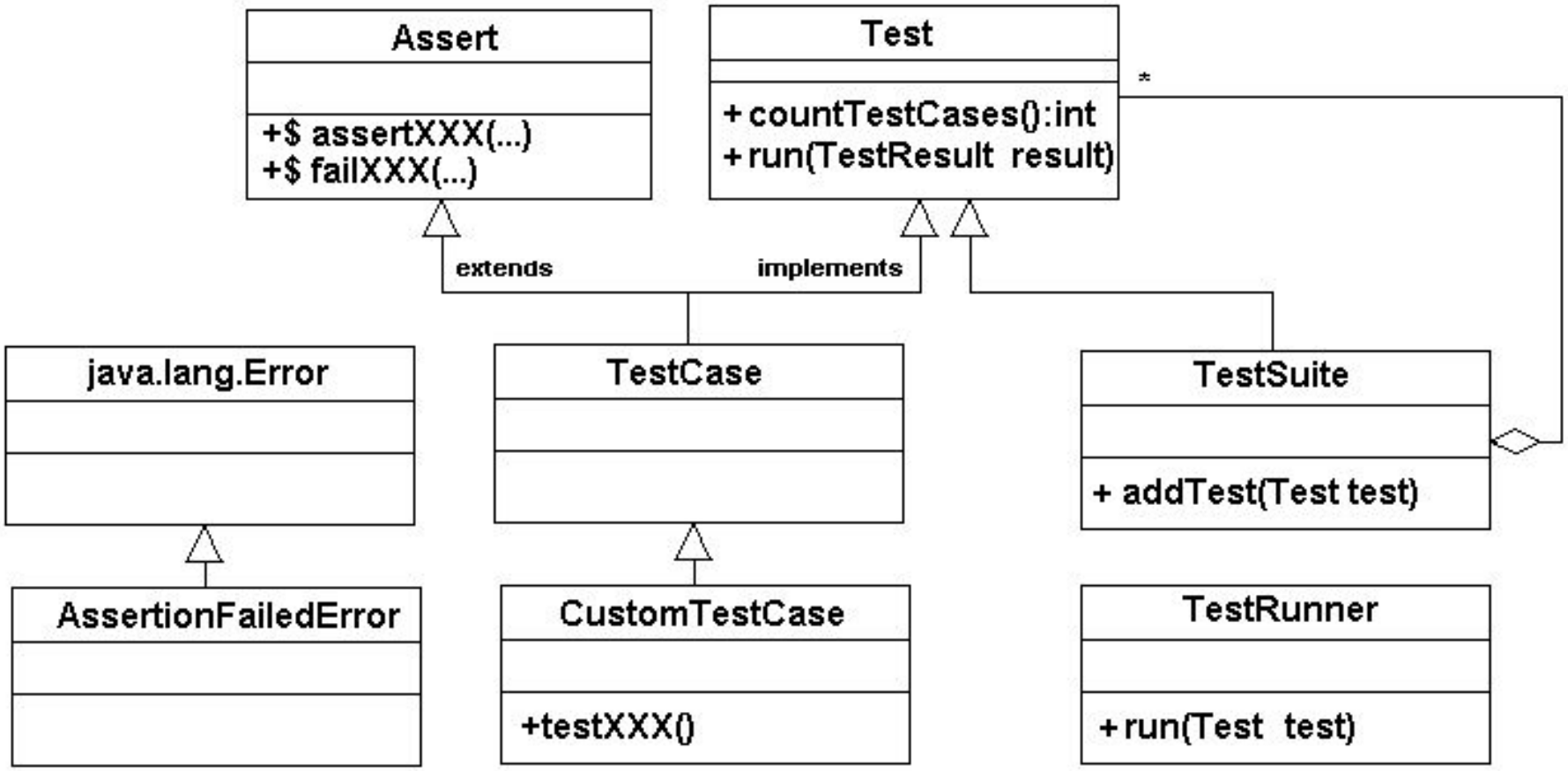
```
public class ClassToTest {
  static public int increment(int a) {
    return a++;
  }
}
```

```
import junit.framework.*;

public class TestClassToTest extends TestCase {
  public TestClassToTest (String name) {
    super(name);
  }
  public void testIncrement() {
        int result = ClassToTest.increment(2);
        assertEquals(result , 3);
  }
}
```

```
…
public static void main(String[] args) {
    TestRunner runner = new TestRunner();
    TestSuite suite = new TestSuite();
    suite.addTest(new TestClassToTest (“testIncrement”));
    runner.run(suite);
}
…
```

JUnit – UML



Есть несколько вариантов TestRunner-а

JUnit – assert…

Проверка (Assert) - метод класса ***TestCase***, который предназначен для сверки реального состояния тестируемого кода с ожидаемым.

java.lang.Object
| +--junit.framework.Assert

- **assertTrue**
- **assertFalse**
- **assertEquals**
- **assertNull**
- **assertNotNull**
- **assertSame**

java.lang.Object
| +--java.lang.Throwable
| +--java.lang.Error
| +--junit.framework.**AssertionFailedError**

http://junit.sourceforge.net/javadoc/junit/framework/TestCase.html
http://junit.sourceforge.net/javadoc/junit/framework/Assert.html
http://junit.sourceforge.net/javadoc/junit/framework/AssertionFailedError.html

JUnit – тестирование исключений

```
…
public class TestException extends TestCase {
…
  public void testException() throws Exception{
    try{
      unsafeCall(…);
      // Test Fail
      fail("No exception was thrown");
    }catch(OurException e){
      // Test OK
    }
  }
…
}
```

JUnit – фикстуры

Фикстура (Fixture) - состояние среды тестирования, которое требуется для успешного выполнения тестового метода. Это может быть набор каких-либо объектов, состояние базы данных, наличие определенных файлов и т.д. Фикстура создается в методе ***setUp()*** перед каждым вызовом метода вида ***testSomething*** теста (**TestCase**) и удаляется в ***tearDown()*** после окончания выполнения тестового метода.

```
public class TestClassToTest extends TestCase {

  // will run before test execution
  protected void setUp() throws Exception {
    …
  }

  // will run after test execution
  protected void tearDown() throws Exception {
    …
  }
}
```

JUnit - исполнение

- **Прямой Java вызов**
  - *TextUI*: Предоставляет текстовый вывод на консоль.
  - *AwtUI*: Предоставляет GUI вывод с использованием AWT (Abstract Window Toolkit).
  - *SwingUI*: Предоставляет GUI вывод с использованием Swing.

- **ANT**

```
<junit printsummary="yes" haltonfailure="yes">
        <classpath>
                <pathelement location="${build.tests}"/>
                <pathelement path="${java.class.path}"/>
        </classpath>
        <formatter type="plain"/>
        <test name="my.test.TestCase" haltonfailure="no" outfile="result">
                <formatter type="xml"/>
        </test>
        <batchtest fork="yes" todir="${reports.tests}">
                <fileset dir="${src.tests}">
                        <include name="**/*Test*.java"/>
                        <exclude name="**/AllTests.java"/>
                </fileset>
        </batchtest>
</junit>
```

- **IDE**
  - Eclipse
  - Clover
  - etc.

JUnit 4

В JUnit 4 за счет использования аннотаций Java 5 удалось полностью отказаться обоих вышеуказанных соглашений (см. слайд «*JUnit - два соглашения*»).
Отпадает необходимость в иерархии классов, а методы, предназначенные для функционирования в качестве тестов, достаточно промаркировать новой аннотацией:
@Test

JUnit 4 отказывается от понятия *«ошибка».* В то время как предшествующие версии JUnit сообщали и о количестве *неудач*, и о количестве *ошибок*, в версии JUnit 4 тест или проходит *успешно*, или завершается *неудачей*.

JUnit 4 – объявление теста

Мы не наследуемся от ***TestCase***.
Префикс ***test*** в имени тестового метода заменяет ***аннотация***.

```
import junit.framework.TestCase;
import org.junit.Test;

public class TestClassToTest extends TestCase {

    @Test
    public void testincrement() {
        ....
    }
}
```

JUnit 4 – assert…

Мы больше не наследуемся от *TestCase*, однако нам все еще нужны методы *assert…()*. В этом случае мы пользуемся нововведением Java 5 – статическими импортами.

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

public class TestClassToTest {

    @Test
    public void increment() {
        ...
        assertEquals( result, 3 );
    }
}
```

JUnit 4 – фикстуры

Методы ***setUp*** и ***tearDown*** упразднены. Теперь необходимые для инициализации и освобождения ресурсов методы мы маркируем помощью аннотаций ***@Before*** или ***@After***.

Мы можем промаркировать несколько методов как ***@Before*** или ***@After.*** Порядок их вызова может быть любой – какой, решает среда исполнения.

```
public class TestClassToTest {

    @Before
    public void prepareTestData() { ... }

    @Before
    public void setupConnection() { ... }

    @After
    public void freeConnection() { ... }
}
```

JUnit 4 – тестирование исключений

В JUnit 4 нам не надо использовать блоки try-catch. Нам нужно лишь объявить ожидаемое исключение в аннотации ***@Test***

```
public class TestClassToTest {

    @Test(expected=OurException.class)
    public void testException() {
        unsafeCall(…);
    }
}
```

JUnit 4 - timeout

Выполнение некоторых unit-тестов может занимать больше времени, чем у нас есть (например тест требует соединения с внешним асинхронным ресурсом).
Все что нужно сделать – это указать параметр ***timeout*** с необходимым значением в аннотации ***@Test***.

```
@Test(timeout=5000)
public void increment() {
    ...
}
```

Если максимальное отведенное тесту время истекает, то мы получаем понятное сообщение об ошибке и о не выполнении теста:

java.lang.Exception: test timed out after 5000 milliseconds

JUnit 4 – игнорирование тестов

В некоторых ситуациях может понадобиться отключить некоторые тесты.

```
public class TestClassToTest {

    @Ignore("Not running because <reason here>")
    @Test
    public void increment() {
      …
    }
}
```

JUnit 4 – наборы тестов

```
// Old JUnit style
public class AllTests extends TestCase {

    public static Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTestSuite(TestClassToTest1.class);
        suite.addTestSuite(TestClassToTest2.class);
        return suite;
    }
```

```
@RunWith(value=Suite.class)
@SuiteClasses(value={TestClassToTest1.class, TestClassToTest2.class})
public class AllTests {
  ...
}
```

JUnit 4 – параметризованные тесты

```
@RunWith(value=Parameterized.class)
public class TestClassToTest {
    private int  expected;
    private int value;

    @Parameters
    public static Collection data() {
        return Arrays.asList( new Object[][] {
                        { 3, 2 },   // expected, value
                        { 2, 3 }
                        });
    }

    public TestClassToTest (int expected, int value) {
        this.expected = expected;
        this.value = value;
    }

    @Test
    public void increment() {
        assertEquals(expected, ClassToTest.increment(value));
    }
}
```

Полезные ссылки

- http://junit.sourceforge.net/doc/testinfected/testing.htm
- http://junit.sourceforge.net/doc/faq/faq.htm
- http://junit.sourceforge.net/javadoc/index.html
- http://www.javaworld.com/javaworld/jw-12-2000/jw-1221-junit.html
- http://www.ibm.com/developerworks/ru/edu/j-junit4/index.html
- http://ru.wikipedia.org/wiki/Юнит-тестирование
- http://litvinyuk.com/articles/junit.htm
- http://wiki.agiledev.ru/doku.php?id=tdd:glossary

Delivering Excellence in Software Engineering

JUnit

For more information, please contact:
Yauhen Peshkur
Team Leader
EPAM Systems, Inc.
Email: yauhen_peshkur@epam.com
http://www.epam.com