Seminar: JUnit

TH2013-LTUD Java-NVKhiet-HTThanh

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Nội dung

- Unit test
- JUnit

UNIT TEST

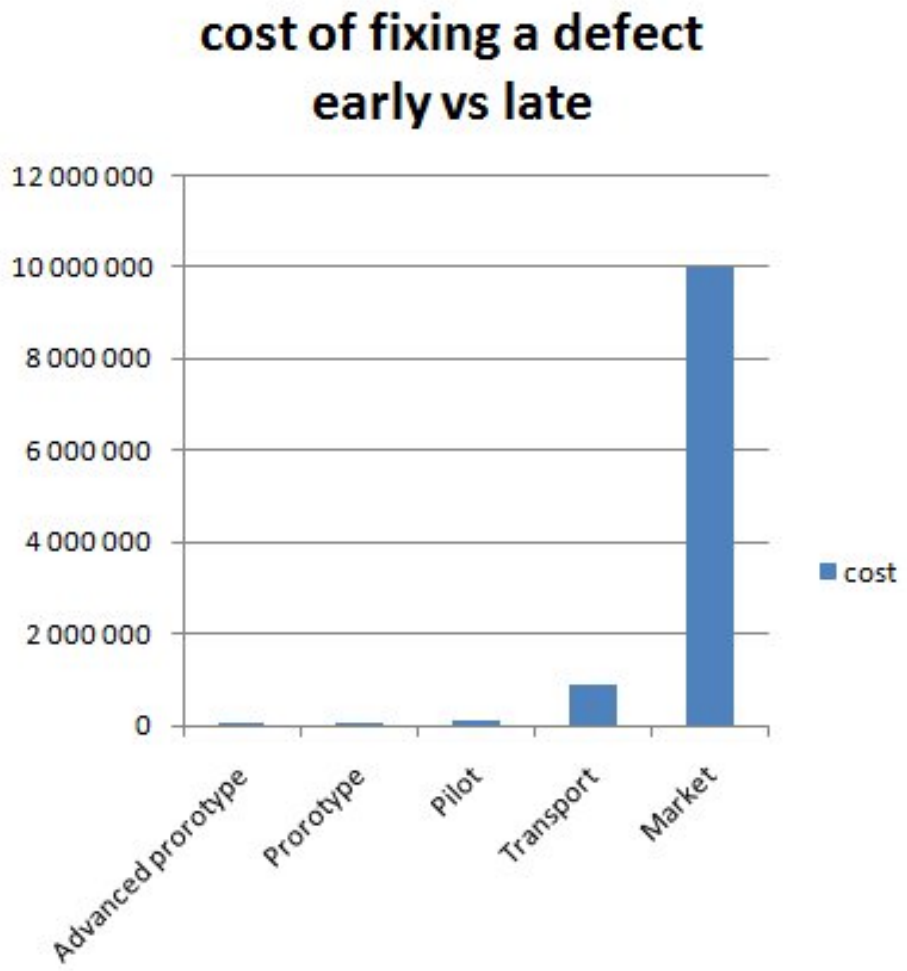
Example of Toyota's Prius

- A defect found in the production phase is about 50 times more expensive than if it is found during prototyping.
- If the defect is found after production it will be about 1,000 – 10,000 times more expensive!

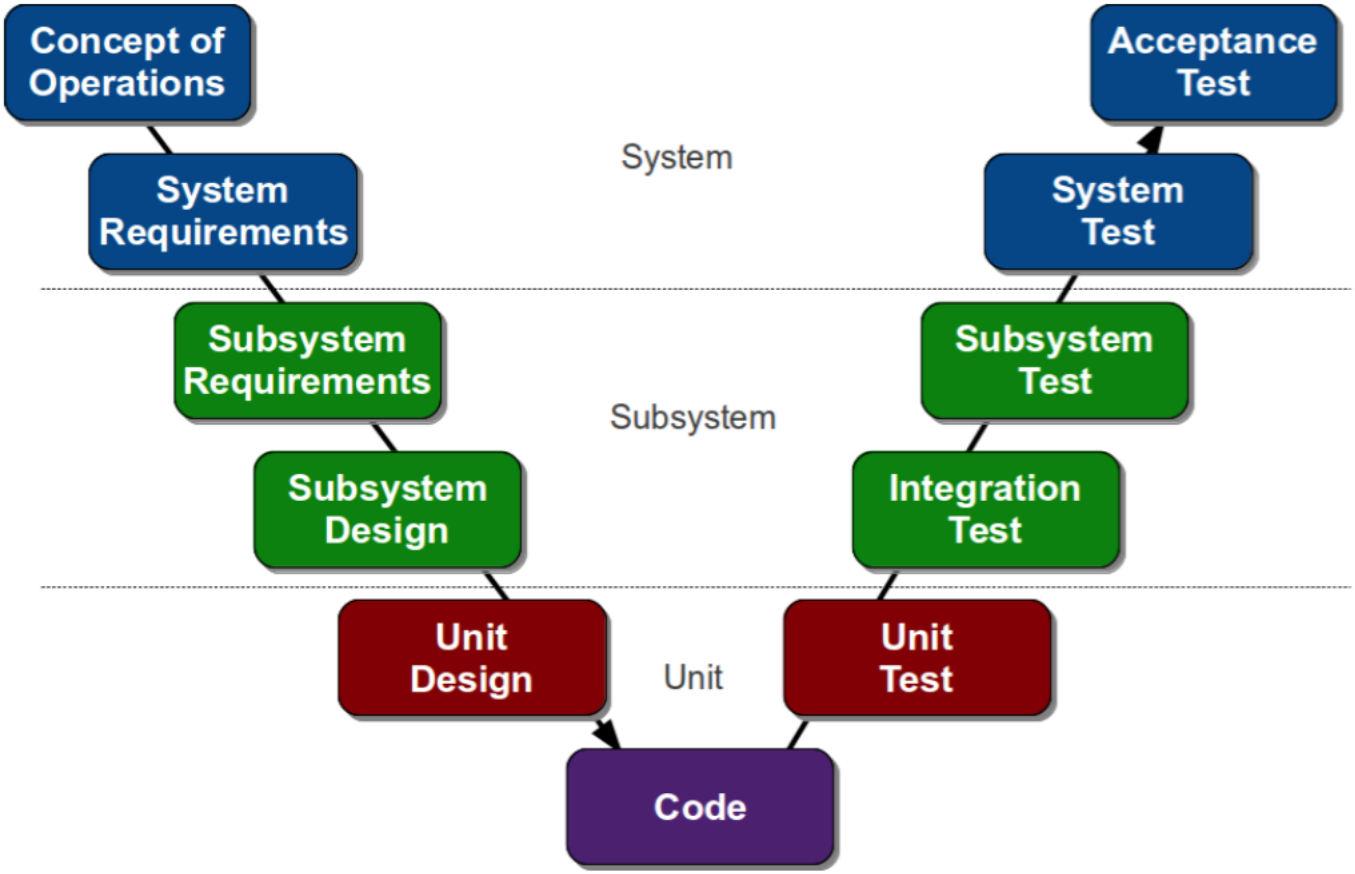
Example of Toyota's Prius

- *Reality turned out to be worse, a lot worse!*
- As Toyota's current problems with the Prius braking systems is costing them over $2 000 000 000 (2 billion!) to fix because of all the recalls and lost sales.
- "Toyota announced that a glitch with the software program that controls the vehicle's anti-lock braking system was to blame".

Cost of fixing a bug



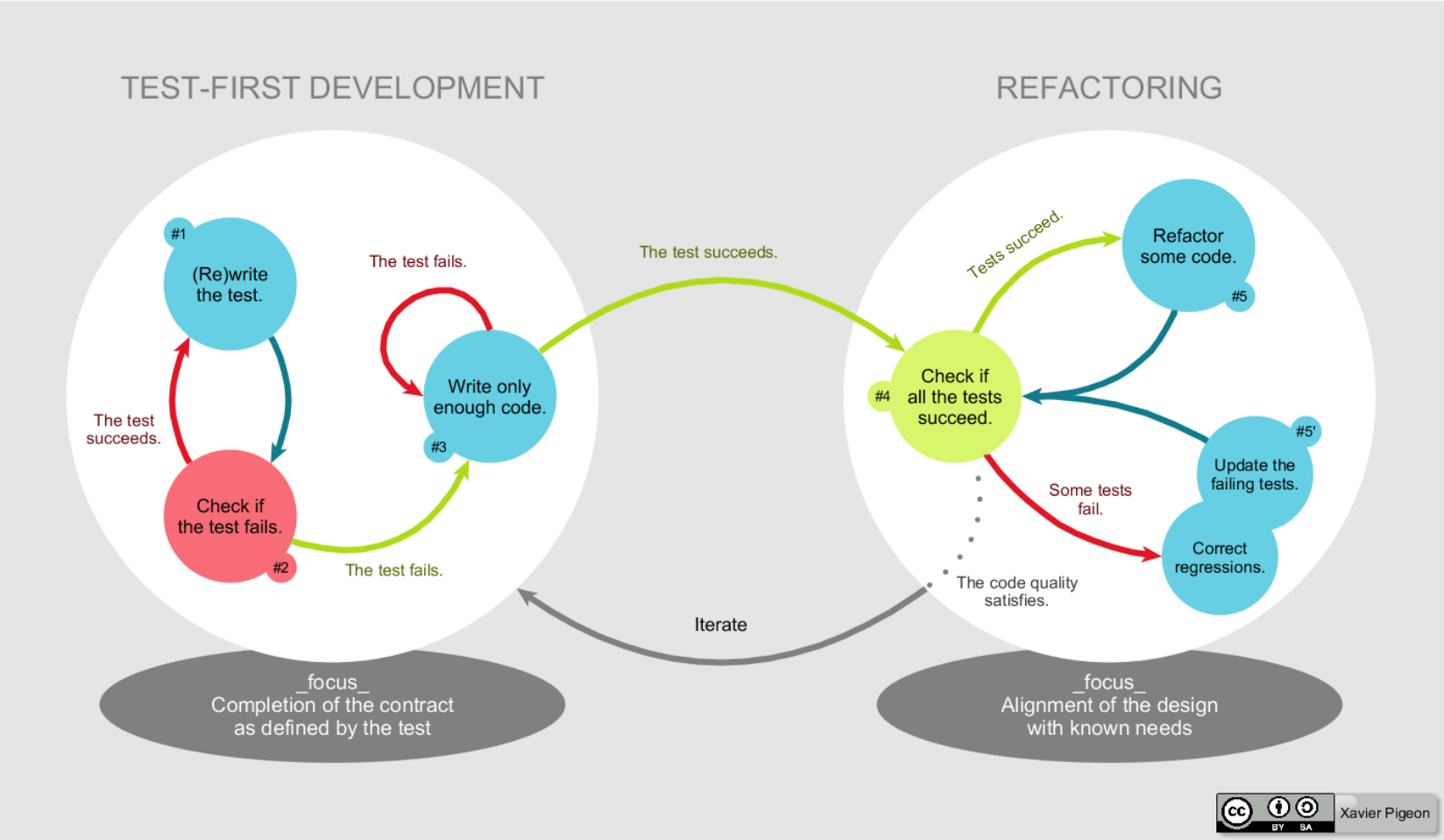
Test level


Concept of Operations
System Requirements
Subsystem Requirements
Subsystem Design
Unit Design
Code
Unit Test
Integration Test
Subsystem Test
System Test
Acceptance Test
System
Subsystem
Unit

Test-driven development (TDD)

- Một hướng tiếp cận trong quá trình viết source code:
  - Đầu tiên, viết test case
  - Sau đó, viết code để pass test case đã viết

TDD

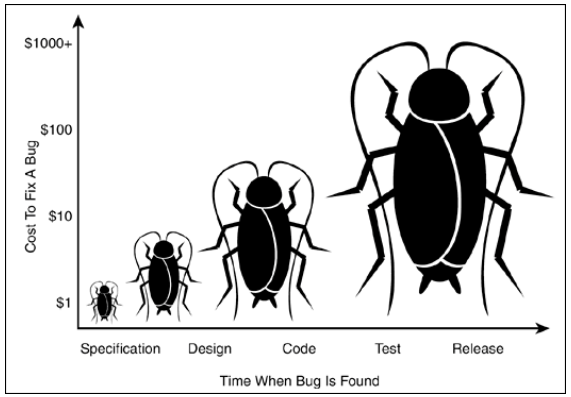

TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
The test succeeds.
Check if the test fails.
#2
The test fails.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
focus
Completion of the contract as defined by the test
focus
Alignment of the design with known needs
Xavier Pigeon

Unit test là gì?

- Chia chương trình (program) ra thành các đơn vị nhỏ nhất (unit)
  - Unit: function, class
- Test từng unit, với các input khác nhau, đảm bảo các unit này chạy đúng (correct)
- Write code (test method) to test code
- Do developer thực hiện, test trên chính các đoạn code mà mình viết ra
  - Không phải là công việc của tester, customer
  - Unit test là white box test

Tại sao cần unit test

- Phản hồi tức thời
  - Sớm phát hiện bug và chỉnh sửa
  - Càng phát hiện bug sớm, càng giảm chi phí sửa bug
  - Đảm bảo các unit ko bị lỗi trước khi tích hợp (integration test)



Tại sao cần unit test

- Giúp code của developer tốt hơn, sạch hơn, đẹp hơn
  - Code khó unit test → khó để sử dụng, khó để bảo trì, khó để nâng cấp, chỉnh sửa về sau.

Tại sao cần unit test

- Regression checker
  - Khi code có thay đổi, các unit test case sẽ được ***thực thi tự động***, đảm bảo code mới ko gây ra lỗi

Tại sao cần unit test

- Tiết kiệm chi phí
  - Reusable
  - Repeatable
  - Automated
  - Write once, run forever

Qui trình thực hiện unit test

- Đề bài:
  - Trong chương trình tính chu vi, diện tích tam giác, chúng ta có các class, các hàm sau:
    - class Diem: default constructor, constructor 1 tham số, constructor 2 tham số, hàm tính khoảng cách 2 điểm…
    - class TamGiac: default constructor, constructor 3 tham số, hàm tính chu vi, hàm tính diện tích...
- Câu hỏi:
  - Làm sao đảm bảo các hàm trên chạy đúng

Qui trình thực hiện unit test

- Giải pháp:
  - Tiến hành unit test các hàm
- Nhắc lại:
  - Unit test: write code to test code
  - Viết vài hàm, để test cho 1 hàm
- Qui trình:
  - Viết các hàm (test case, test method) để test 1 hàm
  - Sử dụng unit test framework (NUnit, JUnit…) để chạy các test method
  - Ghi nhận kết quả của test method (passed, failed, ignored)
  - Sửa lỗi ở những hàm mà test method của nó là failed
    - VD: test method thứ 3 của hàm tính chu vi failed → code của hàm tính chu vi sai → sửa lại

Làm sao để viết 1 test method

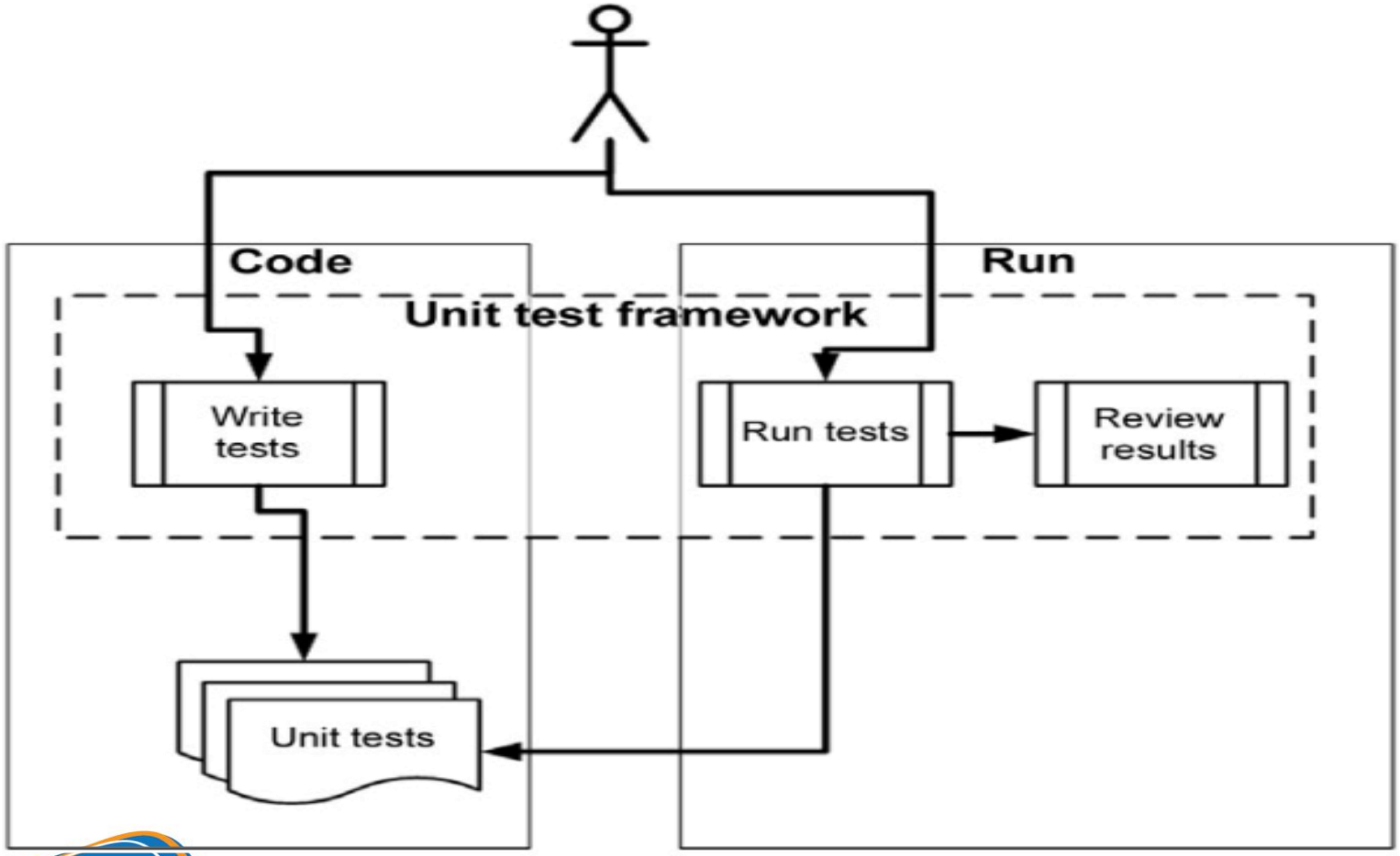
- Test method 1: tính chu vi tam giác vuông
  - Khởi tạo tam giác có tọa độ A(0,0), B(0,3) và C(4,0)
  - Kết quả mong đợi: expected=3+4+5=12
  - Kết quả thực tế: actual=gọi hàm tính chu vi
  - Assert để kiểm tra kết quả thực tế có giống kết quả mong đợi ko
- Test method 2: tính chu vi tam giác vuông cân
  - Khởi tạo tam giác có tọa độ A(0,0), B(0,1) và C(1,0)
  - Kết quả mong đợi: expected=1+1+1.4=3.4
  - Kết quả thực tế: actual=gọi hàm tính chu vi
  - Assert để kiểm tra kết quả thực tế có giống kết quả mong đợi ko

Unit test vs debug

- Tại sao ko debug code và tìm lỗi
- Whenever you modify code, you can rerun the test cases to verify you are still getting same answer

JUNIT

Unit test framework



Unit testing framework

- Thư viện hỗ trợ - Script
- Dữ liệu - Data driven
- Thực thi - Run
- Thống kê - Report
- Các framework hỗ trợ cho từng ngôn ngữ lập trình
  - **JUnit - Java**
  - NUnit - .NET
  - CppUnit - C++
  - PyUnit - Python
  - …

Sử dụng JUnit

- Add library
  - mvnrepository → junit4 → pom.xml
- Tạo JUnit test case
  - Eclipse → File → New → JUnit test case
- Thêm annotation @Test vào trước mỗi test method
- Gọi hàm cần test, dùng assertThat để kiểm tra expected vs actual
  - `import static org.junit.Assert.*;`
  - `import org.junit.*;`
  - `import static.org.hamcrest.CoreMatcher.*;`
- Thực thi test method: R-Click → Run → JUnit → xem kết quả ở cửa sổ JUnit

Ví dụ

- Cần unit test hàm Calculator.evaluate();

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

Ví dụ

- Add lib junit4
- Tạo class CalculatorTest
- Tạo test method
- Passed vs Failed???

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

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

Ví dụ test case failed

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum -= Integer.valueOf(summand);
    return sum;
  }
}
```

assertXXX()

```java
@Test
public void testAssertArrayEquals() {
  byte[] expected = "trial".getBytes();
  byte[] actual = "trial".getBytes();
  assertArrayEquals("failure - byte arrays not same", expected, actual);
}
```

```java
@Test
public void testAssertEquals() {
  assertEquals("failure - strings are not equal", "text", "text");
}
```

```java
@Test
public void testAssertFalse() {
  assertFalse("failure - should be false", false);
}
```

assertXXX

```java
@Test
public void testAssertNotNull() {
  assertNotNull("should not be null", new Object());
}
```

```java
@Test
public void testAssertNotSame() {
  assertNotSame("should not be same Object", new Object(), new Object());
}
```

```java
@Test
public void testAssertSame() {
  Integer aNumber = Integer.valueOf(768);
  assertSame("should be same", aNumber, aNumber);
}
```

```java
@Test
public void testAssertNull() {
  assertNull("should be null", null);
}
```

assertThat(value,matcher)

```
@Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertThat(sum, is(equalTo(6));
  }
```

```
assertThat(num, is(12));
assertThat(num, is(equalTo(12));
```

```
assertThat(someString, is(equalTo("blah"));
assertThat(someString, equalTo("blah");

assertThat(someObject, is(nullValue()));
assertThat(someObject, nullValue());
```

assertThat(value,matcher)

```
assertThat(foo, is(equalTo(bar)));
```

```
assertThat(foo, is(true));
```

```
assertThat(str, containsString(substr));
```

```
@Test
public void testAssertThatHasItems() {
  assertThat(Arrays.asList("one", "two", "three"), hasItems("one", "three"));
}
```

```
@Test
public void testAssertThatBothContainsString() {
  assertThat("albumen", both(containsString("a")).and(containsString("b")));
}
```

```
@Test
  public void testAssertThatEveryItemContainsString() {
    assertThat(Arrays.asList(new String[] { "fun", "ban", "net" }), everyItem(containsString("n")));
  }
```

assertThat(value,matcher)

```
@Test
public void testAssertThatHamcrestCoreMatchers() {
  assertThat("good", allOf(equalTo("good"), startsWith("good")));
  assertThat("good", not(allOf(equalTo("bad"), equalTo("good"))));
  assertThat("good", anyOf(equalTo("bad"), equalTo("good")));
  assertThat(7, not(CombinableMatcher.<Integer> either(equalTo(3)).or(equalTo(4))));
  assertThat(new Object(), not(sameInstance(new Object())));
}
```

```
@Test
  public void testReverse() {
    assertThat("oof", is(equalTo(StringUtils.reverseString("foo"))));
    assertThat("rab", is(equalTo(StringUtils.reverseString("bar"))));
 }
```

assertThat(value,matcher)

```
@Test
public void testPalindromes() {
  String[] matches =
    { "a", "aba", "Aba", "abba", "AbBa",
      "abcdeffedcba", "abcdEffedcba" };
  String[] misMatches =
    { "ax", "axba", "Axba", "abbax", "xAbBa",
      "abcdeffedcdax", "axbcdEffedcda" };
  for(String s: matches) {
    assertThat(StringUtils.isPalindrome(s), is(true));
  }
  for(String s: misMatches) {
    assertThat(StringUtils.isPalindrome(s), is(false));
  }
}
```

`@Test` annotation

- Đặt trước 1 hàm public, void
- JUnit xem hàm này là 1 test method (test case)

Expect

- Test case phải throw 1 exception xác định

```
@Test(expected = IndexOutOfBoundsException.class)
public void empty() {
     new ArrayList<Object>().get(0);
}
```

```
@Test
public void example1() {
    try {
        find("something");
        fail();
    } catch (NotFoundException e) {
        assertThat(e.getMessage(), containsString("could not find something"));
    }
    // ... could have more assertions here
}
```

Timeout

- Test case fail nếu chạy quá thời gian qui định

```
@Test(timeout=1000)
public void testWithTimeout() {
   ...
}
```

```
public class HasGlobalTimeout {
    public static String log;
    private final CountDownLatch latch = new CountDownLatch(1);

    @Rule
    public Timeout globalTimeout = Timeout.seconds(10); // 10 seconds max per method tested

    @Test
    public void testSleepForTooLong() throws Exception {
        log += "ran1";
        TimeUnit.SECONDS.sleep(100); // sleep for 100 seconds
    }

    @Test
    public void testBlockForever() throws Exception {
        log += "ran2";
        latch.await(); // will block
    }
}
```

@Before, @After

- Đặt trước 1 hàm public, void
- Có tác dụng "chuẩn bị" trước khi chạy 1 test method và "dọn dẹp" sau khi 1 test method chạy xong
  - Hàm @Before chạy trước mỗi test method
  - Hàm @After chạy sau khi 1 test method chạy xong
    - Nếu hàm @Before hay test method throw exception, hàm @After vẫn sẽ chạy

@BeforeClass, @AfterClass

- Hàm @BeforeClass chạy trước khi tất cả các test method trong class chạy
- Hàm @AfterClass chạy sau khi tất cả các test method trong class chạy xong

Ví dụ

```
private Collection collection;

@BeforeClass
public static void oneTimeSetUp() {
    // one-time initialization code
    System.out.println("@BeforeClass - oneTimeSetUp");
}

@AfterClass
public static void oneTimeTearDown() {
    // one-time cleanup code
    System.out.println("@AfterClass - oneTimeTearDown");
}
```

```
@Before
public void setUp() {
    collection = new ArrayList();
    System.out.println("@Before - setUp");
}

@After
public void tearDown() {
    collection.clear();
    System.out.println("@After - tearDown");
}
```

```
@Test
public void testEmptyCollection() {
    assertTrue(collection.isEmpty());
    System.out.println("@Test - testEmptyCollection");
}

@Test
public void testOneItemCollection() {
    collection.add("itemA");
    assertEquals(1, collection.size());
    System.out.println("@Test - testOneItemCollection");
}
```

```
@BeforeClass - oneTimeSetUp
@Before - setUp
@Test - testEmptyCollection
@After - tearDown
@Before - setUp
@Test - testOneItemCollection
@After - tearDown
@AfterClass - oneTimeTearDown
```

@Ignore

- Đánh dấu 1 test method sẽ được bỏ qua, ko cần chạy

```
@Ignore("Not Ready to Run")
@Test
public void divisionWithException() {
  System.out.println("Method is not ready yet");
}
```

@Parameterized

```
public class Fibonacci {
    public static int compute(int n) {
        int result = 0;

        if (n <= 1) {
            result = n;
        } else {
            result = compute(n - 1) + compute(n - 2);
        }

        return result;
    }
}
```

@Parameterized

```
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {
                { 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 }, { 5, 5 }, { 6, 8 }
           });
    }

    private int fInput;

    private int fExpected;

    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }

    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
```