NGÔN NGỮ LẬP TRÌNH JAVA

Nội dung

Cơ bản về ngôn ngữ lập trình Java

- Lập trình hướng đối tượng
- Biến, từ khoá, kiểu dữ liệu
- Biểu thức, các cấu trúc điều khiển
- Dữ liệu kiểu mảng

Các khía cạnh nâng cao của lập trình hướng đối tượng

- Thiết kế lớp
- Thiết kế lớp nâng cao
- Xử lý ngoại lệ
- Generics
- Java Collection Framework
- Multithread& Concurrency
- Database Programming
- Network Programming
- Send mail
- Testing

TESTING

Nội dung

1. Giới thiệu về testing
2. Junit – Unit Testing
3. Example

1. Giới thiệu về testing

- Testing (kiểm thử) là một quá trình kiểm tra chức năng của pm, để xem pm có thực hiện theo đúng yêu cầu hay không

Phân loại testing

- Kiểm thử đơn vị (**Unit Test**):
  - kiểm thử các lớp hoặc component riêng lẻ (độc lập)
- Kiểm thử tích hợp (Integration Test):
  - kiểm thử một nhóm các lớp hoặc components
- Kiểm thử chức năng (Functional Test) :
  - kiểm thử trên một hệ thống đã tích hợp hoàn chỉnh
  - kiểm thử trên giao diện người dùng
- Kiểm thử sự suy thoái (Regression Test):
  - kiểm thử để đảm bảo một sự thay đổi (nâng cấp, vá lỗi, …) không làm hỏng hệ thống hoặc gây ra thêm lỗi mới.

Unit testing

- Kiểm thử từng lớp/component một cách độc lập, đảm bảo rằng từng lớp/component chạy theo đúng đặc tả.
- Unit testing rất cần thiết cho việc đảm bảo chất lượng phần mềm
- Trước khi có khái niệm Unit testing
  - LTV sử dụng các câu lệnh write (hoặc file trace) để in ra giá trị các biểu thức
    - Quá nhiều câu lệnh in sẽ gây khó khăn trong việc phân tích kết quả
  - Assertion : kiểm tra các giả định về logic của chương trình (pre- & post-conditions, invariants)
- 2 loại kiểm thử đơn vị : manual và automated

Manual testing vs Automated testing

- Thực hiện các test case một cách thủ công, ko có sự trợ giúp của công cụ
- Tốn thời gian, nhàm chán
- Tốn tài nguyên con người
- Kém tin cậy
- Không có lập trình được => ko tạo được các test case phức tạp

- Sử dụng công cụ để chạy các test case
- Thực hiện test case tự động => nhanh hơn
- Yêu cầu ít tài nguyên con người hơn
- Tin cậy hơn
- Lập trình được => tạo được các test case phức tạp

xUnit Unit Testing Framework

- xUnit : một nhóm các unit testing frameworks chia sẻ chung một kiến trúc phần mềm
  - Java => JUnit
  - .NET => Nunit
  - C++ => CppUnit
  - PHP => PHPUnit
  - …v.v…
- Một số thành phần cơ bản của xUnit:
  - Test case/ Test suites
  - Test fixtures: thiết lập các phương thức “set up” và “clean up” cho mỗi lần triệu gọi phương thức và lớp
  - Test runner: cài đặt cách thức các test case được thực thi
  - Test result formatter: định dạng lại kết quả test
  - Assertions: assertion methods for all primitive types and Objects and arrays

History of JUnit

- Kent Beck phát triển xUnit automated test tool cho Smalltalk vào giữa thập kỷ 90's
- Beck và Gamma (tác giả của "design patterns Gang of Four") phát triển JUnit trên chuyến bay từ Zurich tới Washington, D.C.
- Junit trở thành công cụ test chuẩn cho Java (Junit.org)
- Junit test generators trở thành thành phần ko thể thiếu của nhiều Java IDEs (Eclipse, BlueJ, Jbuilder, DrJava)
- Kể từ đây, xUnit tools được phát triển cho nhiều ngôn ngữ (Perl, C++, Python, Visual Basic, C#, …)

API's JUnit

Serial No	Class Name	Functionality
1	Assert	A set of assert methods.
2	TestCase	A test case defines the fixture to run multiple tests.
3	TestResult	A TestResult collects the results of executing a test case.
4	TestSuite	A TestSuite is a Composite of Tests.

Assert class

```
public class Assert extends java.lang.Object
```

- Mỗi assertEquals method có các tham số như sau: *message, expected-value, actual-value*

```
public static void assertEquals([String message,] long expected, long actual)
      // int and long: expected == actual
public static void assertEquals([String message,] double expected, double actual, double epsilon)
      // double: expect == actual within tolerance of epsilon
public static void assertEquals([String message,] Object expected, Object actual)
      // Object: expected.equals(actual)
public static void assertNotEquals(.....)
```

Assert methods

```
public static void assertSame([String message,] Object expected, Object actual)
      // Same Object: expected == actual
public static void assertNotSame(.....)

public static void assertTrue([String message,] boolean condition)
      // boolean: condition == true
public static void assertFalse([String message,] boolean condition)
      // boolean: condition == false

public static void assertNull([String message,] Object object)
      // object == null
public static void assertNotNull(......)

public static void assertArrayEquals([String message,], int[] expecteds, int[] actuals)
public static void assertArrayEquals([String message,], byte[] expecteds, byte[] actuals)
public static void assertArrayEquals([String message,], char[] expecteds, char[] actuals)
public static void assertArrayEquals([String message,], long[] expecteds, long[] actuals)
public static void assertArrayEquals([String message,], byte[] expecteds, byte[] actuals)
public static void assertArrayEquals([String message,], short[] expecteds, short[] actuals)
public static void assertArrayEquals([String message,], Object[] expecteds, Object[] actuals)

public static <T> void assertThat([String message,], T actual, org.hamcrest.Matcher<T> matcher)
```

TestCase Class

```
public abstract class TestCase extends Assert implements Test
```

S.N.	Methods & Description
1	**int countTestCases()** Counts the number of test cases executed by run(TestResult result).
2	**TestResult createResult()** Creates a default TestResult object.
3	**String getName()** Gets the name of a TestCase.
4	**TestResult run()** A convenience method to run this test, collecting the results with a default TestResult object.
5	**void run(TestResult result)** Runs the test case and collects the results in TestResult.
6	**void setName(String name)** Sets the name of a TestCase.
7	**void setUp()** Sets up the fixture, for example, open a network connection.
8	**void tearDown()** Tears down the fixture, for example, close a network connection.
9	**String toString()** Returns a string representation of the test case.

TestResult Class

```
public class TestResult extends Object
```

S.N.	Methods & Description
1	**void addError(Test test, Throwable t)** Adds an error to the list of errors.
2	**void addFailure(Test test, AssertionFailedError t)** Adds a failure to the list of failures.
3	**void endTest(Test test)** Informs the result that a test was completed.
4	**int errorCount()** Gets the number of detected errors.
5	**Enumeration<TestFailure> errors()** Returns an Enumeration for the errors.
6	**int failureCount()** Gets the number of detected failures.
7	**void run(TestCase test)** Runs a TestCase.
8	**int int runCount()** Gets the number of run tests.
9	**void startTest(Test test)** Informs the result that a test will be started.
10	**void stop()** Marks that the test run should stop.

- TestResult thu thập các kết quả của việc thực thi một test case.
- Là một thể hiện của mẫu Collecting Parameter pattern.
- Test framework phân biệt giữa failures và errors.
- Failure được dự đoán và kiểm tra với assertions.
- Errors là các vấn đề ko dự đoán được (ngoại lệ như ArrayIndexOutOfBoundsException).

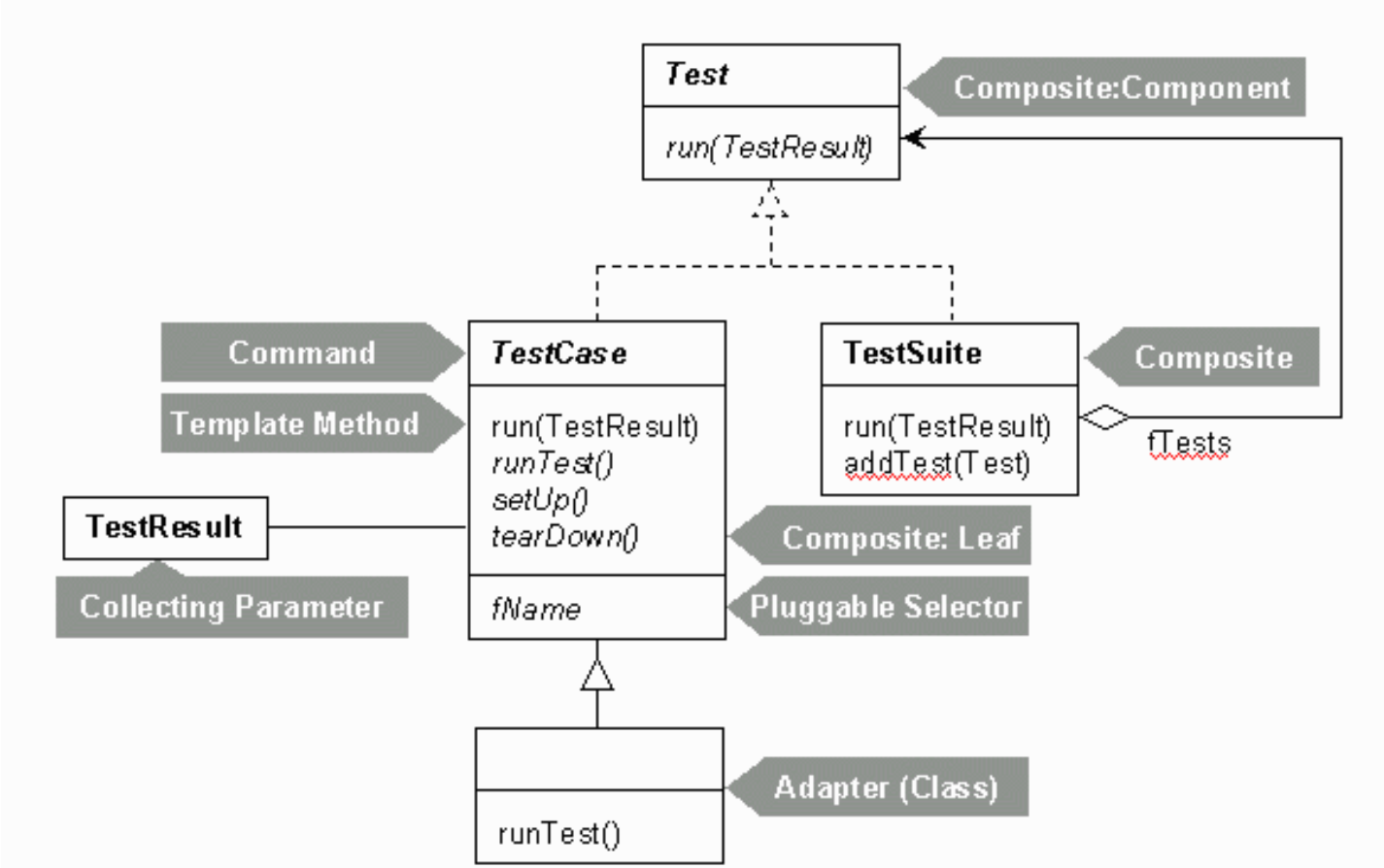
TestSuite Class

```
public class TestSuite extends Object implements Test
```

- TestSuite nhóm một tập các testcase, và thực hiện chúng

S.N.	Methods & Description
1	**void addTest(Test test)** Adds a test to the suite.
2	**void addTestSuite(Class<? extends TestCase> testClass)** Adds the tests from the given class to the suite.
3	**int countTestCases()** Counts the number of test cases that will be run by this test.
4	**String getName()** Returns the name of the suite.
5	**void run(TestResult result)** Runs the tests and collects their result in a TestResult.
6	**void setName(String name)** Sets the name of the suite.
7	**Test testAt(int index)** Returns the test at the given index.
8	**int testCount()** Returns the number of tests in this suite.
9	**static Test warning(String message)** Returns a test which will fail and log a warning message.

Kiến trúc JUnit



- **TestRunner** chạy các tests và trả về kết quả **TestResult**
- Kiểm thử một lớp bằng cách kế thừa abstract class ***TestCase***
- Để viết các test case, cần phải biết và hiểu **Assert** class

Cài đặt JUnit

- Goto http://junit.org/
  - ⇒ "Download and Install Guide"
  - ⇒ Download the "junit.jar" and "hamcrest-core.jar".
- Cấu hình
  - include JUnit jar-files "junit-4.##.jar" and "hamcrest-core-1.##.jar" in your CLASSPATH.
- Cấu hình Junit với Eclipse
  - Create a new Java project ⇒ right-click on the project ⇒ Properties ⇒ Java Build Path ⇒ "Libraries" tab
  - ⇒ Add Library ⇒ JUnit ⇒ In "JUnit library version", choose "JUnit 4"
  - ⇒ In "current location" use the eclipse's JUnit or your own download.
- Tạo test case (hoặc test suite)
  - ⇒ File ⇒ Others ⇒ Java ⇒ JUnit ⇒ JUnit test case (or JUnit test suite).
- Chạy test case (hoặc test suite)
  - ⇒ Run As ⇒ JUnit Test.

Viết testcase

- Để bắt đầu sử dụng JUnit, tạo một lớp kế thừa của *TestCase*:

```
import junit.framework.TestCase;
public class TestBowl extends TestCase {
} //Test my class Bowl
```

- Tên của lớp là quan trọng – nên có dạng ***Test***MyClass hoặc MyClass***Test***
- Quy ước đặt tên như vậy cho phép TestRunner tự động tìm được các lớp test

Viết testcase

- Khuôn mẫu cho 1 testcase:
  - Set up preconditions
  - Exercise functionality being tested
  - Check postconditions

- Example:

```
public void testEmptyList() {
    Bowl emptyBowl = new Bowl();
    assertEquals("Size of an empty list should be zero.",
                 0, emptyList.size());
    assertTrue("An empty bowl should report empty.",
    emptyBowl.isEmpty());
}
```

3. Ví dụ - Calculator

```
1  /**
2   * The Calculator class provides static methods for
3   *  arithmetic operations on two integers.
4   */
5  public class Calculator {
6     public static int add(int number1, int number2) {
7        return number1 + number2;
8     }
9
10    public static int sub(int number1, int number2) {
11       return number1 - number2;
12    }
13
14    public static int mul(int number1, int number2) {
15       return number1 * number2;
16    }
17
18    // Integer divide. Return a truncated int.
19    public static int divInt(int number1, int number2) {
20       if (number2 == 0) {
21          throw new IllegalArgumentException("Cannot divide by 0!");
22       }
23       return number1 / number2;
24    }
25
26    // Real number divide. Return a double.
27    public static double divReal(int number1, int number2) {
28       if (number2 == 0) {
29          throw new IllegalArgumentException("Cannot divide by 0!");
30       }
31       return (double) number1 / number2;
32    }
33 }
```

Viết testcase

- Create a new Eclipse Java project called "JUnitTest".
- Create a new class called "Calculator" under "src" folder, with the above program code.
- Create a new folder called "test" for storing test scripts ⇒ Right-click on the project ⇒ New ⇒ Folder ⇒ In folder name, enter "test". Make "test" a source folder by right-click on "test" ⇒ Build Path ⇒ Use as source folder.
- Create the first test case called "AddSubTest" ⇒ Right-click on folder "test" ⇒ New ⇒ Other ⇒ Java ⇒ JUnit ⇒ JUnit Test Case ⇒ New JUnit 4 test ⇒ In Name, enter "AddSubTest". Enter the following codes:

Viết testcase

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class AddSubTest {
5     @Test
6     public void testAddPass() {
7        // assertEquals(String message, long expected, long actual)
8        assertEquals("error in add()",  3, Calculator.add(1, 2));
9        assertEquals("error in add()", -3, Calculator.add(-1, -2));
10       assertEquals("error in add()",  9, Calculator.add(9, 0));
11    }
12
13    @Test
14    public void testAddFail() {
15       // assertNotEquals(String message, long expected, long actual)
16       assertNotEquals("error in add()", 0, Calculator.add(1, 2));
17    }
18
19    @Test
20    public void testSubPass() {
21       assertEquals("error in sub()",  1, Calculator.sub(2, 1));
22       assertEquals("error in sub()", -1, Calculator.sub(-2, -1));
23       assertEquals("error in sub()",  0, Calculator.sub(2, 2));
24    }
25
26    @Test
27    public void testSubFail() {
28       assertNotEquals("error in sub()", 0, Calculator.sub(2, 1));
29    }
30 }
```

Kiểm thử với TestRunner

```
 1  import org.junit.runner.JUnitCore;
 2  import org.junit.runner.Result;
 3  import org.junit.runner.notification.Failure;
 4
 5  public class RunTestStandalone {
 6     public static void main(String[] args) {
 7        Result result = JUnitCore.runClasses(AddSubTest.class);
 8        for (Failure failure : result.getFailures()) {
 9           System.out.println(failure.toString());
10        }
11        System.out.println(result.wasSuccessful());
12     }
13  }
```

Thêm testcase

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class DivTest {
5     @Test
6     public void testDivIntPass() {
7        assertEquals("error in divInt()", 3, Calculator.divInt(9, 3));
8        assertEquals("error in divInt()", 0, Calculator.divInt(1, 9));
9     }
10
11    @Test
12    public void testDivIntFail() {
13       assertNotEquals("error in divInt()", 1, Calculator.divInt(9, 3));
14    }
15
16    @Test(expected = IllegalArgumentException.class)
17    public void testDivIntByZero() {
18       Calculator.divInt(9, 0); // expect an IllegalArgumentException
19    }
20
21    @Test
22    public void testDivRealPass() {
23       assertEquals("error in divInt()", 0.333333, Calculator.divReal(1, 3), 1e-6);
24       assertEquals("error in divInt()", 0.111111, Calculator.divReal(1, 9), 1e-6);
25    }
26
27    @Test(expected = IllegalArgumentException.class)
28    public void testDivRealByZero() {
29       Calculator.divReal(9, 0); // expect an IllegalArgumentException
30    }
31 }
```

Viết TestSuite

To create a test suite under Eclipse ⇒ right-click on the test folder ⇒ New ⇒ other ⇒ Java ⇒ JUnit ⇒ JUnit Test Suite ⇒ In Name, enter "AllTests" ⇒ Select test cases to be included - AddSubTest and DivTest.

```
1  import org.junit.runner.RunWith;
2  import org.junit.runners.Suite;
3  import org.junit.runners.Suite.SuiteClasses;
4
5  @RunWith(Suite.class)
6  @SuiteClasses({ AddSubTest.class, DivTest.class })
7  public class AllTests {
8
9  }
```

To run the test suite ⇒ right-click on the file ⇒ Run as ⇒ JUnit Test. Observe the test results produced.

Junits 4's annotations

`@Test`	The annotated method is to be run as a test method.
`@Before`	The annotated method is to be run before EACH of the test method.
`@After`	The annotated method is to be run after EACH of the test method.
`@BeforeClass`	The annotated method is to be run ONCE before any of the test method.
`@AfterClass`	The annotated method is to be run ONCE after all the test methods.
`@Ignore`	Ignore (don't run) the test method. This is a convenient way to mark out a test method

Example 2

- Create a Java project called "JUnitTest2"
- Create a new Java class called "MyNumber", as follow:

```
5  public class MyNumber {
6     int number;
7
8     // Constructor
9     public MyNumber() {
10       this.number = 0;
11    }
12
13    public MyNumber(int number) {
14       this.number = number;
15    }
16
17    // Getter and setter
18    public int getNumber() {
19       return number;
20    }
21
22    public void setNumber(int number) {
23       this.number = number;
24    }
```

```
    // Public methods
    public MyNumber add(MyNumber rhs) {
       this.number += rhs.number;
       return this;
    }

    public MyNumber div(MyNumber rhs) {
       if (rhs.number == 0) throw new Illegal
       this.number /= rhs.number;
       return this;
    }
}
```

```
IllegalArgumentException("Cannot divide by 0!");
```

- Create a new source folder called "test" for storing test scripts. Make it a source folder by right-click ⇒ Build Path ⇒ Use as source folder.
- Create the first test case called MyNumberTest (under "test" folder), as follows:

```
 1  import static org.junit.Assert.*;
 2  import org.junit.After;
 3  import org.junit.Before;
 4  import org.junit.Test;
 5
 6  public class MyNumberTest {
 7     private MyNumber number1, number2; // Test fixtures
 8
 9     @Before
10     public void setUp() throws Exception {
11        System.out.println("Run @Before"); // for illustration
12        number1 = new MyNumber(11);
13        number2 = new MyNumber(22);
14     }
```

```
16      @After
17      public void tearDown() throws Exception {
18         System.out.println("Run @After"); // for illustration
19      }
20
21      @Test
22      public void testGetterSetter() {
23         System.out.println("Run @Test testGetterSetter"); // for illustration
24         int value = 33;
25         number1.setNumber(value);
26         assertEquals("error in getter/setter", value, number1.getNumber());
27      }
28
29      @Test
30      public void testAdd() {
31         System.out.println("Run @Test testAdd"); // for illustration
32         assertEquals("error in add()", 33, number1.add(number2).getNumber());
33         assertEquals("error in add()", 55, number1.add(number2).getNumber());
34      }
35
36      @Test
37      public void testDiv() {
38         System.out.println("Run @Test testDiv"); // for illustration
39         assertEquals("error in div()", 2, number2.div(number1).getNumber());
40         assertEquals("error in div()", 0, number2.div(number1).getNumber());
41      }
42
43      @Test(expected = IllegalArgumentException.class)
44      public void testDivByZero() {
45         System.out.println("Run @Test testDivByZero"); // for illustration
46         number2.setNumber(0);
47         number1.div(number2);
48      }
49   }
```

The output, used for illustrating the sequence of operations, is as follows:

```
Run @Before
Run @Test testDivByZero
Run @After
Run @Before
Run @Test testAdd
Run @After
Run @Before
Run @Test testDiv
Run @After
Run @Before
Run @Test testGetterSetter
Run @After
```