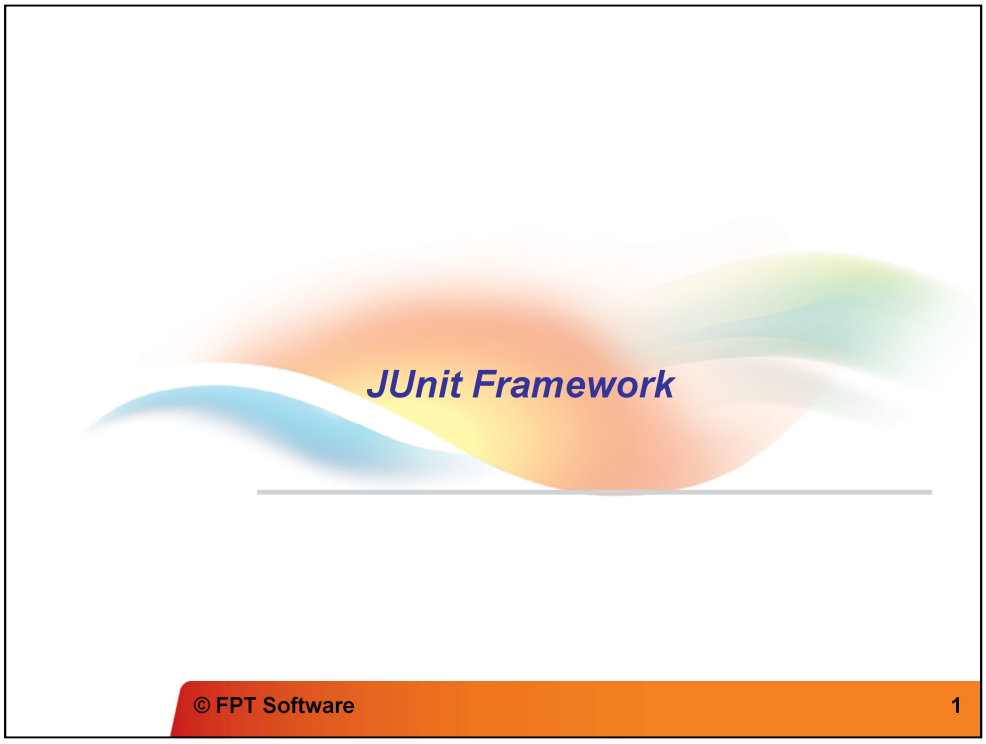



Latest updated by: HanhTT1

# Introduction

- **Duration:** *2 hours*
- **Purpose:** *introduce about unit testing and JUnit*
- **Audience:** developers
- **Test:** *Practice test*

# Objectives

After the course, student will:

- Understand Unit testing and Junit
- Know how to write a JUnit test class
- Know how to install JUnit and run a JUnit test case
- Know about some tips of Unit testing

# Agenda

- Introduction
- What is JUnit?
- Setting up JUnit
- Exploring JUnit
- Samples
- Calculate Code coverage
- Tips

**Notice**: This training is for JUnit 3

## Introduction

- **You know**

  Every programmer knows they should write tests for their code.

- **But**

  The universal response to "Why not?" is "I'm in too much of a hurry."

**JUnit comes as a solution for this situation ……..**

## What is JUnit?

- JUnit is de facto Java unit testing framework
- Integrated nicely with many IDEs, Ant
- Easy to use and to learn

# Agenda

- Introduction
- What is JUnit?
- **Setting up JUnit**
- Exploring JUnit
- Samples
- Calculate Code coverage
- Tips

## Setting up JUnit

- Download JUnit from *http://junit.org/*
- Installation
  - Unzip the junit.zip file
  - Add **junit.jar** to the CLASSPATH.
- Testing

Test the installation by using either the batch or the graphical TestRunner tool to run the tests that come with this release.

All the tests should pass OK.

**Notice**:

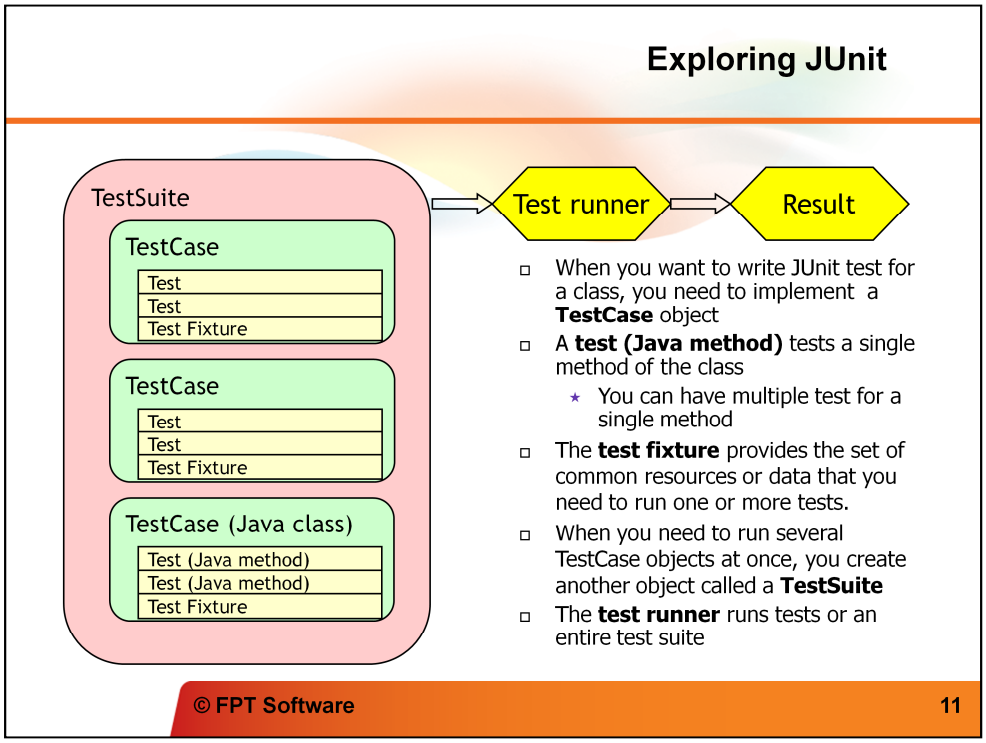
- The tests are not contained in the junit.jar but in the installation directory directly. Therefore make sure that the installation directory is on the class path

## Setting up JUnit (cont)

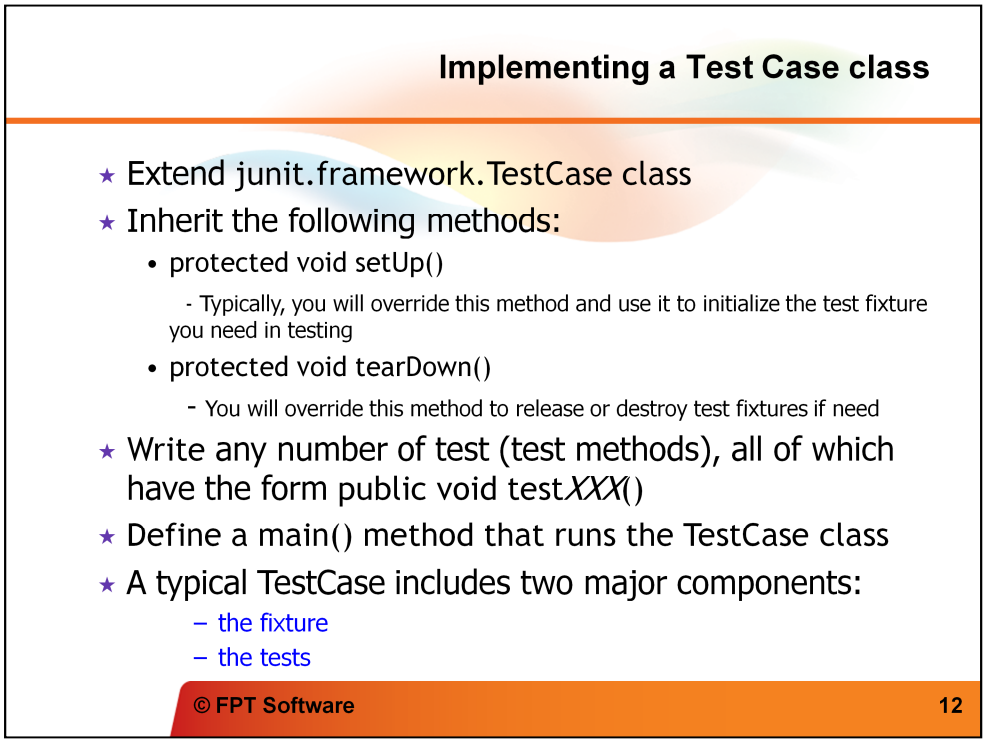
- Testing
  - for the console TestRunner type:
    *java junit.textui.TestRunner junit.samples.AllTests*
  - for the graphical TestRunner type:
    *java junit.awtui.TestRunner junit.samples.AllTests*
  - for the Swing based graphical TestRunner type:
    *java junit.swingui.TestRunner junit.samples.AllTests*

# Agenda

- Introduction
- What is JUnit?
- Setting up JUnit
- **Exploring JUnit**
- Samples
- Calculate Code coverage
- Tips



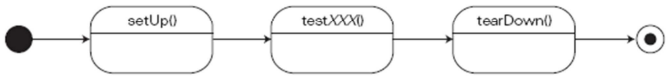

- When you write JUnit test, most often you write TestCases and Tests




- The framework ships with ready-to-use graphical and textual TestRunners. The framework can also generate a default runtime TestSuite for you. So, the only class you absolutely must provide yourself is the TestCase.

## Test Fixture

- The set of background resources that you need to run a test is commonly called a *test fixture*. Eg: database connection, file, test data…
- A fixture is automatically created and destroyed by a TestCase through its setUp and tearDown methods.
- Order when run a test method



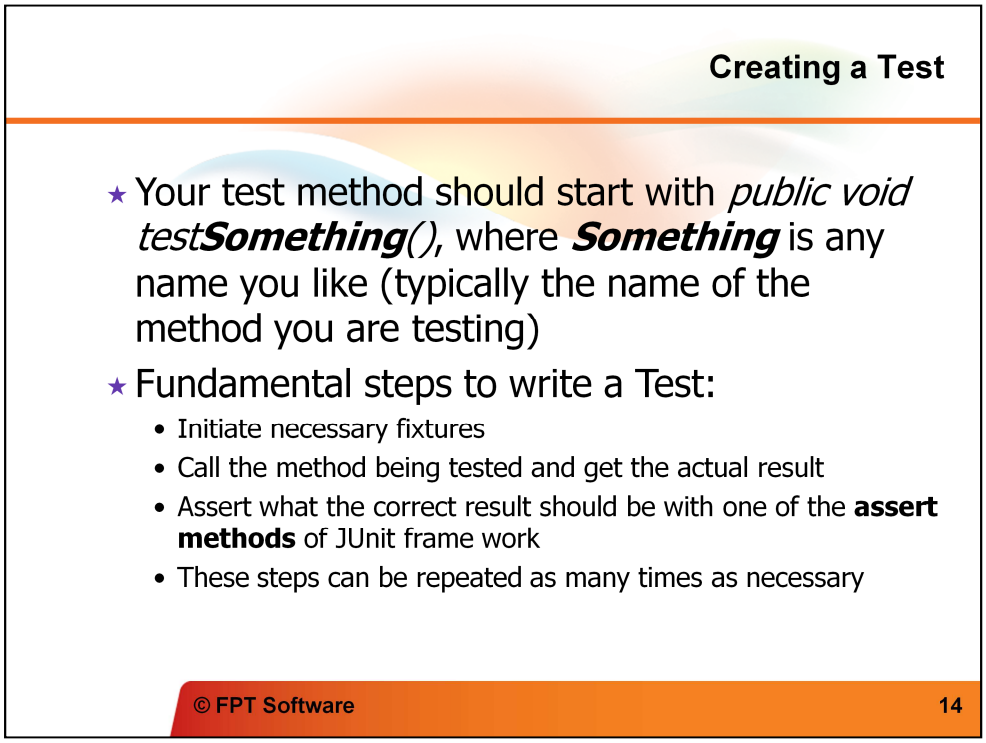




- A database connection is a good example of why you might need a fixture. If a TestCase includes several database tests, they each need a fresh connection to the

database. A fixture makes it easy for you to open a new connection for each test without replicating code.

-




- The framework ships with ready-to-use graphical and textual TestRunners. The framework can also generate a default runtime TestSuite for you. So, the only class you absolutely must provide yourself is the TestCase. A typical TestCase includes two major components: the fixture and the unit tests.

- You do not need to call these methods, they will be called automatically

- You do not need to do anything if the tests fail; the framework will take care of this for you

## Assert methods

- An assert method is a JUnit method that performs a test, and throws an AssertionFailedError if the test fails.
- There are 8 core assert methods:
  - assertTrue: Asserts that a condition is true
  - assertFalse: Asserts that a condition is true.
  - assertEquals: Asserts that two objects are equal.
  - assertNotNull: Asserts that an object isn't null.
  - assertNull: Asserts that an object is null.
  - assertSame: Asserts that two objects refer to the same object.
  - assertNotSame: Asserts that two objects do not refer to the same object.
  - fail: Fails a test immediately with the given message.

## The Counter class

```
public class Counter {
      int count = 0;

      public int increment() {
         return ++count;
      }

      public int decrement() {
         return --count;
      }

     public int getCount() {
        return count;
     }
}
```

- The constructor will create a counter and set it to zero
- The increment method will add one to the counter and return the new value
- The decrement method will subtract one from the counter and return the new value

## JUnit test for Counter

```
public class CounterTest extends junit.framework.TestCase {
   Counter counter;

   public CounterTest() { }  // default constructor

   protected void setUp() {   // creates a (simple) test fixture
      counter = new Counter();
   }

   protected void tearDown() { } // no resources to release

   public void testIncrement() {
      assertTrue(counter.increment() == 1);
      assertTrue(counter.increment() == 2);
    }

   public void testDecrement() {
      assertEqual(counter.decrement(), -1);
   }

  public static void main(String[] args) {
     junit.textui.TestRunner.run(CounterTest .class);
  }
}
```

Note that each test begins with a *brand new* counter

This means you don't have to worry about the order in which the tests are run

## Creating a TestSuite

- Create a class and implement suite() method to create TestSuite and add TestCase classes to TestSuite

```
public static Test suite() {
    TestSuite suite = new TestSuite("test");
    suite.addTest(Counter.class);
    suite.addTest(Caculator.class);
    return suite;
}
```

- Define a main() method that runs the TestSuite

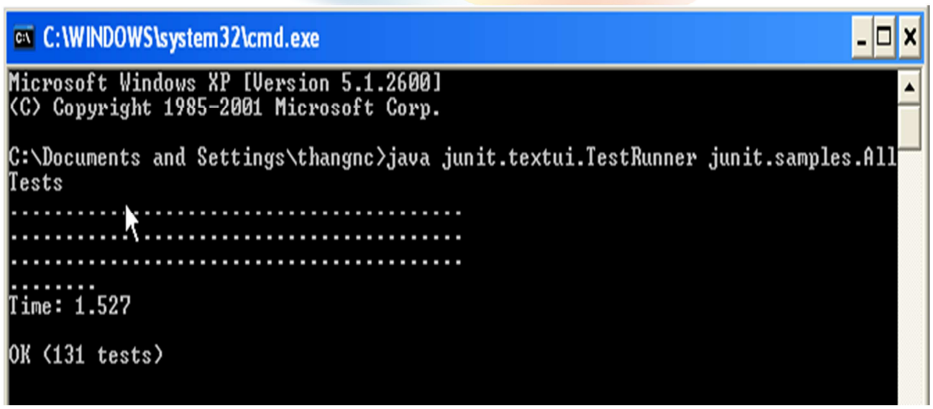
```
public static void main (String [] args) {
   junit.textui.TestRunner.run(suite());
  //junit.swingui.TestRunner.run(suite());
}
```

## Test Runner

- The JUnit distribution includes three TestRunner classes:
  - For the text console: *junit.textui.TestRunner*
  - For Swing: *junit.swingui.TestRunner*
  - for AWT: *junit.awtui.TestRunner*
- Run a TestSuite or a TestCase
  - java *junit.textui.TestRunner* suite class/testcase class
  - java *junit.swingui.TestRunner* suiteclass/testcase class
  - Java *junit.awtui.TestRunner* *suiteclass*/testcase class
- Many IDEs support their own TestRunner to run: Eclipse, JDeveloper…

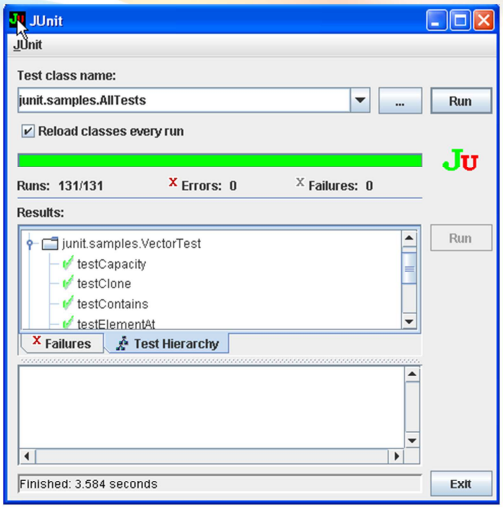
## Test Runner (cont)

- Text console



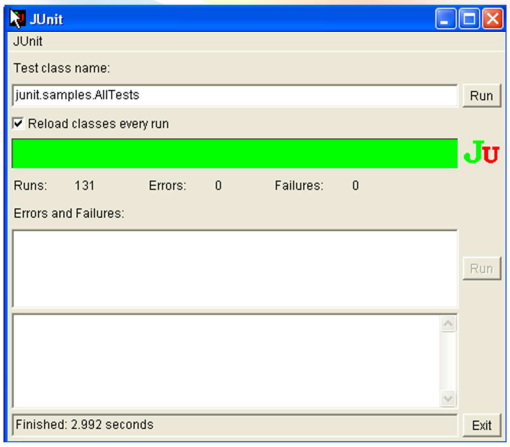
## Test Runner (cont)

- Swing



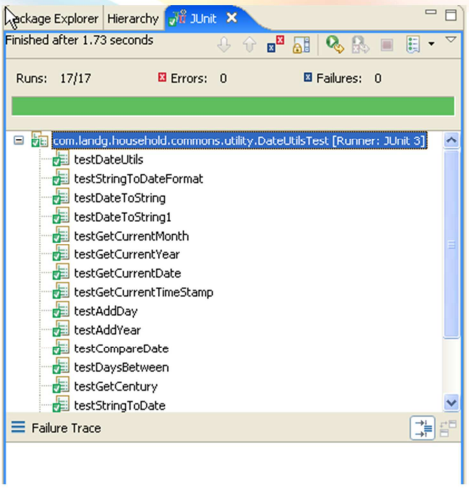
## Test Runner (cont)

- Awt



## Test Runner (cont)

- Eclipse



## Test Result

- A test method in a test case can have one of three results:
  - Pass – all assertions matched expected values
  - Failed – an assertion did not match expected value
  - Error – an unexpected exception was thrown during execution of test method

# Agenda

- Introduction
- What is JUnit?
- Setting up JUnit
- Exploring JUnit
- Samples
- Calculate Code coverage
- Tips

## Sample

- **Demo Shopping Cart sample**

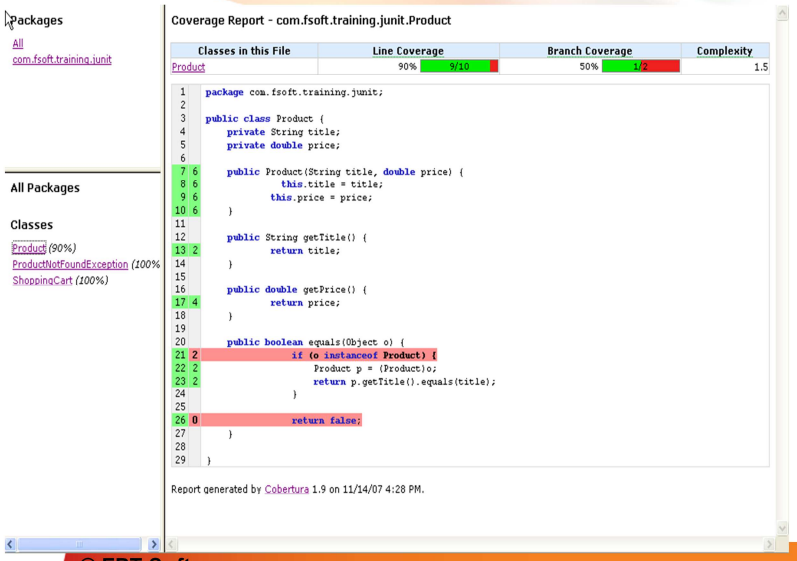
## Agenda

- Introduction
- What is JUnit?
- Setting up JUnit
- Exploring JUnit
- Samples
- Calculate Code Coverage
- Tips

## Code coverage

- Good unit tests are thorough; they test everything that's likely to break. You can aim to test every line of code, every possible branch the code might take, every exception it throws, and so on
- Use code coverage tools  to determine how much of the code under test is actually being exercised (Clover, Cobertura, EMMA  ...)

# Cobertura tool

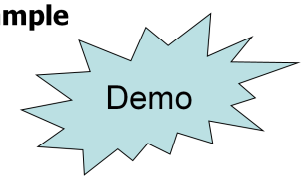


## Cobertura tool

- What is **Cobertura**?
  - free Java tool that calculates the percentage of code accessed by tests.
  - used to identify which parts of your Java program are lacking test coverage.
- Features
  - Can be executed from ant or from the command line.
  - Instruments Java bytecode after it has been compiled.
  - Can generate reports in HTML or XML.
  - Shows the percentage of lines and branches covered for each class, each package, and for the verall project.
  - Can sort HTML results in ascending or descending order by
    - class name
    - percent of lines covered
    - percent of branches covered
- Download at : *http://cobertura.sourceforge.net/*

## Cobertura tool

- How to setup ?
  - Setting value for system values
    - JAVA_HOME
    - ANT_HOME
    - Include ANT_HOME/bin and JAVA_HOME/bin to PATH variable
    - Point CLASSPATH to necessary Jar, zip files
  - Create build properties file – Configure report structure
  - Create Ant build file – build.xml file
- Demo **Shopping Cart sample**



# Agenda

- Introduction
- What is JUnit?
- Setting up JUnit
- Exploring JUnit
- Samples
- Calculate Code coverage
- Tips

## Tips (Best practices)

- Do not use the test-case constructor to set up a test case
- Don't assume the order in which tests within a test case run
- Call a superclass's setUp() and tearDown() methods when subclassing
- Utilize JUnit's assert/fail methods and exception handling for clean test code
- Keep tests small and fast
- Do not load data from hard-coded locations on a filesystem
- Name tests properly
- For each Java package in your application, define a TestSuite class that contains all the tests for validating the code in the package.
- Define similar TestSuite classes that create higher-level and lower-level test suites in the other packages (and sub-packages) of the application.
- Make sure your build process includes the compilation of all tests.

## Mock Object

- Unit-testing each method in isolation from the other methods or the environment is certainly a nice goal. But how do you perform this feat?
- The answer is called *mock objects*.
- A *mock object* (or *mock* for short) is an object created to stand in for an object that your code will be collaborating with. Your code can call methods on the mock object, which will deliver results as set up by your tests.
- Some mock object frame works:
  - EasyMock: *www.easymock.org/*
  - jMock: *www.jmock.org*
  - Mocquer: *mocquer.dev.java.net/*

## JUnit extensions

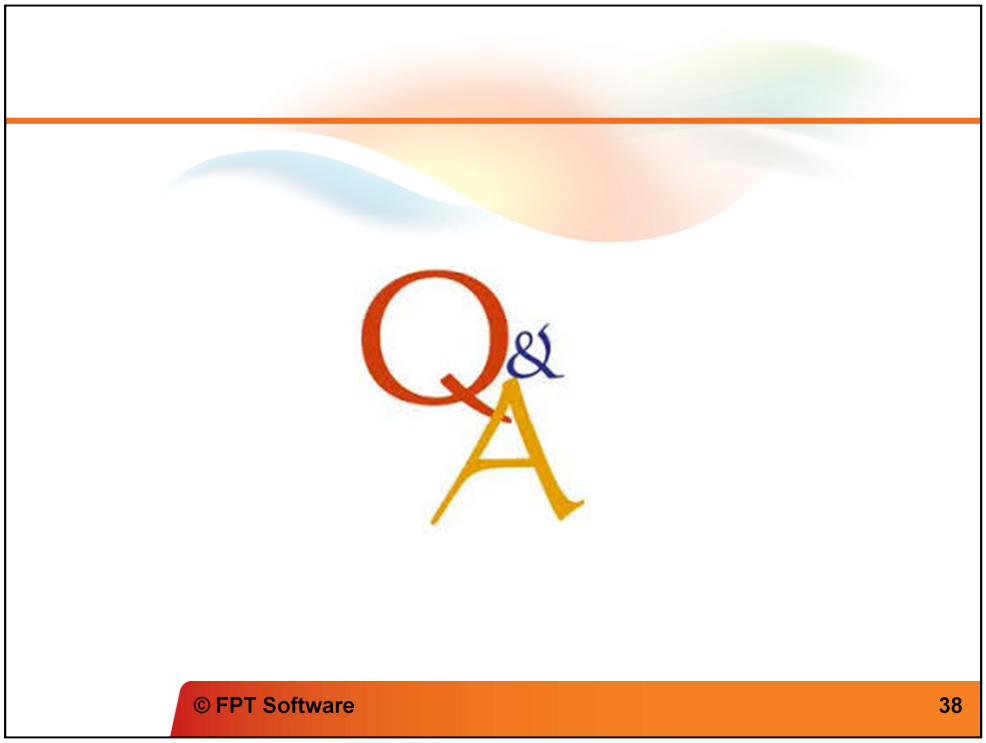
- There are many JUnit extensions supporting for implementing JUnit test:
  - DBUnit to test database
  - HttpUnit, JWebUnit, Canoo WebTest to test interface
  - XMLUnit to test XML
  - StrutsTestCase to test Struts
  - ...

## CONCLUSION

- JUnit is de facto Java unit testing framework
- Use TestRunner,  Ant or IDEs to run test cases
- Extend junit.framework.TestCase class
- setUp() method is used to initialize test case
- tearDown() is used to release or destroy
- Use TestRunner or IDEs to run your test cases
- There are many JUnit extensions such as DBUnit, HttpUnit, StrutsTestCase ...

## Resources & references

- Resources
  - www.junit.org/
  - http://junit.sourceforge.net
- Recommended readings
  - Manning – Junit in Action
  - Test Driven Development: By Example. Boston: Addison-Wesley, 2003


Q&A