08 JUnit (Overview & Writing test)

PowerPoint by Wanida Khamprapai

JUnit

- JUnit (http://www.junit.org) is an open-source testing framework, released under IBM's Common Public License Version 1.0 and hosted on SourceForge, which is used to write and run repeatable automated tests so that we can ensure that our code works as expected.
- JUnit is widely used in industry and can be used as a stand-alone Java program (from the command line) or within an IDE such as Eclipse. Unit testing is the process of examining a small "unit" of software (usually a single class) to verify that it meets its expectations or specifications.
- JUnit quickly became the de facto standard framework for developing unit tests in Java. The underlying testing model, known as xUnit, is on its way to becoming the standard framework for any language. There are xUnit frameworks available for ASP, C++, C#, Eiffel, Delphi, Perl, PHP, Python, REBOL, Smalltalk, and Visual Basic—to name a few!

Junit provides:

- Assertions for testing expected results.
- Test features for sharing common test data.
- Test suites for easily organizing and running tests.
- Graphical and textual test runners.

Junit is used to test:

- An entire object
- Part of an object – a method or some interacting methods
- Interaction between several objects

JUnit

Unit testing

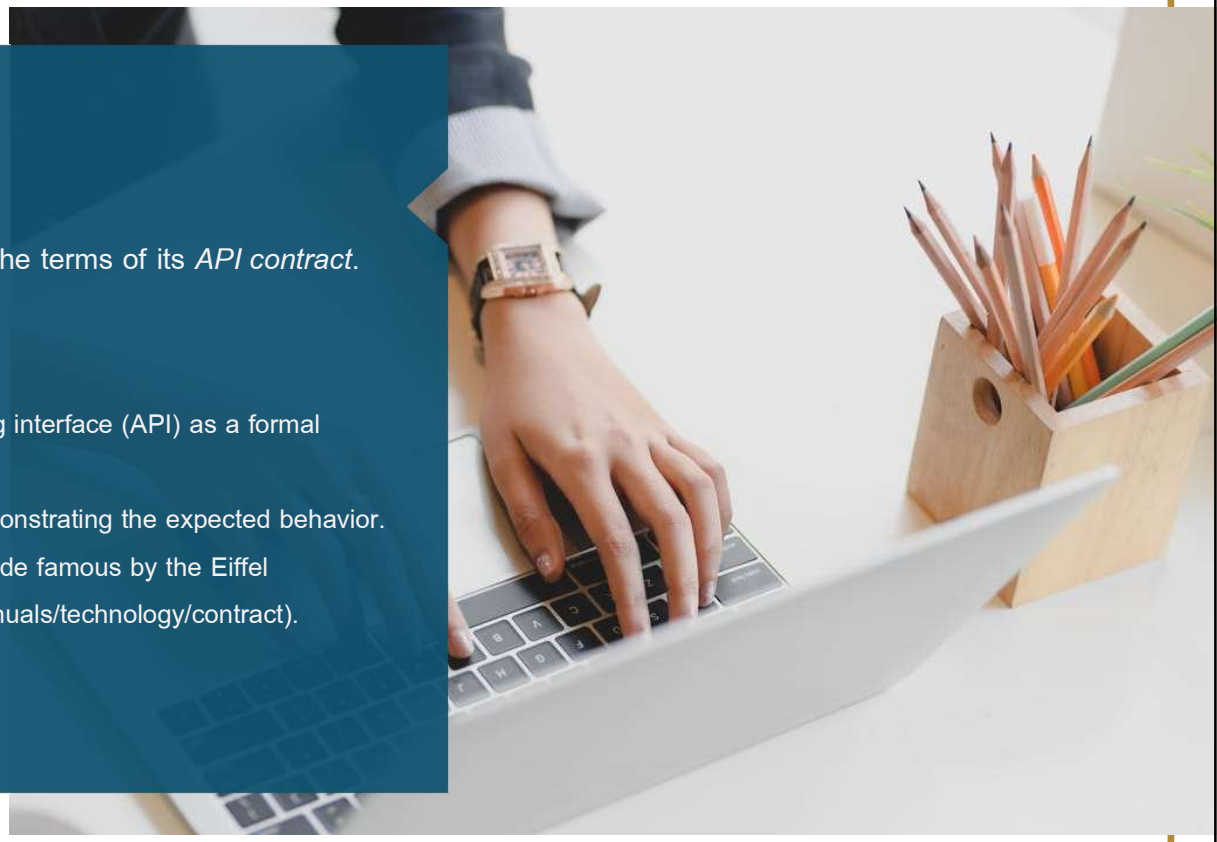
- A unit test examines the behavior of a distinct unit of work. Within a Java application, the “distinct unit of work” is often (but not always) a single method.
- By contrast, integration tests and acceptance tests examine how various components interact.
- A unit of work is a task that isn’t directly dependent on the completion of any other task.

JUnit

Unit tests often focus on testing whether a method follows the terms of its *API contract*.

API contract

- An API contract is a view of an application programming interface (API) as a formal agreement between the caller and the callee.
- Often the unit tests help define the API contract by demonstrating the expected behavior.
- The idea of an API contract is derived from the use, made famous by the Eiffel programming language (http://archive.eiffel.com/doc/manuals/technology/contract).



JUnit

Framework for automating unit testing

- Allows *creation* of unit tests.
- Allows *execution* of unit tests.

A JUnit test case is a Java class containing one or more unit tests.

- A test case is Java class that declares at least one unit test.
- A unit test is a public, void, no-parameter method with the annotation @Test.
- A test suite is a collection of test cases, declared as an empty Java class with the annotations @RunWith and @Suite.

Test can be run as individual test cases or entire test suites.

- Tests run in batch mode, i.e. there is on interaction with the user during the running of the tests.
- Outcome of running a unit test is either pass or fail.

JUnit

Writing test cases with JUnit

JUnit tests are written in the same manner as Java projects, however, while writing tests you must be aware of some JUnit annotations. JUnit annotations are blocks of predefined texts provided by the Java API for better structure and readability.

Example of a few frequently used JUnit annotations:

- @Test – to identify the actual test case
- @Before – to execute some statement before each test case
- @After – to execute some statement after each test case
- @Ignores – to ignore some statement during execution of test
- @BeforeAll – to execute some statement before all the test cases
- @AfterAll – to execute some statement after all the test cases

JUnit

Best practices for writing test cases with JUnit

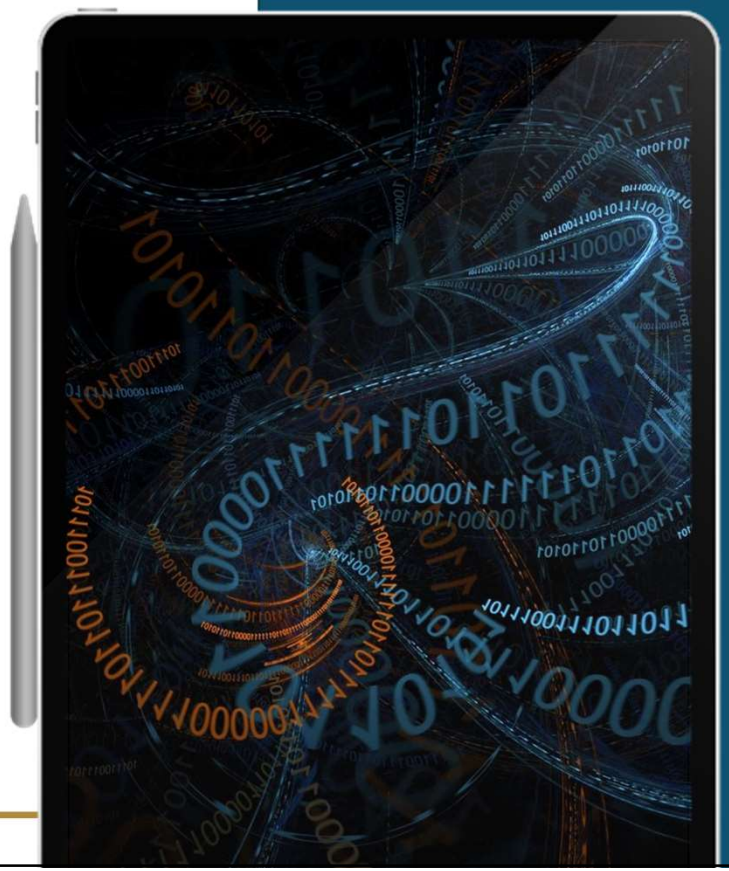
To be more efficient in our testing, let's look at some best practices for unit testing you can follow to ensure that your product is tested more reliably.

1. Test on Real Devices
   - In order to ensure a consistent and seamless user experience across all devices, always test your products on real devices before rolling them out to consumers.
2. Keep in mind the business
   - While writing tests for the product you must keep in mind the test aligns with the business requirements of the project.
3. Using Annotations wisely
   - Annotations are very useful in writing JUnit tests, as they make our code more structured and organized.
4. Test Core methods
   - It is important that you write JUnit tests only for the components that are prone to bugs.
5. Mocking
   - In JUnit testing, you can only test small chunks of code. There are chances that some codes are dependent on external services, for which, you have to use third-party frameworks such as Mockito, EasyMock, and JMock for mocking external services.

JUnit

Writing simple test cases with JUnit

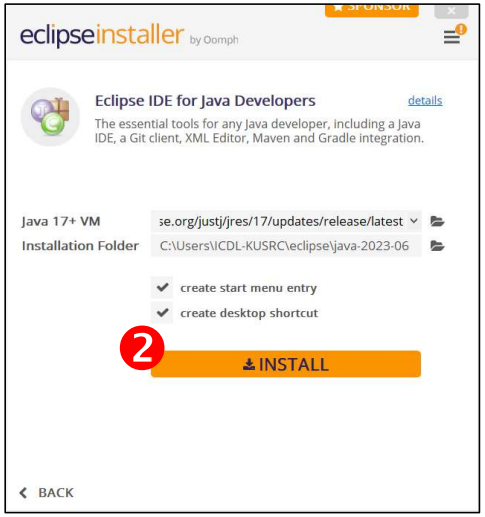
1. Declare a java class that will function as a test case.
   - The name of the class can be any legal name which is meaningful in the given context.
2. Declare one or more unit tests by implementing methods in the test case class.
   - The declaration of the unit test method must have the annotation @Test.
   - A unit test method must be public, void, and have no parameters.
   - The JUnit framework requires that a test case has at least one unit test method.
3. Call assertFact() methods in each unit test method to perform the actual testing.
   - Various overloaded assertFact() methods are declared in the org.junit.Assert class to test different conditions.
   - Typically, the assertEquals() method is used to assert that its two arguments are equal.
   - Statically import assertFact() methods from the org.junit.Assert class for convenience.
4. Can also use assert statements to perform the tests.

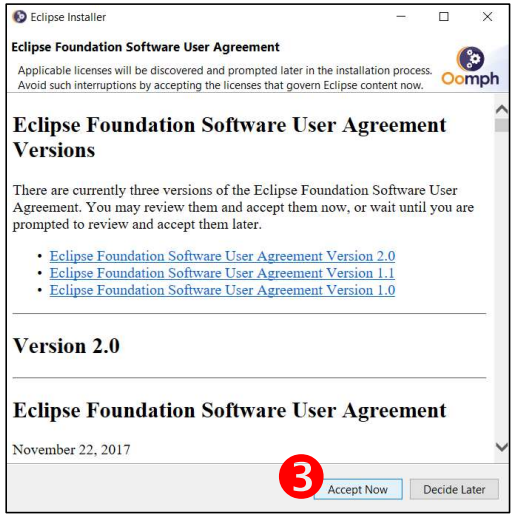


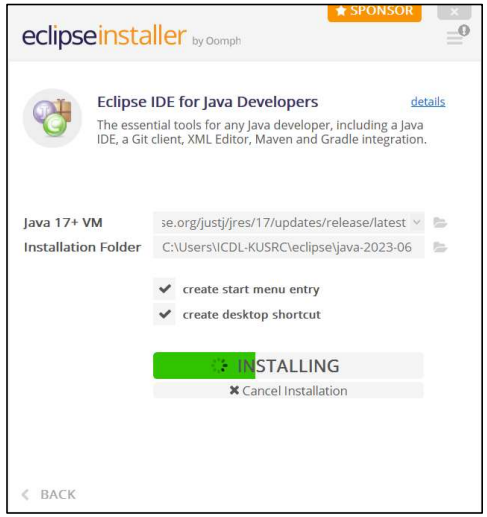
JUnit testing in Eclipse

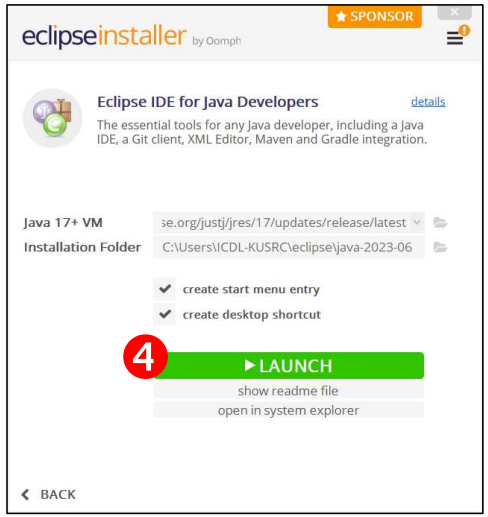
Installation - Eclipse

Download Eclipse from https://www.eclipse.org/downloads/



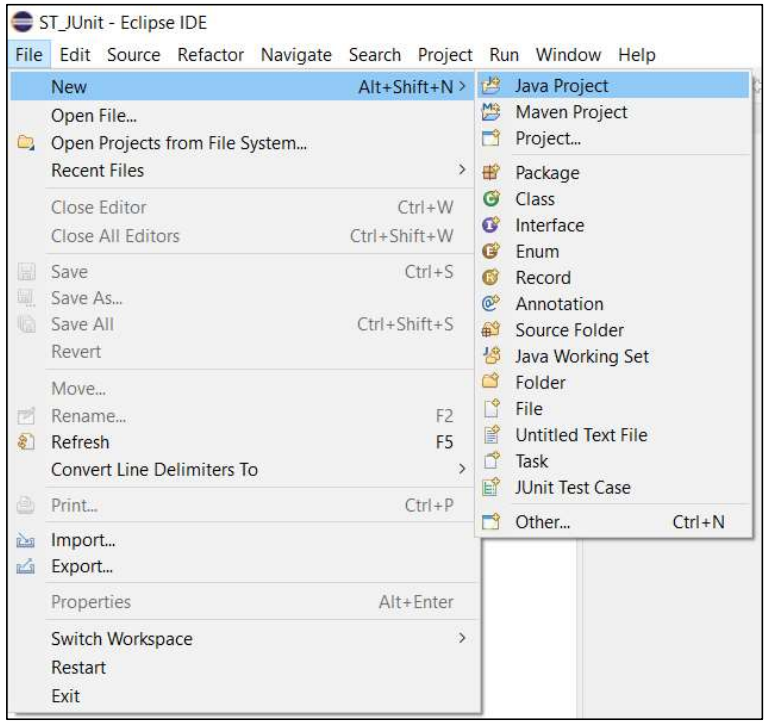


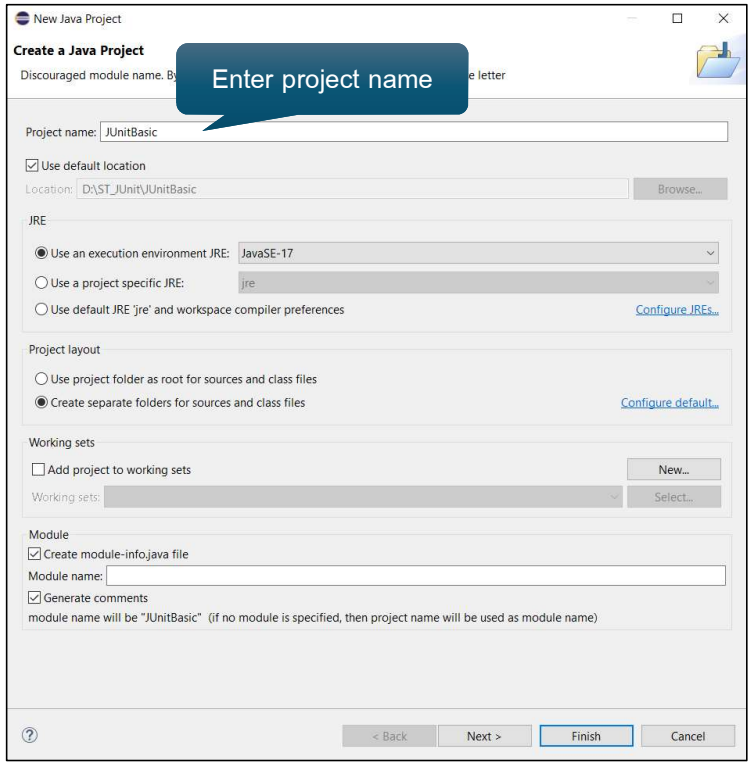




JUnit testing in Eclipse

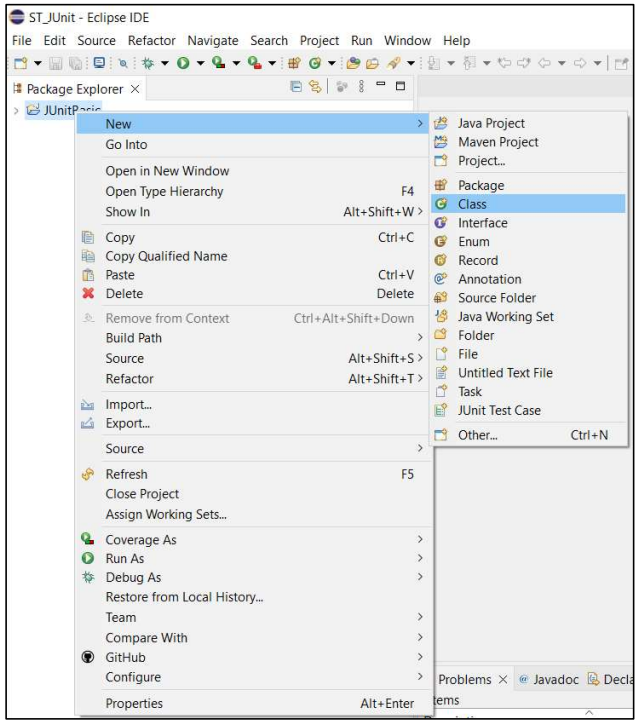
- Create new Java project

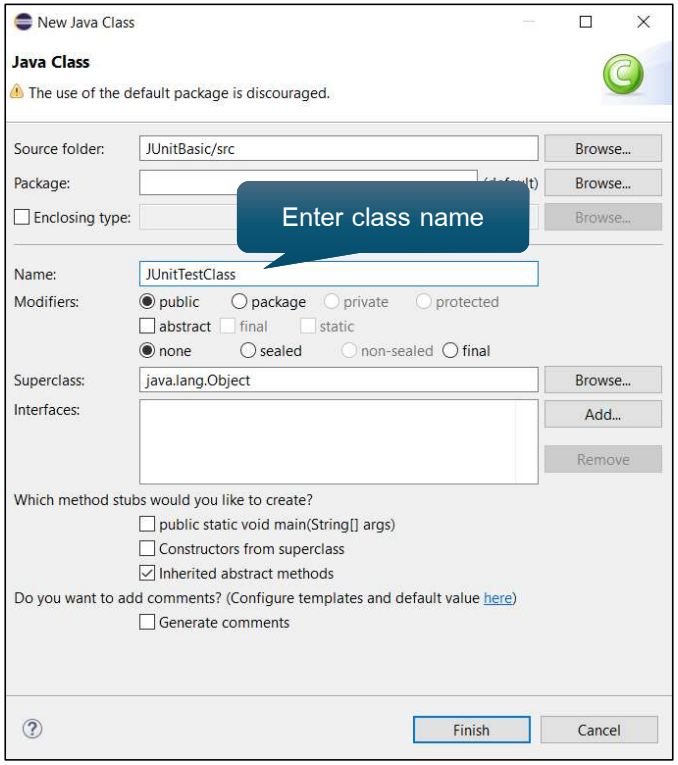




JUnit testing in Eclipse

- Create new Java class >>> Right-click on the project name



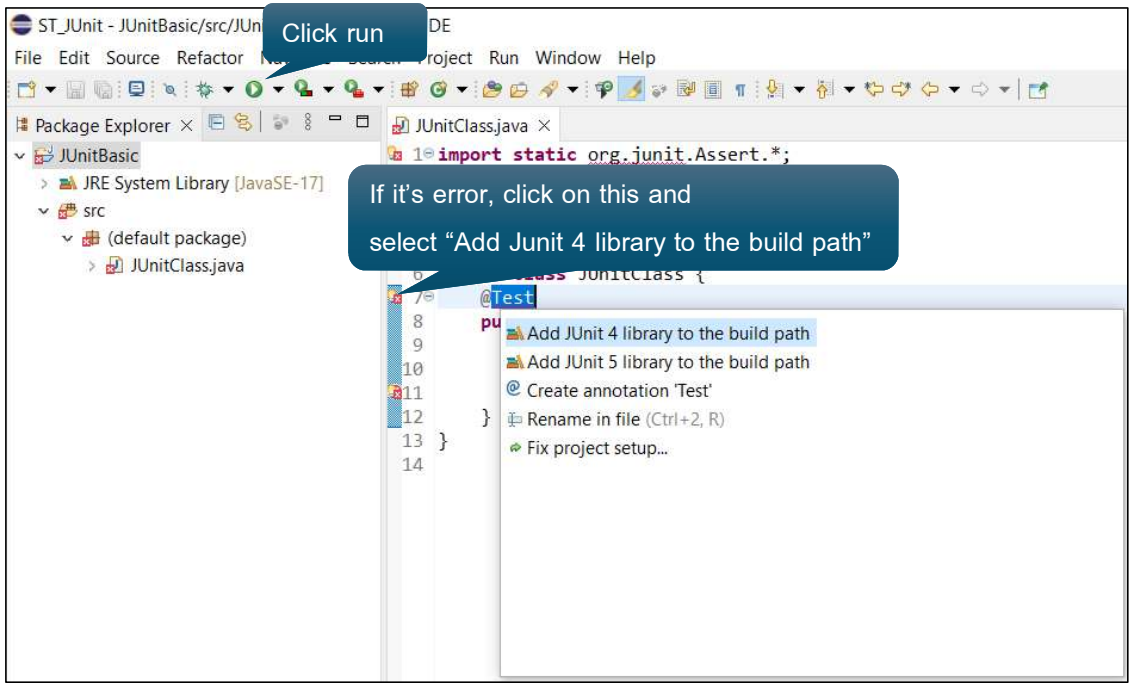


JUnit testing in Eclipse

Example:

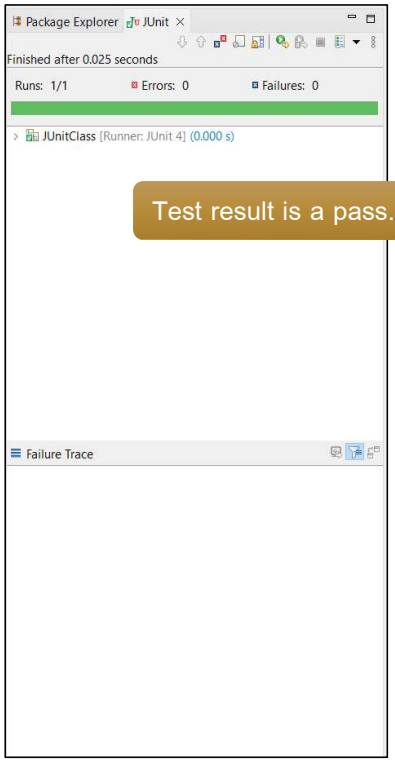
Enter this code

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

public class JUnitClass {
	@Test
	public void test_Junit() {
		System.out.println("Let's check this string");
		String str = "Let's check this string";
		assertEquals("Let's check this string", str);

	}
}
```

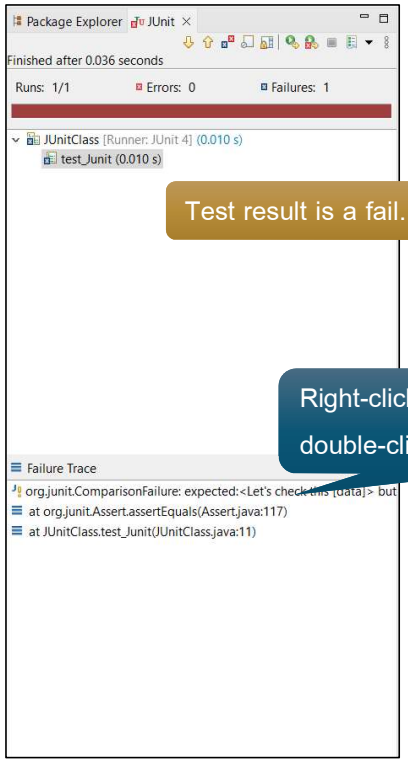


JUnit testing in Eclipse

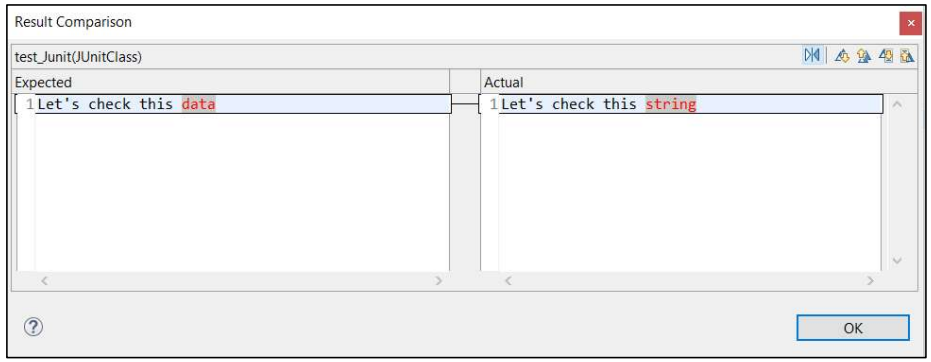
To see the result of a JUnit test, Eclipse uses the JUnit view which shows the results of the tests.



Test result is a pass.



Test result is a fail.



Right-click > select Compare Result or double-click on the line

JUnit testing in Eclipse

Test Failures

- A test failure occurs when an assertFact() method call fails or when an assert statement fails.
  - Indicated by a java.lang.AssertionError in the output, indicating the expected and the actual result.
- A test failure also occurs when a unit test method throws an exception.
  - Indicated by the appropriate exception in the output.
- Execution flow in case of failure:
  - The current unit test method is aborted.
  - Execution continuing with the next unit test method, if any.

JUnit testing in Eclipse

Junit Assert

Assert is a method useful in determining Pass or Fail status of a test case, The assert methods are provided by the class org.junit.Assert which extends java.lang.Object class.

Methods	Descriptions
assertEquals(Object expected, Object actual) assertEquals(String message, Object expected, Object actual)	Compares two values for equality. The test passes if the values are equal. Objects are compared for object value equality by calling the equals() methods.
assertEquals(Type1 expected, Type1 actual, Type1 delta) assertEquals(String message, Type1 expected, Type1 actual, Type1 delta)	Type1 is either float or double. Floating point values are compared for equality within a delta.
assertTrue(boolean condition) assertTrue(String message, boolean condition)	The test passes if the Boolean condition expression evaluates to true.
assertFalse(boolean condition) assertFalse(String message, boolean condition)	The test passes if the Boolean condition expression evaluates to false.

JUnit testing in Eclipse

Junit Assert (Cont.)

Methods	Descriptions
assertNull(Object object) assertNull(String message, Object object)	The test passes if the reference obj is null.
assertNotNull(Object object) assertNotNull(String message, Object object)	The test passes if the reference obj is not null.
assertSame(Object expected, Object actual) assertSame(String message, Object expected, Object actual)	The test passes if the expression (expected == actual) is true, i.e. the references are aliases, denoting the same object.
assertNotSame(Object unexpected, Object actual) assertNotSame(String message, Object unexpected, Object actual)	The test passes if the expression (expected == actual) is false, i.e. the references denote different objects.
fail() fail(String message)	The current test is forced to fail. See section on exception handling in unit testing.

JUnit testing in Eclipse

Granularity of unit tests

- A unit test should only test conditions that are related to one piece of functionality.
- A unit test fails if an assertFact() method call (or an assert statement) fails, and the remaining conditions are not checked.
  - If the remaining conditions pertain to unrelated functionality, this functionality will not be tested -- leading to bad test design.
  - Factoring test conditions into appropriate unit tests ensures that these conditions will be tested -- leading to a better test design.
- A unit test should be structured in such a way that if a test condition fails, the remaining conditions will always fail.

References

Tahchiev, P., Leme, F., Massol, V., Gregory, G. *Junit in Action 2nd edition.*

Mughal, K.A. JUnit4 - Unit Testing Made Easy. http://www.ii.uib.no/~khalid

Joshi, M. How to write JUnit test cases. https://www.browserstack.com/guide/how-to-write-junit-test-cases