JUnit

Introduction of Testing

- Testing is the process of finding differences between the expected behavior specified by system models and the observed behavior of the implemented system. Unit testing finds differences between a specification of an object and its realization as a component
  - Three testing methods: Structural testing finds differences between the system design model and a subset of integrated subsystems.
  - Functional testing finds differences between the use case model and the system.
  - performance testing finds differences between nonfunctional requirements and actual system performance.

An Overview of Testing

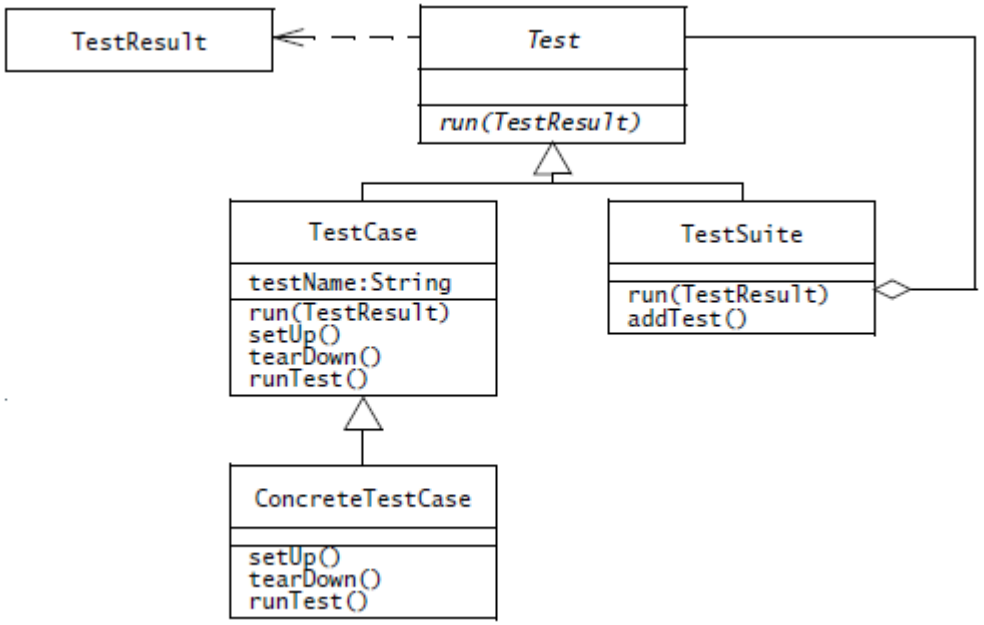
- **Reliability** is a measure of success with which the observed behavior of a system conforms to the specification of its behavior
- **Software reliability** is the probability that a software system will not cause system failure for a specified time under specified conditions
- **Failure** is any deviation of the observed behavior from the specified behavior
- **erroneous state** (also called an *error*) means the system is in a state such that further processing by the system will lead to a failure, which then causes the system to deviate from its intended behavior
- A **fault**, also called “defect” or “bug,” is the mechanical or algorithmic cause of an erroneous state

An overview of testing activities

- **Test planning** allocates resources and schedules the testing. This activity should occur early in the development phase so that sufficient time and skill is dedicated to testing. For example, developers can design test cases as soon as the models they validate
- **Usability testing** tries to find faults in the user interface design of the system. Often, systems fail to accomplish their intended purpose simply because their users are confused by the user interface and unwillingly introduce erroneous data.
- **Unit testing** tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
- **Integration testing** is the activity of finding faults by testing individual components in combination. **Structural testing** is the culmination of integration testing involving all components of the system. Integration tests and structural tests exploit knowledge from the SDD (System Design Document) using an integration strategy described in the Test Plan (TP).
- **System testing** tests all the components together, seen as a single system to identify faults with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively: **Functional testing** tests the requirements from the RAD and the user manual.
- **Performance testing** checks the nonfunctional requirements and additional design goals from the SDD(System Design Document). Functional and performance testing are done by developers.
- **Acceptance testing** and **installation testing** check the system against the project agreement and is done by the client, if necessary, with help by the developers.

JUnit Introduction

- JUnit is a framework for writing and automating the execution of unit tests for Java classes.

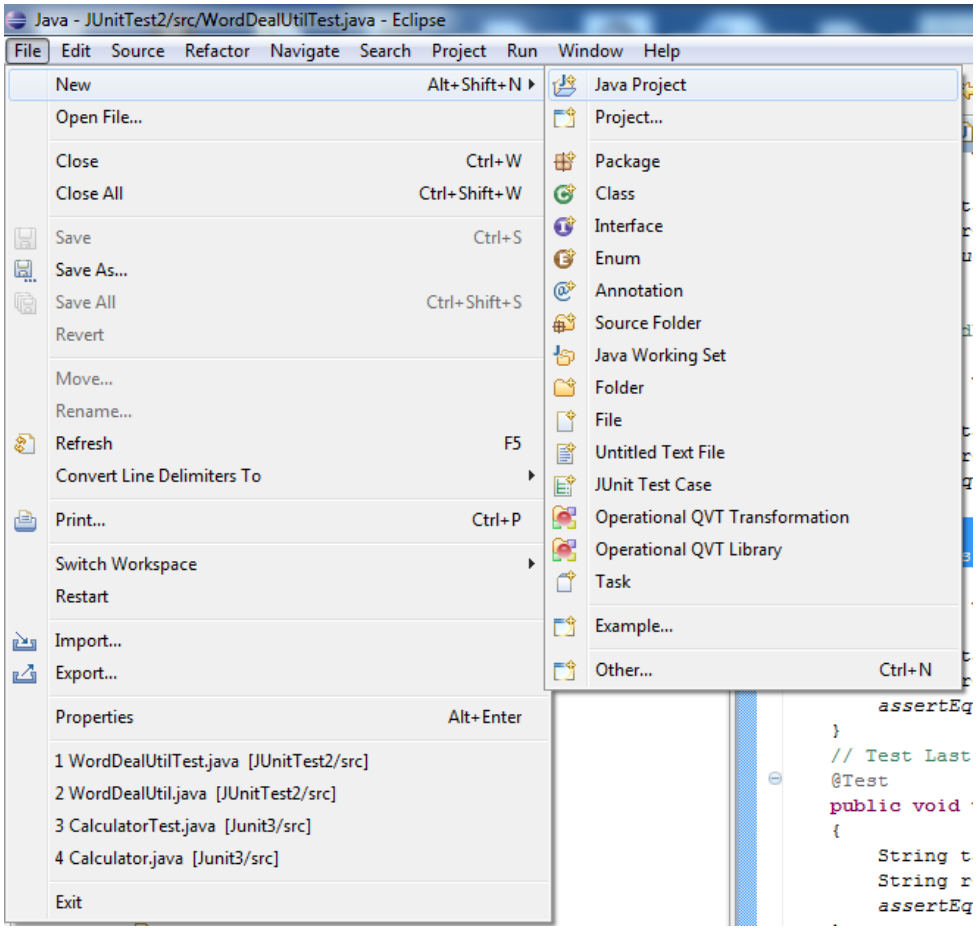


JUnit Introduction

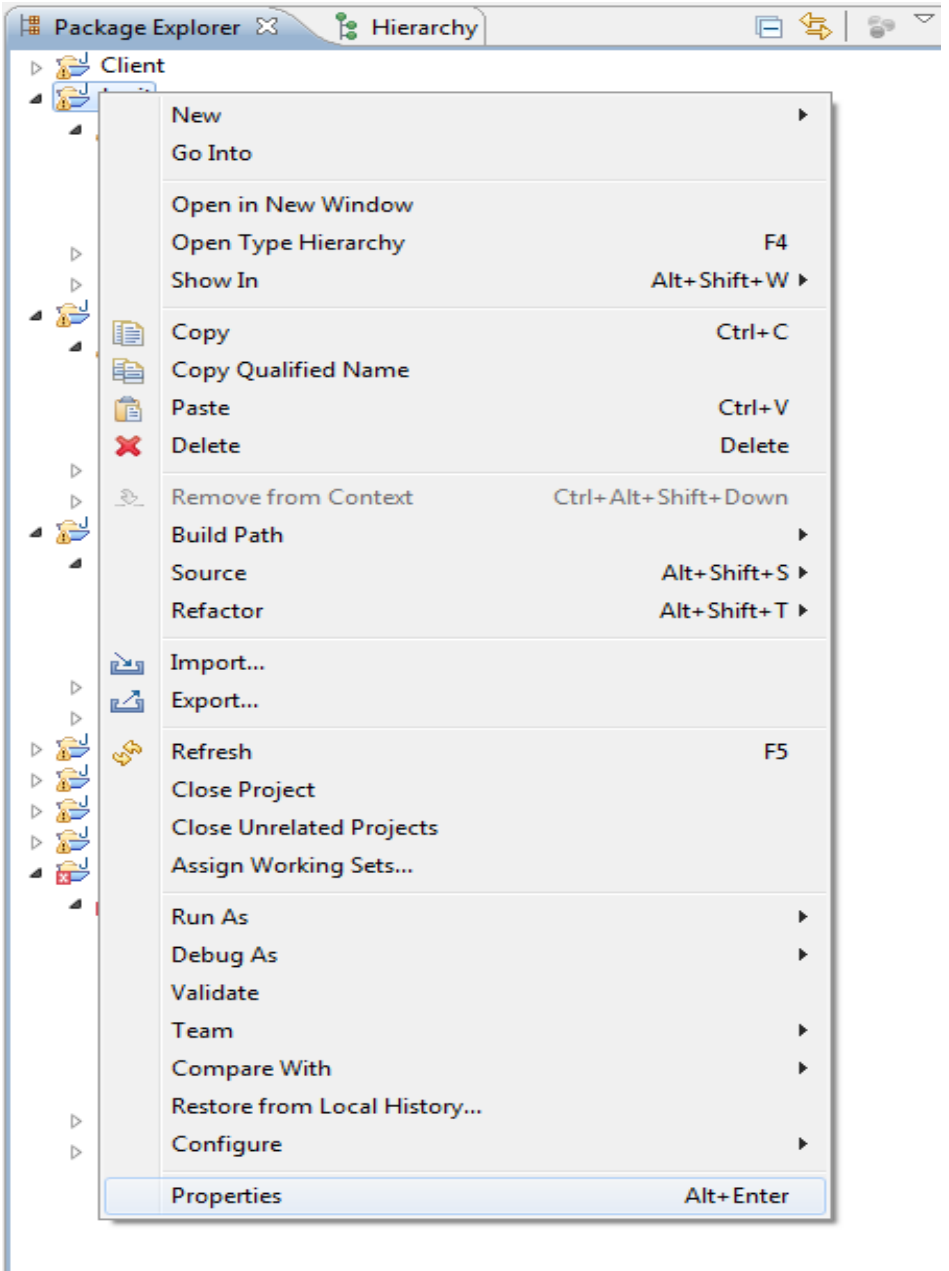
- ConcreteTestCase Class: the setUp() and tearDown() methods of the concrete test case initialize and clean up the testing environment. runTest() method includes the actual test code that exercises the class under test and compares the results with an expected condition.
- TestResult: report the testing results, either success or failure.

How to start JUnit

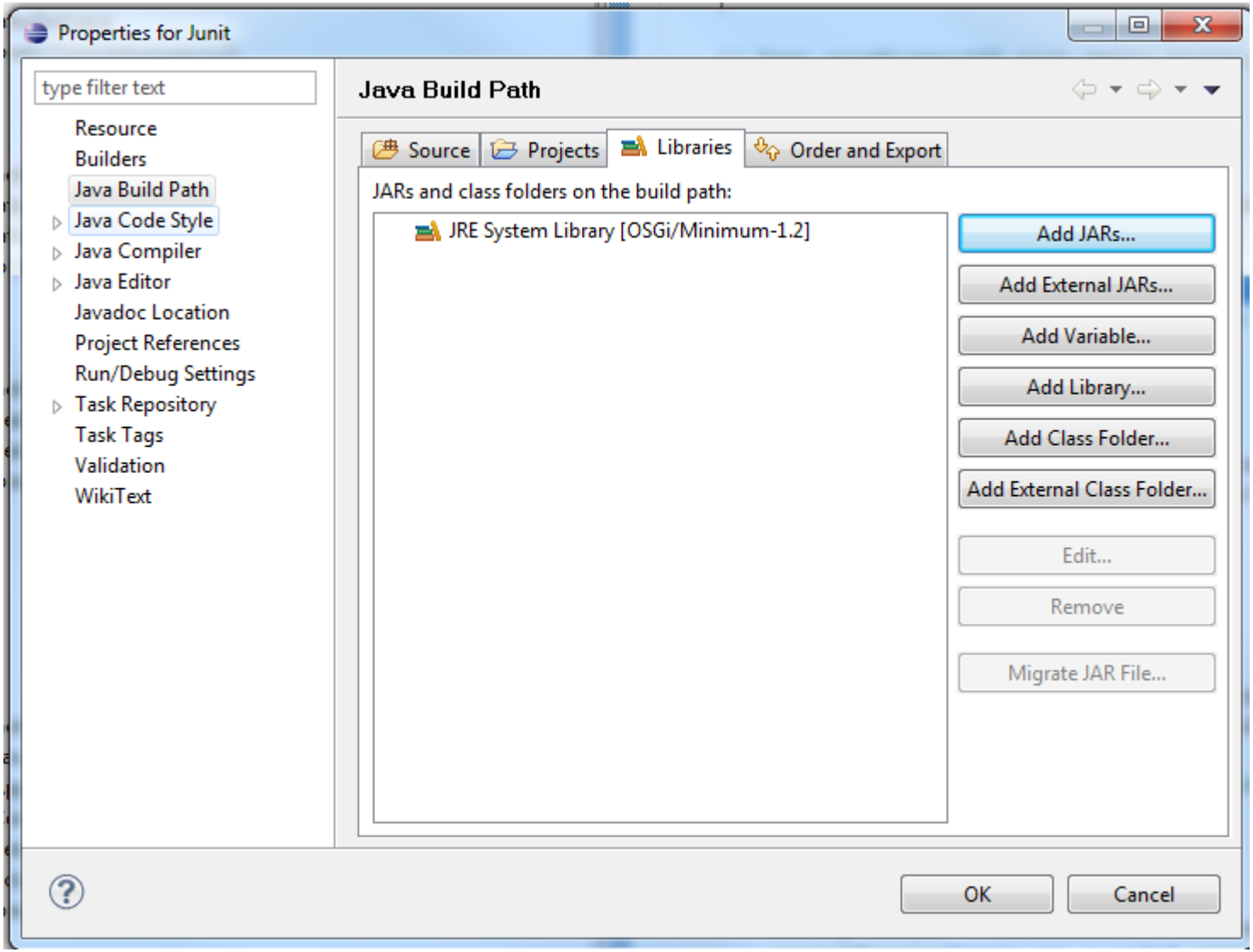
- Using Eclipse create a new project (for example: Calcuator)



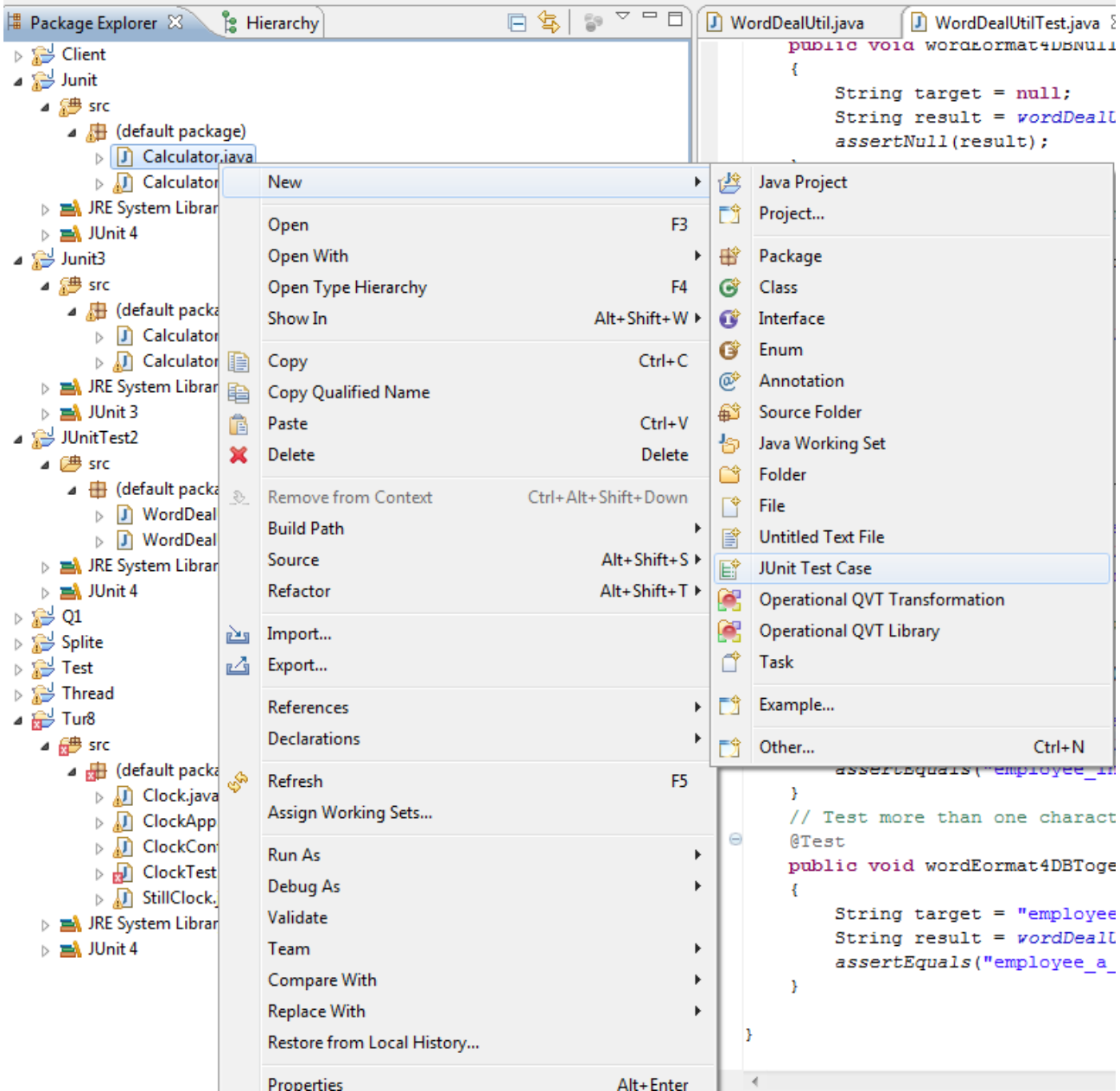
- Right Click the project name and go to the properties

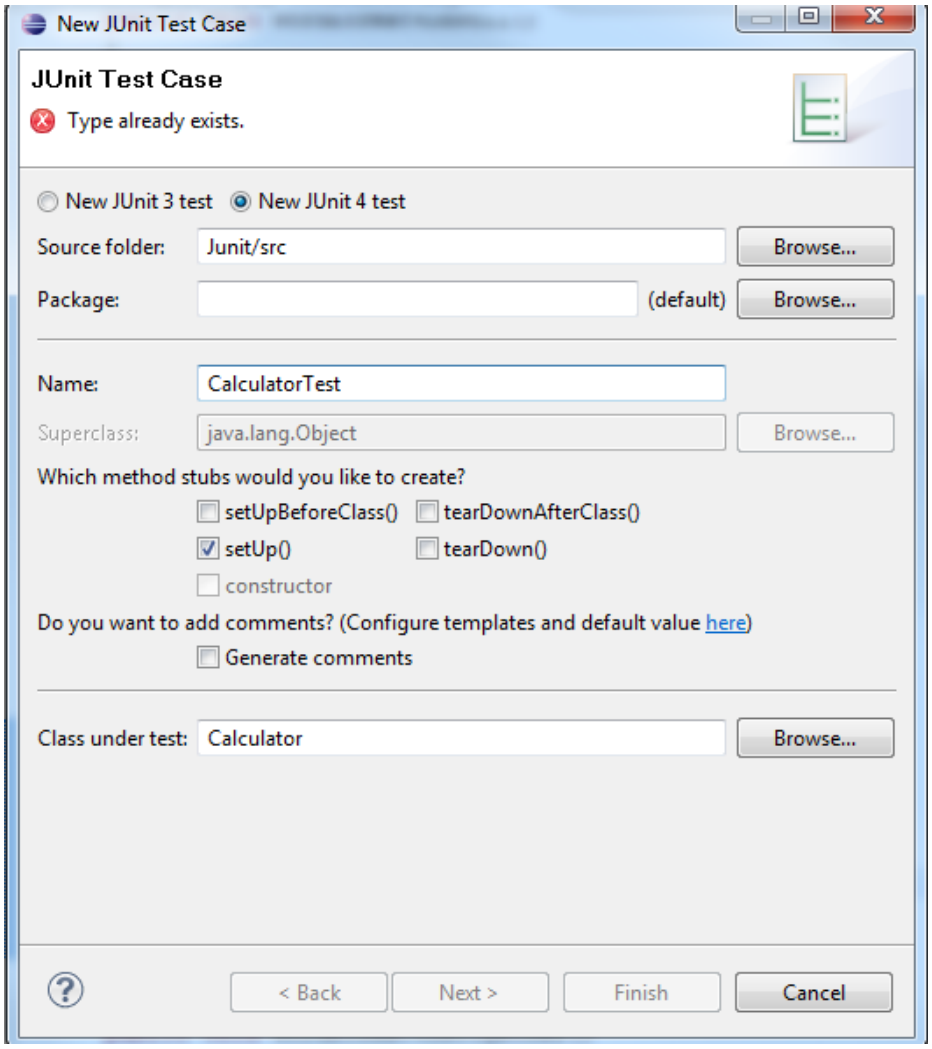


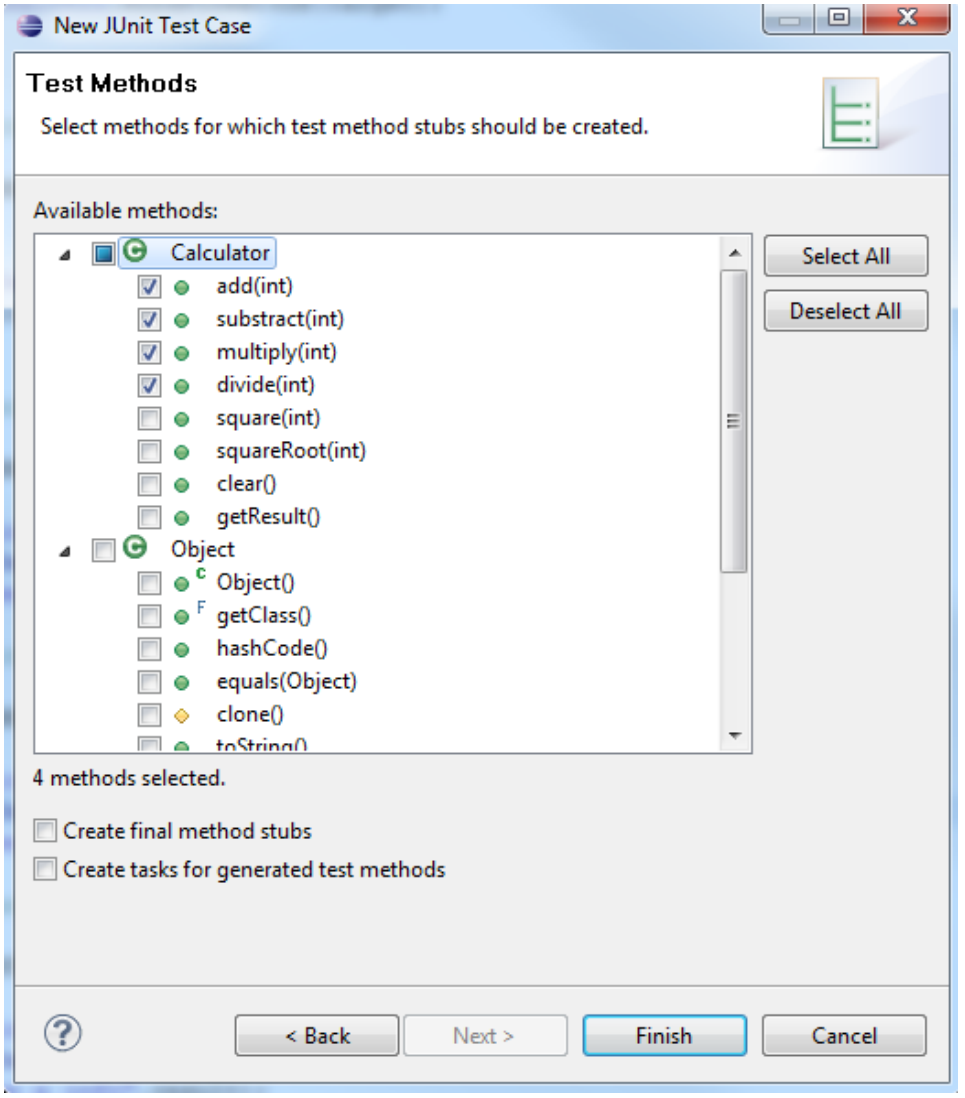
- Go to the build path -> Add library -> Junit-> JUnit4



- Calculator-> New->Junit Test Case
-




New JUnit Test Case
JUnit Test Case
Type already exists.
New JUnit 3 test
New JUnit 4 test
Source folder:
Junit/src
Browse...
Package:
(default)
Browse...
Name:
CalculatorTest
Superclass:
java.lang.Object
Browse...
Which method stubs would you like to create?
setUpBeforeClass()
tearDownAfterClass()
setUp()
tearDown()
constructor
Do you want to add comments? (Configure templates and default value here)
Generate comments
Class under test:
Calculator
Browse...
< Back
Next >
Finish
Cancel


New JUnit Test Case
Test Methods
Select methods for which test method stubs should be created.
Available methods:
Calculator
add(int)
substract(int)
multiply(int)
divide(int)
square(int)
squareRoot(int)
clear()
getResult()
Object
Object()
getClass()
hashCode()
equals(Object)
clone()
Select All
Deselect All
4 methods selected.
Create final method stubs
Create tasks for generated test methods
< Back
Next >
Finish
Cancel

Example of Calculator

- import static org.junit.Assert.*;
- import org.junit.Before;
- import org.junit.Test;
- import org.junit.Before;
- import org.junit.Ignore;
- import org.junit.Test;

- public class CalculatorTest {
  - private static Calculator calculator = new Calculator();
  - @Before
  - public void setUp() throws Exception {
    - calculator.clear();
  - }
  - @Test
  - public void testAdd() {
    - calculator.add(2);
    - calculator.add(3);
    - assertEquals(5, calculator.getResult());
  - }
- }