JUnit

Unit Test Framework for Java

The unit-test case

- The structure of unit-test case
  - Create the object
  - Invoke the methods
  - Verify the results
  - If the results are not matching ,modify the code and again run the method.
  - Proceed with more test methods

Unit-Test Framework

- The unit-testing framework automates the testing process.
  - Allows to write test case methods as test cases
  - Configure the test methods
  - With annotations
  - Executes the test cases
  - Verify/Assert the test results
  - Combine multiple test cases
  - Generate the test results.

Enter xUnit standard

- The frameworks based on a design by Kent Beck.
- The **xUnit** frameworks allow testing of different elements (units) of software, such as functions and classes.
- provide an automated solution to execute the same tests many times, and no need to remember what should be the result of each test.

JUnit as xUnit Framework

- JUnit originally written by Erich Gamma and Kent Beck.(junit.org)
- JUnit is an open source unit-testing framework for java used to write and run repeatable tests.
- It follows the xUnit architecture for unit testing frameworks.

JUnit Features

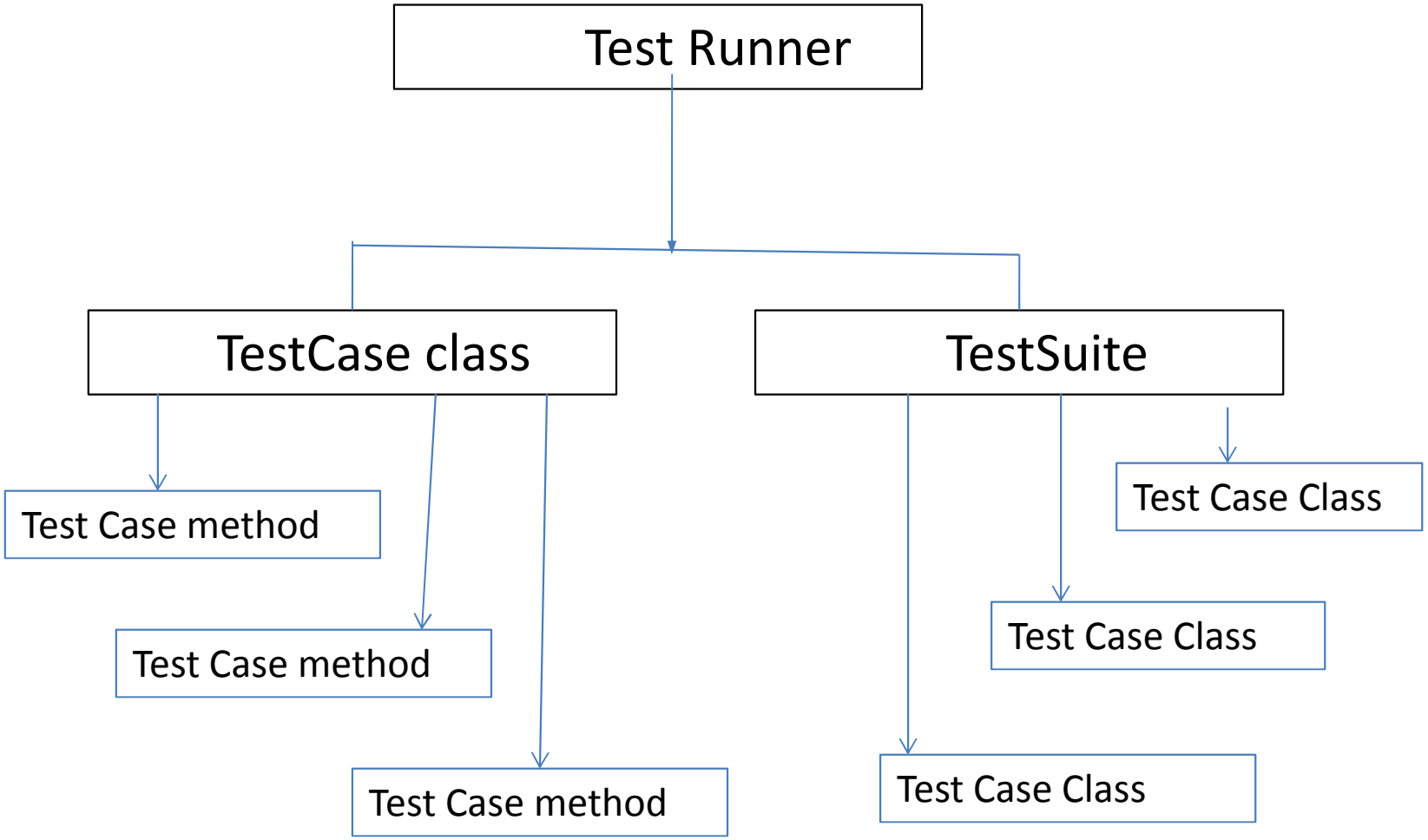
The JUnit features include:

- Define the test cases by extending the TestCase class or decorating test methods with annotations
- Provide test fixture to execute the test cases
- The Test fixtures for sharing common test data
- Supports the assertions for testing/verifying expected results
- Test suites for easily organizing and running tests
- Graphical and textual **Test Runners**

Test Case in JUnit4.0 onwards

- The JUnit4 recognizes any test method decorated with '@Test' annotation as test case in any class.
- You can write multiple test methods in single test class.
- Inside each test method the test case result can be verified with assertions supported by JUnit.

Test implementations



TestCase class Cycle

- It defines the test fixture to run multiple tests.
- Supports fixture initialization and closedown with two methods decoreateed with annotations
- **setUp**() and **tearDown**() to provide your own mechanism of initialization and destroying data.
- Static methods with annotations at class define the class level initializations and tear down.
- The test-runner runs a collection of test case methods.

TestSuite class

- The TestSuite is a container of multiple Test case classes and other TestSuite instances.
- The test-run is invoked by the framework

Test Success or Failure

- The success or failure of a test method is defined by the outcome of a comparison method that compares(asserts) the results.
- If the comparison (assertion) returns false , an **AssertionFailedError** exception is thrown which indicates the failure of the test.
- If no exception occurs in test method it is treated as success.

A set of assertions

- The comparison of test results is done by a set of assert methods that returns the result
- The class **Assert** provides **a** set of static methods to assert the values of different types.
- Messages are displayed only when an assert fails.

Assert methods

- **assertEquals**(boolean expected, boolean actual)
- **assertEquals**(java.lang.String message, byte expected, byte actual)
- **assertNotNull**(java.lang.Object object)
- **assertSame**(java.lang.Object expected, java.lang.Object actual)
- **assertNotSame**(java.lang.Object expected, java.lang.Object actual)

Annotated test-case


```
public class AccountTest {

  @Test
  public void testAccountDeposit()
  {
     Account ac= new Account(12000);
     ac.deposit(1000);
     Assert.assertEquals(11000, ac.getBalance());
  }
```

If the assert method throws exception it is treated as test failure otherwise success.

Test Configuration

- To specify the test timeout specify with the annotation as

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

- To expect an exception in the test method

```
@Test(expected=ArrayIndexOutOfBoundsException.class)
public void testBounds() {
new ArrayList<Object>().get(1);
}
```

- If the method doesn't throw an exception of this type or if it throws a different exception than the one declared, the test is treated as failure.

Ignore the tests

- To temporarily disable a test or a group of tests use ignore annotation.
- Methods annotated with Test and @Ignore will not be executed as tests.
- A class containing test methods can also be annotate d with @Ignore and none of the containing tests will be executed.
- @Ignore @Test public void notYetRun() { ...}
- @Ignore public class TryNotMe {..}

Assertions

- Assert methods include:
  - *assertEquals(expected,actual*)
  - *assertTrue*(boolean)
  - *assertFalse* (boolean)
  - *assertNull*(object)
  - *assertNotNull*(object)
  - assertSame(firstObject, secondObject)
  - assertNotSame(firstObject, secondObject)
  - assertArrayEquals(expected, actual)

Test Fixture methods


```
public class AccountTest {
  @Test
  public void testAccountDeposit()
  {  …..//some code to test   }
   @BeforeClass
   public static void initClass()
    {  //class level setup code  }
   @AfterClass
  public static void closeTestClass()
   {   //class level close     }
   @Before
  public void initTestSetUp()
   {    //set up for test}
   @After
  public void tearDown()
  { //close test setup }
```

Fixtures in the test case

- Test Setup:
  - Use the **`@Before`** annotation on a method containing code to run before each test case.
- Test Teardown:
  - Use the **`@After`** annotation on a method containing code to run after each test case.
  - These methods will run even if exceptions are thrown in the test case or an assertion fails.
- It is allowed to have any number of these annotations.
  - All methods annotated with **`@Before`** will be run before each test case, but they may be run in any order.

Static Fixtures at class level

- Test Class Setup:
  - Use the **@BeforeClass** annotation on a method containing code to run once before when the test class starts running the tests.
- Test Class Teardown:
  - Use the **@ AfterClass** annotation on a method containing code to run after all the test cases have been finished.

Test Suite annotation


```
@RunWith(Suite.class)
//Add multiple test implementations here to run
@SuiteClasses({com.data.test.TestBackupFirst.class,com.data.test.UserTest.class,c
    om.data.test.TestBackupSecond.class})

public class RunAllTests {

public static Test suite() {
 TestSuite suite = new TestSuite("Test for com.data.test AllTests");
 return suite;
}
```

Explicitly fail the test

- Static method : class org.junit.Assert.fail(): Fails a test with no message.
- Static method : class org.junit.Assert.fail(String errorMessage): Fails a test with given message.