4フェイズ・テスト

```
public class FourPhaseTest {
  @Test
  public void testCase()
            throws Exception {
    // 1.初期化
    Calc sut = new Calc();
    int expected = 7;
    // 2.テストの実行
    int actual = sut.add(3, 4);
    // 3.アサーション
    assertThat(actual,
                  is(expected));
    // 4.（必要ならば）終了処理
    sut.shutdown();
  }
}
```

例外送出のテスト

```
@Test(expected=Exception.class)
public void testCase() {
}
```

タイムアウト

```
@Test(timeout=1000L)
public void testCase() {
}
```

アノテーション

@Test	テストケース
@Ignore	テストの実行を除外
@Before	初期化
@After	後処理
@BeforeClass	初期化/クラス毎
@AfterClass	後処理/クラス毎
@RunWith	テストランナー

JUnit 実践入門 チートシート

アサーション

```
assertThat(actual,is(expected));
```

```
fail("未実装");
```

CoreMatchers

is	equalsによる比較
nullValue	null
not	Matcherの否定
notNullValue	非null
sameInstance	同一インスタンス
instanceOf	型

JUnitMatchers

hasItem	要素
hasItems	要素（複数）

ルール

TemporaryFolder	一時ファイル
ExternalResource	外部リソース
Verifier	事後検証
ErrorCollector	エラーの収集
ExpectedException	例外の検証
Timeout	タイムアウト
TestWatcher	テストの監視
TestName	テストケース名

```
public class RuleTest {
  @Rule
  public TestName testName
              = new TestName();

}
```

構造化テスト

```
@RunWith(Enclosed.class)
public class EnclosedTest {
  public static class XXの場合 {
    @Before
    public void setUp() {
    }
    @Test
    public void testCase() {
    }
  }
  public static class YYの場合 {
    @Before
    public void setUp() {
    }
    @Test
    public void testCase() {
    }
  }
}
```

パラメータ化・テスト

```
@RunWith(Theories.class)
public class ParameteriedTest {
  @DataPoints
  public static int[] PARAMS
              = {1, 2, 3, 4};
  @Theory
  public void test(int x) {
  }
}
```

カテゴリ化テスト

```
@Category(DbTests.class)
public class CategoriedTest {
  @Test
  @Category(SlowTests.class)
  public void testCase() {
  }
}
```

mockito を利用したモック/スタブ

```
@Test
public void 戻り値を返すスタブ() throws Exception {
  List<String> stub = mock(List.class);
  when(stub.get(0)).thenReturn("Hello");
  assertThat(stub.get(0), is("Hello"));
}
```

```
@Test(expected = IndexOutOfBoundsException.class)
public void 例外を送出するスタブ() throws Exception {
  List<String> stub = mock(List.class);
  when(stub.get(0)).thenThrow(new IndexOutOfBoundsException());
  stub.get(0);
}
```

```
@Test
public void モックによる検証() throws Exception {
  List<String> mock = mock(List.class);
  mock.clear();
  mock.add("JUnit");
  mock.add("JUnit");
  verify(mock).clear();
  verify(mock, times(2)).add("JUnit");
  verify(mock, never()).add("mockito");
}
```