Թեստավորումը օգտագործվում է բագերի բացահայտման համար։

Առանցքային գաղափարը Test Case – ն է՝ թեստավորման սցենար, որը նկարագրում է այն քայլերը որոնք անհրաժեշտ են թեստավորվող ֆունկցիայի ստուգման համար։ Սխեման հետևյալն է

Action -> Expected Result -> Test Result

Կան թեստավորման տարբեր մակարդակներ, որքան թեստավորման մակարդաը բարձրանում է այնքան այն հեռանում է կոդից։

* մոդուլային թեստավորում, ստուգում է առանձին մոդուլները՝ կլասսներ մեթոդներ JUnit գրադարանի միջոցով։ Ամենացածր մակարդակի թեստավորումն է։
* ինտեգրացիոն թեստավորում՝ մի քանի մոդուլների ընդհանուր աշխատանքի թեստավորում
* համակարգային, սիստեմային թեստավորում, ամենաբարձր մակարդակի թեստավորում։

Developer – ը զբաղվում է մոդուլային թեստավորմամբ։

Մոդուլային թեստավորումը (unit) դա source կոդի առանձին հատվածների աշխատունակության ստուգումն է:

Թեստը՝

* գեներացնում է որոշակի տվյալներ, անում է նախնական գործողություններ
* իրականացնում է թեստավորվող մեթոդը, որի սպասելի արդյունքը ակնհայտ է
* համադրում է ստացված արդյունքը սպասելի արդյունքի հետ, պրոցեսը կոչվում է assert։ Եթե թեստը fail է լինում գեներացվում է AssertionError exeption:

Թեստի սցենարը test case – ը իրենից ներկայացնում է մեթոդ որը պետք է նշել @Test անոտացիայով։

Թեստ մեթոդների և կլասսների համար կան անունների կոնվենցիա, բոլորը պետք է վերջանան Test – ով։ Թեստ մեթոդները պետք է իրենց անվան մեջ պարունակեն should բառը։

@Test – ը սահմանում է մեթոդը որպես թեստային, որը պարտադիր պետք է լինի public void, չպետք է լինեն ստատիկ։ Անոտացիան կարող է ունենալ պարամետրեր՝

* expected – նշվում է exeption – ը որը ակնկալվում է տեսնել, եթե exeption չեղավ ուրեմն թեստը համարվում է failed
* timeout – սահմանվում է առավելագույն ժամանակը կոդի իրականացման համար։

Juinit – ի կարևոր կլասս է Assert – ը որը համադրում է սպասելի արդյունքը փաստացի արդյունքի հետ։ (expected/actual)

Ունի ստատիկ մեթոդներ որոնք համադրում են ակնկալվող և փաստացի արդյունքները՝

assertTrue() – ստուգում է որ մեթոդը վերադարնի true

assertFalse() – ստուգում է որ մեթոդը վերադարնի false

assertEquals() – ակնկալվող արդյունքը հավասար է փաստացիին

assertArrayEquals()

assertNotEquals()

assertSame()

assertNotSame()

fail() – սրակ կանչի դեպքում թեստը fail է լինում

Թեստ մեթոդը կարող է ունենալ @Ignor անոտացիա որի դեպքում թեստը չի իրականացվի

Assert կլասսի փոխարեն կարելի է օգտագործել նաև Assume – ը որը ունի նույնանուն մեթոդներ։ Ի տարբերություն Assert – ի թեստի fail – ի դեպքում այն ուղակի ignore է լինում։

Մեոդները որպես օժանդակող սահմանելու համար կան անոտացիաներ՝

@BeforeClass – աշխատում է մեկ անգամ թեստ կլասսի համար, պետք է լինեն ստատիկ

@Before – աշխատում է յուրաքանչյուր թեստ մեթոդից առաջ

@After – աշխատում է յուրաքանչյուր թեստ մեթոդից հետո

@AfterClass – աշխատում է կլասսի բոլոր թեստերից հետո եթե դրանք բարեհաջող ավարտվել են

Թեստ կլասսըի կյանքի ցիկլերը՝

* @BeforeClass – statik մեթոդ է, դրա համար աշխատեում է մեկ անգամ կլասսի համար
* ամեն @Test մեթոդի համար ստեղծվում է կլասսի էկզեմպլյար
* @Before
* @Test
* @After
* @AfterClass

@BeforeClass և @Before մեթոդների մեջ կարելի է հայտարարել այն օբյեկտները որոնք օգտագործվում են բոլոր թեստ մեթոդների մեջ։

JUnit5 vs JUnit4 annotations

|  |  |
| --- | --- |
| @Test | @Test |
| @BeforeClass | @BeforeAll |
| @Before | @BeforeEach |
| @After | @AfterEach |
| @AfterClass | @AfterAll |
| @Ignore | @Disabled |
| NA | @TestFactory |

Թեստերի ժամանակ օգտագործվում են mock – երը։

Mock – ը օբյեկտ է որը իմիտացնում է իրական օբյեկտի վարքագիծը, նպատակը request response թեստավորումն է։

Mock – եր ստեղծել կարելի է WireMock – ի միջոցով։

Wiremock – ը opensource գործիք է որի միջոցով կարելի է սիմուլացնել HTTP requst response – ներ։ Կարելի է ստեղծել նաև վիրտուալ HTTP սերվերներ որոնք կարեղ են պատասխանել request – ին վերադարձնել համապատասխան կոդը, body – ն կամ header – ը։

Dependency` testImplementation group: 'com.github.tomakehurst', name: 'wiremock', version: '2.27.2', ext: 'pom'

Wiremock – ը և Junit – ը կարող են ինտեգրվել @Rule – անոտացիայի միջոցով, որը թույլ է տալիս Junit – ն կառավարել սերվերի կյանքի ցիկլը ամեն թեստից առաջ:

@Rule **public** **WireMockRule** wireMockRule = **new** **WireMockRule**(**int** port);

Default port` 8080

Հաջորդ քայլը stub (заглушка) կարգավորումն է: Սրա միջոցով սահմանվում են սպասելի request response – ները, կարելի է նաև սահմանել սահմանափակումներ: Օրինակ HTTP request ուղարկող կլիենտի համար կարելի է ստեղծել заглушка որը կսիմուլացնի սերվերի պատասխանը կոնկրետ request – ի դեպքում: Սրա միջոցով ստեղծվում են թեստ սցենարներ: