=== TDD ===

**Testy funkcjonalne**

Testy zakładające, że moduł jest „czarnym pudełkiem”, o którym wiadomo tylko, jak ma się zachowywać. Nazywamy je też testami czarnejskrzynki ( black-box testing).

**Testy strukturalne**

Testy skupiające się na wewnętrznej pracy modułu, a nie na jego kooperacji z innymi modułami. Nazywamy je też testami białej skrzynki ( white-box testing).

== TWORZENIE TESTÓW ==

* Alt+enter na nazwie klasy -> create test
* Annotacja @Test nad każdą metodą do testowania
* Testy jednostkowe powinny być tworzone do KAŻDEJ metody
* Testy jednostkowe powinny być tworzone do każdego wariantu metody (dla każdej możliwości if-a itp)

|  |  |
| --- | --- |
| Puste testy | Wstawiamy metodę fail()  @Test  public void testObjectManipulation() {  if(obj == null) {  fail("obj should not be null");  } |
| Niezależność testów | To co dzieje się w jednym teście nie może wpłynąć na inne testy |
| Jeden assert na test | Umieszczanie w tescie więcej jest złą praktyką |
| Metoda wywołana przed każdym testem | // JUNIT4  @Before  // JUNIT5  @BeforeEach  public void setup(){} |
| Metoda wywołana raz, przed wszystkimi testami | // JUNIT4  @BeforeClass  // JUNIT5  @BeforeAll  public void setup(){} |
| Metoda wywoływana po testach, musi być statyczna | // JUNIT4  @AfterClass  @After  // JUNIT5  @AfterEach  @AfterAll  public void setup(){} |
| Sprawdzanie równości | assertEquals(expected, actual);  AssertEquals używa wbudowanych w obiekty metod equals.  assertEquals(expected, actual, delta);  Delta okresla dopuszczalne odchylenie wyniku i jego dokładność. Jak mamy wynik w stylu 13.999999999999 != 14.00 wtedy dajemy delta = 0.01 i wszystko gra.  assertNotEquals();  Pass jeżeli nie sa równe.  assertArrayEquals();  Pass jeżeli długości array i ich zawratośc sa takie same. |
| Sprawdzanie booleanów | assertTrue();  assertFalse(); |
| Sprawdzanie nulla | assertNull();  assertNotNull(); |
| Sprawdzanie referencji do obiektu | assertSame();  assertNotSame();  W przeciwieństwie do assertEquals() sprawdza czy są to referencje do tych samych obiektów, a nie używa equals() do porównania ich. |
| WYJĄTKI | = JUNIT4 =  @Test(expected = KlasaWyjątku.class)  = JUNIT5 =  assertThrows(KlasaWyjątki.class, () -> {  // wyrażenie rzucające wyjątek  }); |
| IGNOROWANIE TESTÓW | = JUNIT 4 =  @Ignore  = JUNIT 5 =  @Disabled |
| NAZYWANIE KLAS TESTUJĄCYCH | = JUNIT 5 =  @DisplayName("nazwa klasy") |
| ASSERT THAT | Porównywanie wartości z ustalonym Matcherem - HAMCREST MATCHERS: assertThat();   * Dodać zalezności do hamcrest-core i hamcrest-library   Wygląda to np. tak:  assertThat(library.size(), CoreMathers.is(CoreMatchers.equalTo(10)));  Co po statycznych importach można uprościć na:  assertThat(library.size(), is(euqualTo(10)));   * INNE PRZYKŁADY:   assertNotSame(new Object(), new Object());  to to samo co:  assertThat(new Object(),  not(sameInstance(new Object())));  assertThat("Effective Java", constainsString("Java"));  assertThat("Effective Java", both(containsString("Java"))  .and(containsString("Effective")));  assertThat("Effective Java", allOf(containsString("Java"), containsString("Effective")));  assertThat("Effective Java", anyOf(containsString("Java"), containsString("Effective")));  assertThat("Effective Java", not(anyOf((containsString("Java"), containsString("Effective"))));  **Compose Matchers:**  assertThat("Effective Java", either(containsString("Java"))  .or(containsString("Effective")));  assertThat("Effective Java", either(containsString("Java"))  .or(equalTo("Effective Java")));   * Obsługa kolekcji   assertThat(bookTitles, hasItem("Effective Java"));  assertThat(bookTitles, hasItems("Effective Java", "Design Patterns"));  assertThat(bookTitles, hasItem(startsWith("Effective"));  assertThat(bookTitles, hasItem(endsWith("Java"))); |
| ŁACZENIE ASERCJI JUNIT 5 | assertAll("wiadomość",  () -> assertThat(pierwszy warunek),  .  .  .  () -> /\*ostatni warunek\*/  ); |
| POWTARZANIE TESTU JUNIT 5 | @RepeatedTest(5)  //powtórzy test 5 razy |
| ASUMPCJE | * Kod wykona się tylko przy spełnieniu założenia * pakiet Assume. * Np:   //ten kod się wykona zawsze:  assumeTrue(jakieśZałożenie);  //ten kod się nie wykona jeżeli będzie  false |
|  | assumeThat(boolean assumption, Executable executable);  /\* w executable umieszczamy wyrażenie lambda zawierające wszystkie nasze asercje. jeżeli warunek nie będzie spełniony to nie zostaną w ogóle sprawdzone \*/ |
| TESTY PARAMETRYZOWANE | === JUNIT 4 ===   * Przy definicji klasy testu musimy dać adnotację @RunWith(Parametrized.class) * Z adnotacją @Parameterized.Parameters tworzymy metodą zwracającą kolekcję parametrów.   Parametry te JUNIT potem wstrzyknie do odpowiednich metod przy tesowaniu. Na przykład:  //mamy trzy pola  private int amount;  private boolean isUserLogged;  private int expected;  //więc musimy zwrócić kolecję wypełniającą te pola:  @Parameterized.Parameters  public static Collection<Object> testConditions(){  return Arrays.asList(new Object[][]{  //amount,isUserLogged,expected  {100.00, true, 1100.00},  {200.00,true,200.00},  });  } |
| ŁĄCZENIE TESTÓW | == JUNIT 4 ==   * Annotacja nad klasą   @RunWith(Suite.class)  @SuiteClasses({klasaTest1.class, klasaTest2.class}) |
| TESTY INTEGRACYJNE | * Stawia kontekst Springa, daje dostęp np. do plików konfiguracyjnych * Annotacje:   //jUnit4  @RunWith(SpringRunner.class)  @SpringBootTest  //jUnit5  @ExtendWith(SpringExtension.class)  @SpringBootTest |