Dubluri de testare

ALIN ZAMFIROIU

Recapitulare

- Testing
- Unit testing
- Junit
- Test
- TestCase
- TestSuite
- Assertion
- Right-BICEP
- CORRECT

Dubluri de testare

- În testarea automată este obișnuită folosirea obiectelor care arată și se comportă ca echivalentele lor de producție, dar sunt de fapt simplificate. Acest lucru reduce complexitatea, permite verificarea codului independent de restul sistemului și, uneori, este chiar necesar să se efectueze teste de auto-validare. Un termen generic folosit pentru aceste obiecte este **dublură de testare**.

- O dublură de testare este pur și simplu un alt obiect care se potrivește cu interfața colaboratorului necesar și poate fi trecut în locul său. Există mai multe tipuri de dubluri de testare.

Dubluri de testare

- **Dummy object** – un obiect care respecta interfața dar metodele nu fac nimic sau null.
- **Stub** – Spre deosebire de Dummies, metodele dintr-un Stub vor întoarce răspunsuri conservate / hardcodate.
- **Spy** – este un Stub care gestionează și contorizează numărul de apeluri.
- **Fake** – este un obiect care se comportă asemănător cu unul real, dar are o versiune simplificată.
- **Mock** – diferit de toate celelalte.

Obiecte Dummy

- **Dummy object** – un obiect care respectă interfața, dar metodele nu fac nimic sau returnează 0 sau null.
- Când trebuie să folosim obiectul real, de fapt folosim un obiect dummy.
- Aceste duble sunt folosite atunci când nu trebuie să apelăm metodele din acel obiect. Pentru că nu fac nimic.

Obiecte Dummy

```
@Test
public void test() {
    Companie company=new Companie("Company", new PersoanaDummy(), 0);

    List<IPersoana> lista=new ArrayList<>();
    lista.add(new PersoanaDummy());
    lista.add(new PersoanaDummy());
    lista.add(new PersoanaDummy());
    company.setSalariati(lista);

    assertEquals(3, company.getNumarSlariati());
}
```

Stub

- **Stub** – metodele de la un Stub vor returna răspunsuri conservate / hardcodate.

```
public int getVarsta(){
    return 33;
}
```

- În acest fel, putem folosi aceste obiecte cu apeluri reale.

Stub

```
@Test
public void test_verificareLegalitate() {
    IPersoana persoana=new PersoanaStub("Nume Prenume","43");
    Companie c=new Companie("Companie",persoana,1000);
    assertTrue(c.verificareLegalitate());
}
```

Fake

- **Fake** – este un obiect care se comportă ca unul real, dar are o versiune simplificată.

```
@Override
public int getVarsta() {
    return valoareGetVarsta;
}
```

- De obicei, pentru un fake, putem stabili ce valoare ar trebui să se întoarcă. Nu va fi o valoare hardocdată.

Spy

- **Spy**– este un Stub sau Fake care gestionează și numărul de apeluri realizate pentru metodele acestor obiecte.

```
public int getVarsta() {
    numberGetVarsta++;
    return 33;
}
```

```
public int getVarsta() {
    numberGetVarsta++;
    return valoareGetVarsta;
}
```

Mock

- **Mock** – diferit de toate celelalte, dar funcționează similar.

Mock testing

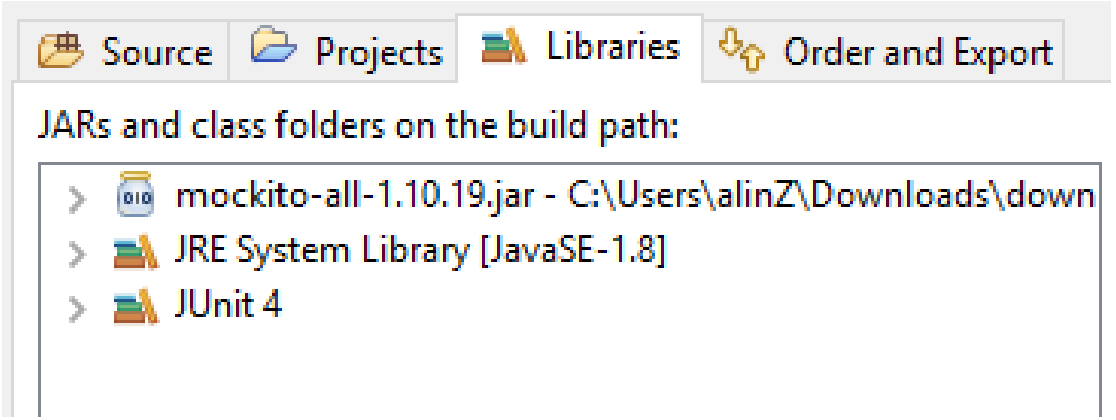
- Mock testing – Este utilizat atunci când dorim ca metoda testată să nu fie influențată de referințe externe.

- Mock object (obiect mock-uit) – un obiect care simulează comportamentul unui obiect real, însă într-un mod controlat.

- Cele mai folosite framework-uri sunt **Mockito**, **EasyMock**, etc

Mockito

- Se descarcă jar-ul și se adaugă la proiectul curent.



- Dacă nu îl găsiți pe google:

https://mvnrepository.com/artifact/org.mockito/mockito-all/1.10.19

Mockito

- Se creează un obiect mock, pe baza unei clase reale:

```
Persoana sot=mock(Persoana.class);
```

- Se setează comportamentul pentru metodele dorite:

```
when(sot.getVarsta()).thenReturn(23);

doReturn(3).when(sot).getVarsta();
```

Mockito

- Metode:
  - doReturn()
  - doAnswer()
  - when()
  - thenReturn()
  - thenAnswer()
  - thenThrow()
  - doThrow()
  - doCallRealMethod()

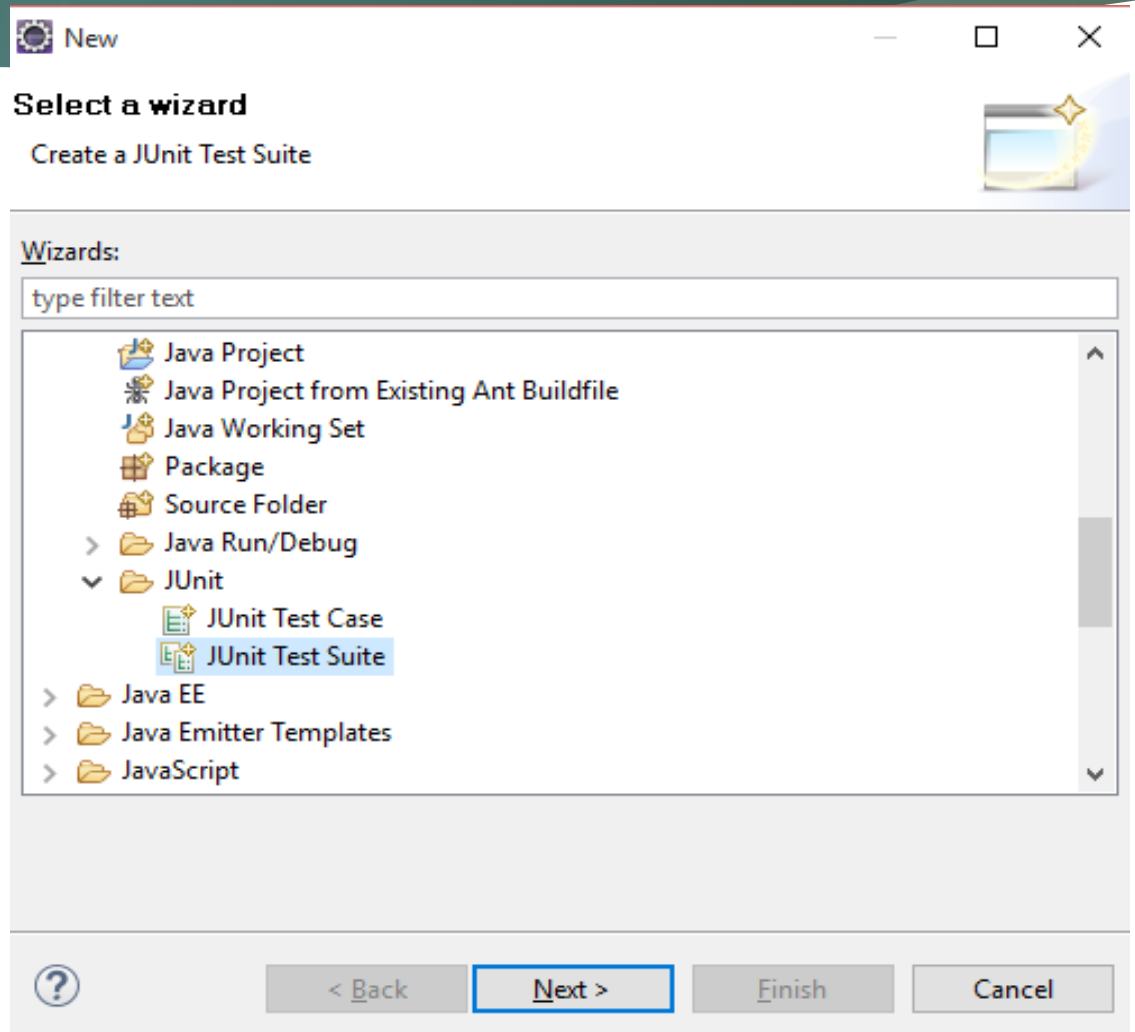
Mockito

```
@Test
public void test_verificareLegalitate_mockito() {
    Persoana p=mock(Persoana.class);
    when(p.getVarsta()).thenReturn(21);
    Companie c=new Companie("Companie", p, 1000);
    assertTrue(c.verificareLegalitate());
}
```

Utilizarea fișierelor pentru datele de test

- Se va realiza un singur TestCase cu metoda de assert apelată într-un loop.
- Pentru fiecare set de date de test din fișier se apelează metoda assert.
- Fișierul, sau fluxul este considerat un fixture și de aceea trebuie deschis în setUp și închis în tearDown.

Unit testing – TestSuite general



Custom TestSuite – JUnit 3

- TestCase-ul creat trebuie să extindă clasa TestCase:
  - **public class** UtilsTest **extends** TestCase{

- Se implementează constructorul cu un parametru de tipul String:

```
public UtilsTest(String method)
{
    super(method);
}
```

Custom TestSuite – JUnit 3

```
public static void main(String[] args) {
    TestSuite suite=new TestSuite();
    suita.addTest(new UtileTest("test_correct_division"));
    suita.addTest(new UtileTest("test_correct_sum"));
    TestRunner.run(suite);
}
```

Custom TestSuite JUnit 4

- Se implementează o clasă sau o interfață CustomSuite.

- Pentru fiecare test dorit să facă parte din acea suită se adaugă adnotarea:
  - @Category(CustomSuite.class)

Custom TestSuite – JUnit 4

- Pentru crearea unei suite custom , sunt incluse toate categoriile dorite.
- În cazul de față se include o singură categorie de teste. Testele din această categorie se regăsesc în două TestCase-uri **CompanyTest** și **PersonTest**:

```
@RunWith(Categories.class)
@IncludeCategory(CustomSuite.class)
@SuiteClasses({ CompanyTest.class, PersonTest.class })
public class NewSuiteTests {
}
```

JUnit5

JUnit5

- Parametrul opțional este pe ultima poziție.

```
@Test
@Ignore
public void testCuMesaj() {
    Persoana persoana1 = new Persoana("Nume Prenume", "1900807381167");
    assertTrue(persoana1.checkCNP(), "CNP incorect");
}
```

JUnit5

- Adnotările pentru structura unui test s-au schimbat

JUnit4	JUnit5 - Jupiter
@BeforeClass	@BeforeAll
@AfterClass	@AfterAll
@Before	@BeforeEach
@After	@AfterEach

JUnit5

- **assertThrows** – pentru testarea condițiilor de eroare.

```
@Test
public void test_checkCNP_conditii_de_eroare() {
    Persoana persoana1 = new Persoana("Nume Prenume", "2iili13144434");
    assertThrows(Exception.class, new Executable() {
        @Override
        public void execute() throws Throwable {
            assertTrue(persoana1.checkCNP());

        }
    });
}
```

JUnit5

- assertTimeout – pentru testarea timpului de rulare

```
@Test
public void test_performanta() {
    Persoana persoana = new Persoana("Nume Prenume", "2971023404186");
    assertTimeout(Duration.ofMillis(10), new Executable() {
        @Override
        public void execute() throws Throwable {
            assertEquals("F", persoana.getSex());
        }
    });
}
```

JUnit5

- @Tag – pentru suitele customizate

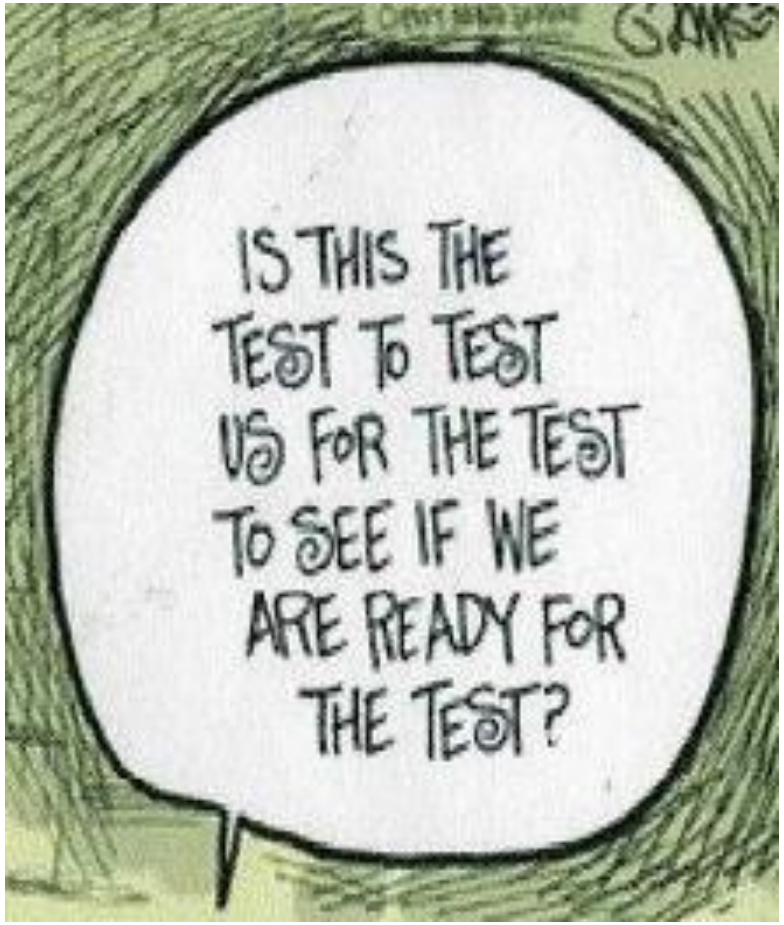
```
@Test
@Tag("Performance")
@Tag("Fast")
```

```
@Test
@Tag("Error")
@Tag("Slow")
```

Integration tests and Unit tests

Integration tests and Unit tests

The end!


IS THIS THE
TEST TO TEST
US FOR THE TEST
TO SEE IF WE
ARE READY FOR
THE TEST?