**Autori:Dobrin Alexandru Dan Ionut**

**Mantaluta Bogdan-Ionut**

**Mockito**

**Testarea unitara** a devenit un concept din ce in ce mai intalnit in aceasta epoca “Agile” in care traim, dezvoltandu-se astfel numeroase instrumente pentru a facilita acest proces.

Mockito este un astfel de instrument, el fiind un framework open source pentru Java,folosit in testarea automata si publicat sub licenta MIT.

Testele unitare sunt concepute pentru a testa comportamentul anumitor clase sau metode, fara a pune accent pe comportamentul dependentelor acestora.

Pentru a obtine Mockito, avem prezente doua sau mai multe modalitati de integrare in proiectul nostru depinzand de structura aleasa (Gradle / Maven).

**Mocking-ul** este un concept des intalnit si foarte important atunci cand vine vorba de testarea unitara in Mockito.Acesta reprezinta de fapt creearea unui obiect ce imlpementeaza comportamentul unui subsistem real, in moduri controlate.Pe scurt, mock-urile, sunt folosite ca si inlocuitor pentru o dependenta.

Pentru a creea o instanta de tip Mock, avem nevoie sa apelam metoda statica:

Mockito.mock()

Odata ce am importat pachetul:

**import** **static** org.mockito.Mockito.\*;

Un alt concept cheie, este acela de **Spy** (Spion), acesta fiind de fapt celalalt tip de test dublu pe care Mockito il creeaza.Spre deosebire de Mocking, creearea unui Spy necesita o instanta pe care sa o putem “spiona”.In mod implicit acest Spy gestioneaza toate apelurile metodei catre obiectul real si inregistreaza ce metoda a fost apelata si de ce parametrii s-a folosit aceasta.

Pentru a creea o instanta de tip Spy, avem nevoie sa apelam metoda statica spy() si sa ii transmite o instanta ca si parametru pe care sa o poata “spiona”.

Un **Stub** reprezinta un obiect ce implementeaza o interfata a unei componente, dar in loc sa returneze ceea ce componenta ar returna atunci cand este apelata, stub-ul poate fi configurat pentru a returna o valoare care se potriveste testului.Acest concept se regaseste sub mai multe forme, precum:

* Obiecte “pure”
* Obiecte “partiale”
* Metode
* Protocoale
* Servicii

de tip Stub.

Mock-urile sunt diferite de Spy-uri, deoarece atunci cand Mockito creeaza un obiect de tip Mock, acesta reprezinta o instanta, copie a unei clase total pregatita pentru a urmari interactiunile ce se vor face asupra sa.Pe de alta parte, un obiect de tip Spy va impacheta o instanta deja existenta, aceasta comportandu-se in continuare la fel ca si o instanta normala, singura diferenta este ca si aceasta v-a urmari toate interactiunile exercitate asupra sa.

Codul folosit in prezentare:

**Partea de cod propriu-zis:**

class Employee:

raise\_amt = 1.05

def init(self, first, last, pay):

self.first = first

self.last = last

self.pay = pay

@property

def email(self):

return '{}.{}@company.com'.format(self.first, self.last)

@property

def fullname(self):

return '{} {}'.format(self.first, self.last)

def apply\_raise(self):

self.pay = int(self.pay \* self.raise\_amt)

def monthly\_schedule(self, month):

response = requests.get(f'http://company.com/%7Bself.last%7D/%7Bmonth%7D%27)

if response.ok:

return response.text

else:

return 'Bad Response!'

**Partea de testare:**

import unittest

import mockito

from unittest.mock import patch

from employee import Employee

class TestEmployee(unittest.TestCase):

@classmethod

def setUpClass(cls):

print('setupClass\n')

@classmethod

def tearDownClass(cls):

print('teardownClass')

def setUp(self):

print('setUp')

self.emp\_1 = Employee('Alexandru', 'Dobrin', 3000)

self.emp\_2 = Employee('Bogdan', 'Mantaluta', 3500)

def tearDown(self):

print('tearDown\n')

def test\_apply\_raise(self):

print('test\_apply\_raise')

self.emp\_1.apply\_raise()

self.emp\_2.apply\_raise()

self.assertEqual(self.emp\_1.pay, 3150)

self.assertEqual(self.emp\_2.pay, 3675)

def test\_email(self):

print('test\_email')

self.assertEqual(self.emp\_1.email, 'Alexandru.Dobrin@company.com')

self.assertEqual(self.emp\_2.email, 'Bogdan.Mantaluta@company.com')

self.emp\_1.first = 'Dan'

self.emp\_2.first = 'Ionut'

self.assertEqual(self.emp\_1.email, 'Dan.Dobrin@company.com')

self.assertEqual(self.emp\_2.email, 'Ionut.Mantaluta@company.com')

def test\_fullname(self):

print('test\_fullname')

self.assertEqual(self.emp\_1.fullname, 'Alexandru Dobrin')

self.assertEqual(self.emp\_2.fullname, 'Bogdan Mantaluta')

self.emp\_1.first = 'Dan'

self.emp\_2.first = 'Ionut'

self.assertEqual(self.emp\_1.fullname, 'Dan Dobrin')

self.assertEqual(self.emp\_2.fullname, 'Ionut Mantaluta')

def test\_monthly\_schedule(self):

print("test\_monthly\_schedule")

with patch('employee.requests.get') as mocked\_get:

mocked\_get.return\_value.ok = True

mocked\_get.return\_value.text = 'Success'

schedule = self.emp\_1.monthly\_schedule('Septembrie')

mocked\_get.assert\_called\_with('http://company.com/Dobrin/Septembrie')

self.assertEqual(schedule, 'Success')

mocked\_get.return\_value.ok = False

schedule = self.emp\_2.monthly\_schedule('Noiembrie')

mocked\_get.assert\_called\_with('http://company.com/Mantaluta/Noiembrie')

self.assertEqual(schedule, 'Bad Response!')

def test\_monthly\_schedule2(self):

print("test\_monthly\_schedule2")

self.mockObject = mockito.mock()

# self.mockObject = mockito.spy(Employee)

mockito.when(self.mockObject).monthly\_schedule('Noiembrie').thenReturn('Bad Response!')

mockito.when(self.mockObject).monthly\_schedule('Septembrie').thenReturn('Success!')

schedule = self.mockObject.monthly\_schedule('Septembrie')

print(schedule)

schedule2 = self.mockObject.monthly\_schedule('Noiembrie')

print(schedule2)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()