Mock-объекты

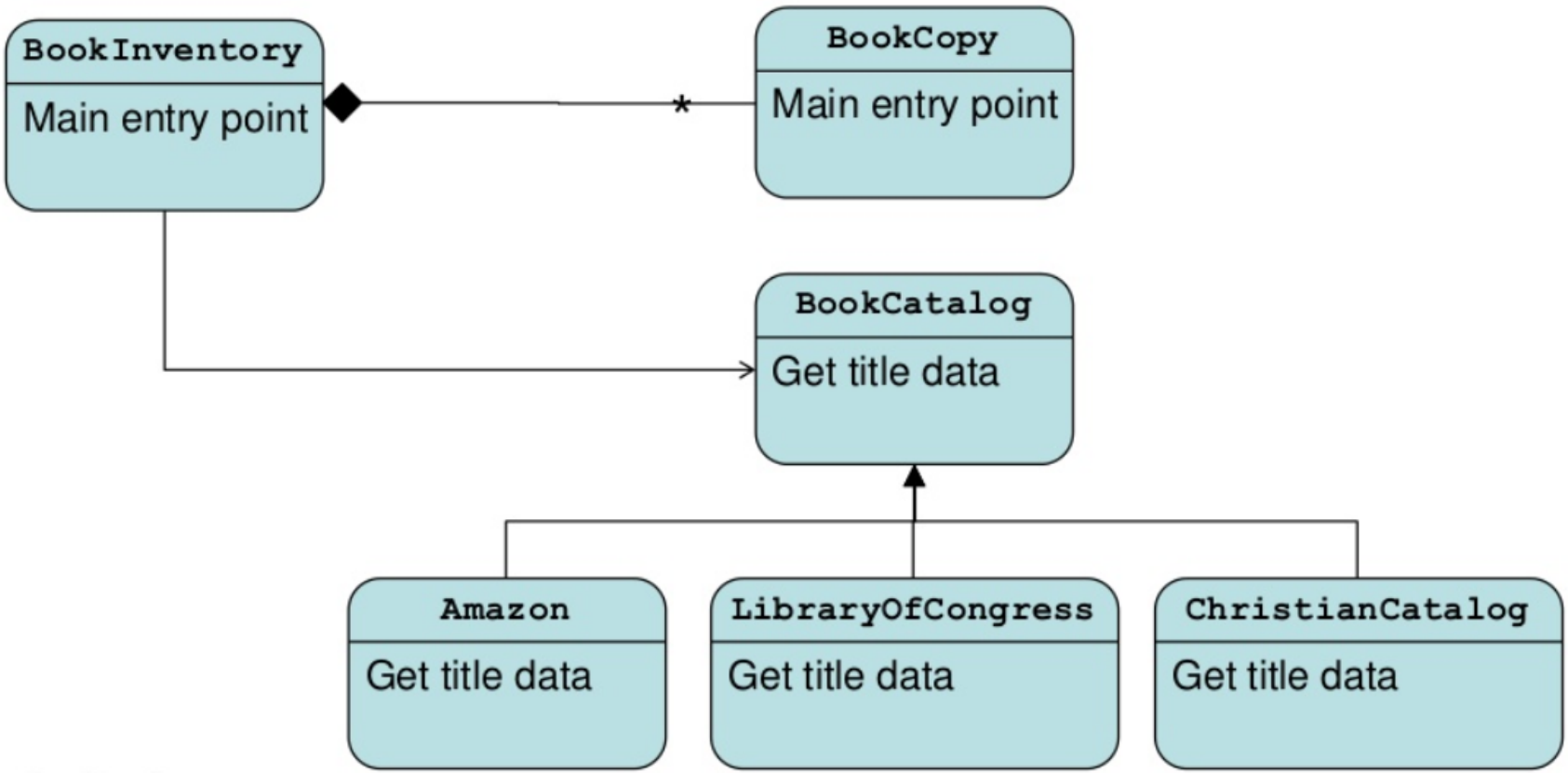
Юрий Литвинов
yurii.litvinov@gmail.com

27.02.2019г

Mock-объекты

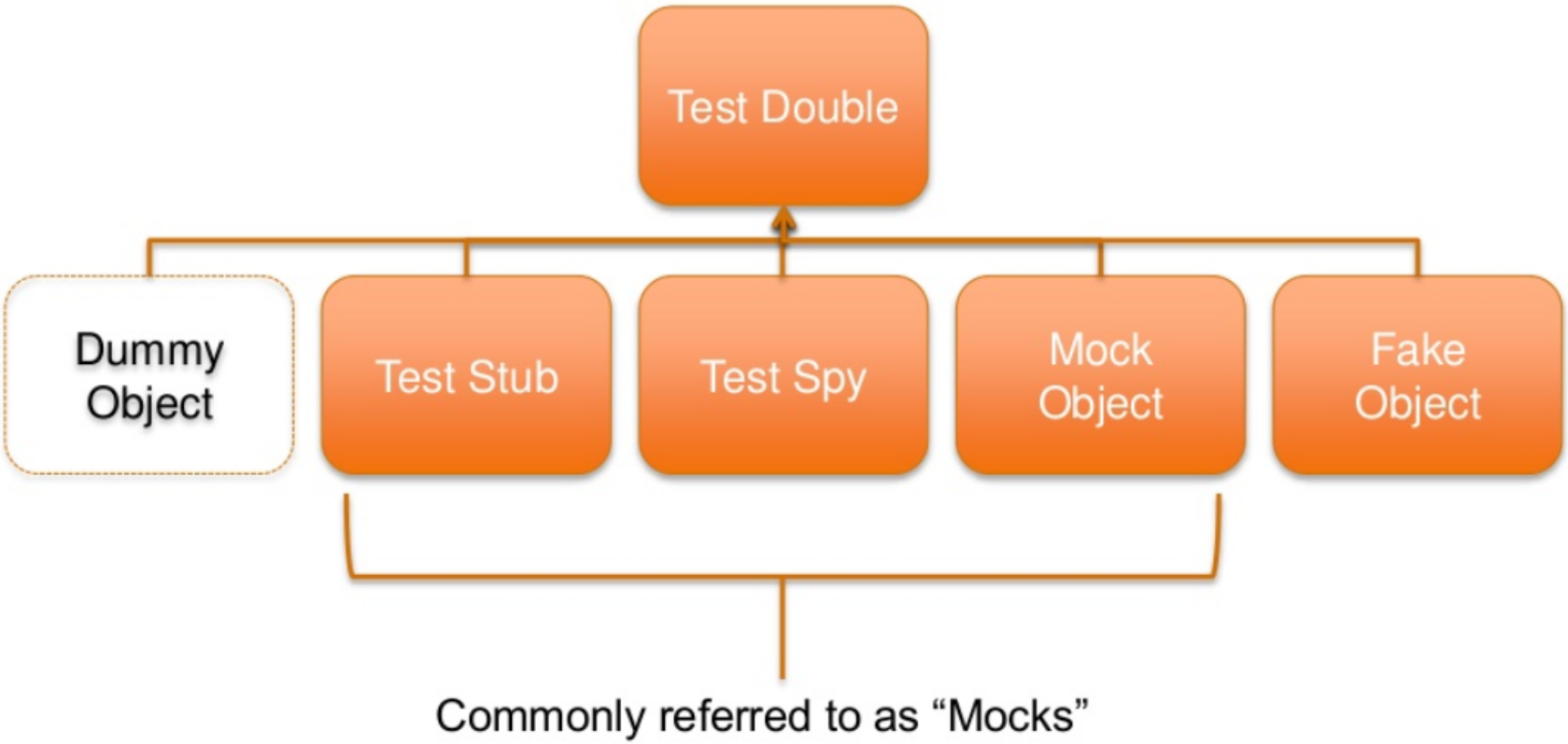
- Объекты-заглушки, симулирующие поведение реальных объектов и контролирующие обращения к своим методам
  - Как правило, такие объекты создаются с помощью библиотек
- Используются, когда реальные объекты использовать
  - Слишком долго
  - Слишком опасно
  - Слишком трудно
  - Для добавления детерминизма в тестовый сценарий
  - Пока реального объекта ещё нет
  - Для изоляции тестируемого объекта
- Для mock-объекта требуется, чтобы был интерфейс, который он мог бы реализовать, и какой-то механизм внедрения объекта
  - Как правило, как параметр конструктора или через поле

Пример



© https://www.slideshare.net/sudharajamanickam/mockito-24306321

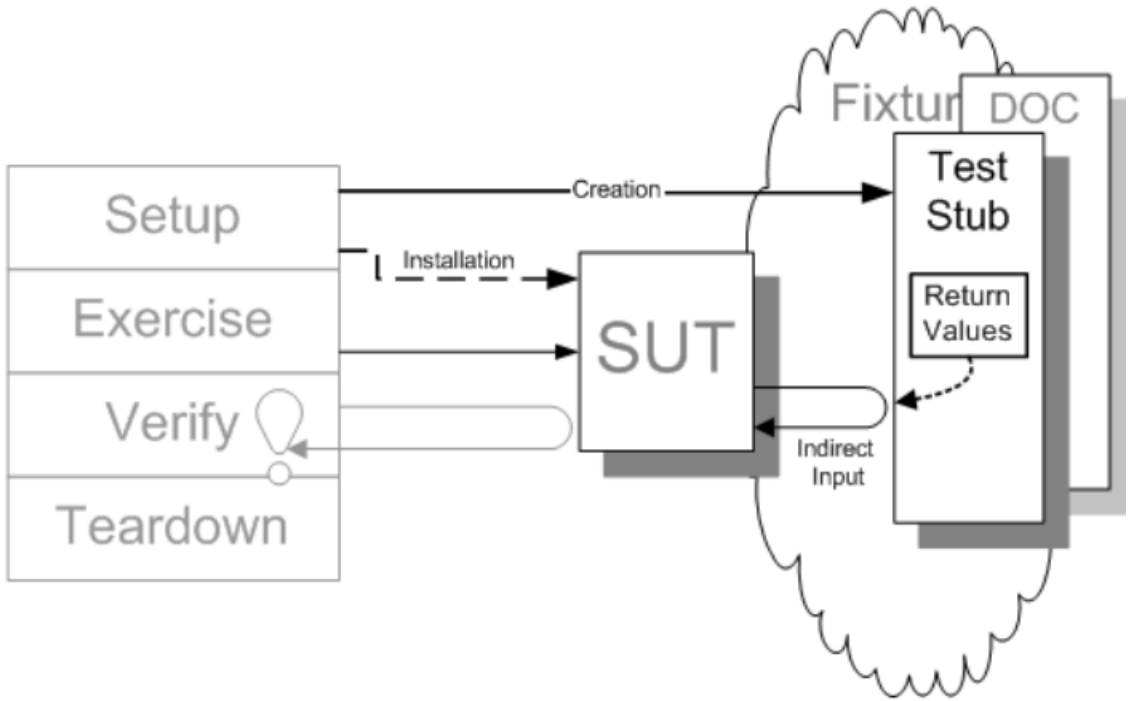
Объекты-заглушки



http://tinyurl.com/testdoubles

Stubs

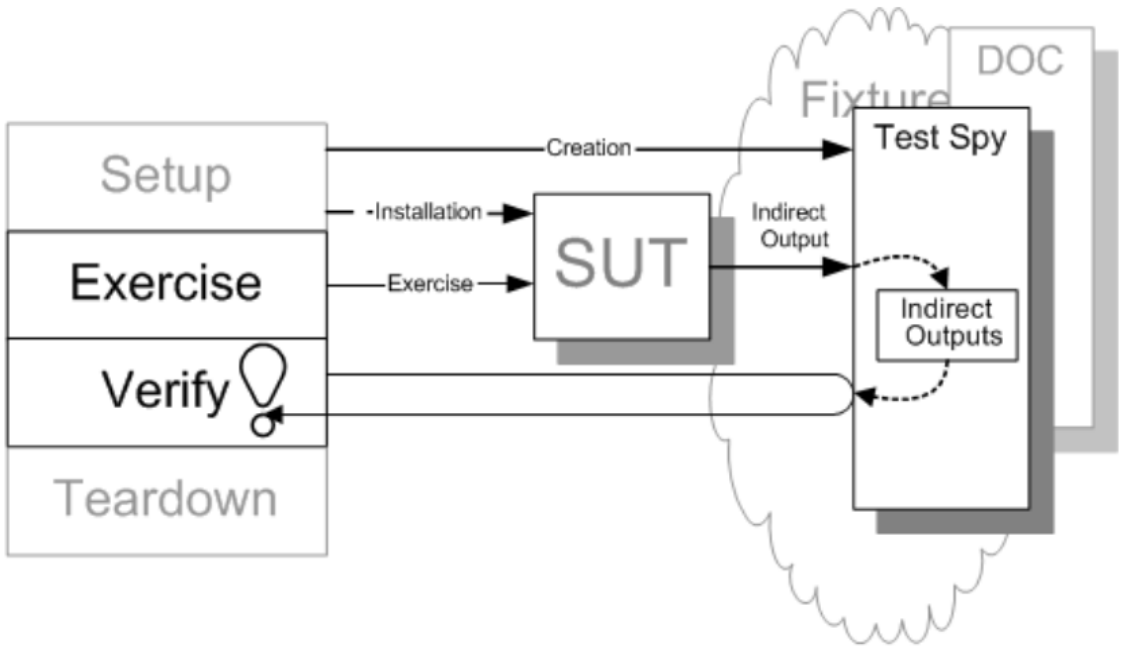
- Захардкоженные объекты, реализующие нужный интерфейс
- Наивная или вообще отсутствующая реализация
  - Возможно какие-то assert'ы или полезная логика для теста



© http://xunitpatterns.com

Spies

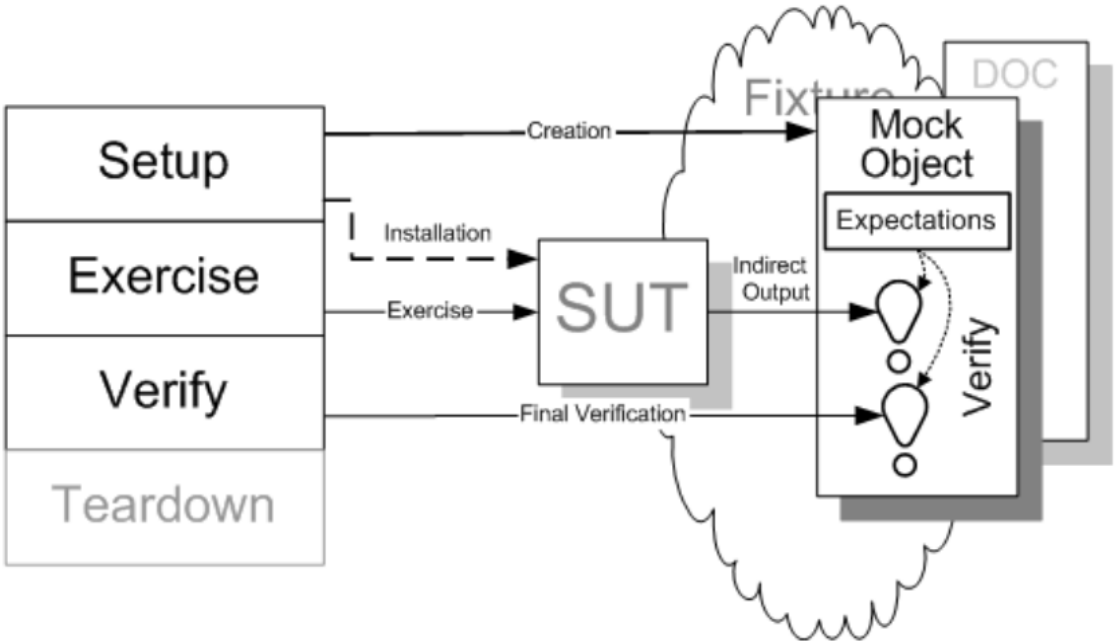
- Запуск теста, затем проверка условий и ограничений



© http://xunitpatterns.com

Mocks

- Конфигурирование объекта перед запуском теста



© http://xunitpatterns.com

Классический тест

```java
@Test
public void test() throws Exception {
    // Arrange
    var testee = new UnitToTest();
    var helper = new Helper();
    // Act
    testee.doSomething(helper);
    // Assert
    assertTrue(helper.somethingHappened());
}
```

Mockito

- http://site.mockito.org/
- https://github.com/mockito/mockito
- MIT license

Тест с mock-объектом

```
@Test
public void test() throws Exception {
    // Arrange, prepare behaviour
    Helper aMock = mock(Helper.class);
    when(aMock.isCalled()).thenReturn(true);
    // Act
    testee.doSomething(aMock);
    // Assert - verify interactions (optional)
    verify(aMock).isCalled();
}
```

Создание mock-объектов

- IMyClass myMock = mock(IMyClass.class)

Создание mock-объектов

- IMyClass myMock = mock(IMyClass.class)
- MyClass myMock = mock(MyClass.class)

Создание mock-объектов

- IMyClass myMock = mock(IMyClass.class)
- MyClass myMock = mock(MyClass.class)
- @mock **private** MyClass myMock;
  - @Before **public void** setup() {
    MockitoAnnotations.initMocks(**this**);
    }

when(...)

- when(mock.someMethod()).thenReturn(10);

when(...)

- `when(mock.someMethod()).thenReturn(10);`
- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException());
  ```
  ```

when(...)

- `when(mock.someMethod()).thenReturn(10);`
- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException());
  ```
- `when(mock.someMethod(anyString())).thenReturn(42);`
  ```

when(...)

- `when(mock.someMethod()).thenReturn(10);`
- `when(mock.someMethod("some arg"))`
  `.thenThrow(new RuntimeException());`
- `when(mock.someMethod(anyString())).thenReturn(42);`
- `when(mock.someMethod(anyInt())).thenReturn("element");`

when(...)

- `when(mock.someMethod()).thenReturn(10);`
- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException());
  ```
- `when(mock.someMethod(anyString())).thenReturn(42);`
- `when(mock.someMethod(anyInt())).thenReturn("element");`
- ```
  when(mock.someMethod(argThat(list -> list.size() < 3)))
      .thenReturn(null);
  ```

when(...)

- `when(mock.someMethod()).thenReturn(10);`
- `when(mock.someMethod("some arg"))`
  `.thenThrow(new RuntimeException());`
- `when(mock.someMethod(anyString())).thenReturn(42);`
- `when(mock.someMethod(anyInt())).thenReturn("element");`
- `when(mock.someMethod(argThat(list -> list.size() < 3)))`
  `.thenReturn(null);`
- `verify(target, times(1)).receiveComplexObject(`
  `argThat(obj -> obj.getSubObject().get(0).equals("expected")));`

Последовательные вызовы

- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
  ```

Последовательные вызовы

- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```

Последовательные вызовы

- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
- ```
  when(mock.someMethod(anyInt())).thenReturn("element");
  ```
  ```

Последовательные вызовы

- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
- ```
  when(mock.someMethod(anyInt())).thenReturn("element");
  ```
- ```
  when(mock.someMethod("some arg"))
      .thenReturn("one", "two", "three");
  ```

Последовательные вызовы

- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
- ```
  when(mock.someMethod("some arg"))
      .thenThrow(new RuntimeException())
      .thenReturn("foo");
  ```
- ```
  when(mock.someMethod(anyInt())).thenReturn("element");
  ```
- ```
  when(mock.someMethod("some arg"))
      .thenReturn("one", "two", "three");
  ```
- Но:

  ```
  when(mock.someMethod("some arg")).thenReturn("one")
  when(mock.someMethod("some arg")).thenReturn("two")
  ```

Подмена действия

```
when(mock.someMethod(anyString())).thenAnswer(new Answer() {
    Object answer(InvocationOnMock invocation) {
        Object[] args = invocation.getArguments();
        Object mock = invocation.getMock();
        return "called with arguments: " + args;
    }
});

//the following prints "called with arguments: foo"
System.out.println(mock.someMethod("foo"));
```

verify(...)

- verify(mock).someMethod();

verify(...)

- verify(mock).someMethod();
- verify(mock, timeout(100)).someMethod();

verify(...)

- `verify(mock).someMethod();`
- `verify(mock, timeout(100)).someMethod();`
- ```
  verify(mock).someMethod(
      anyInt(), anyString(), eq("third argument"));
  ```
  ```

verify(...)

- `verify(mock).someMethod();`
- `verify(mock, timeout(100)).someMethod();`
- `verify(mock).someMethod(anyInt(), anyString(), eq("third argument"));`
- `verify(mock).someMethod(argThat(someString -> someString.length() > 5));`

Повторные вызовы

- verify(mock, times(2)).someMethod();

Повторные вызовы

- verify(mock, times(2)).someMethod();
- verify(mock, never()).someMethod();
  - verifyZeroInteractions(mockTwo);

Повторные вызовы

- verify(mock, times(2)).someMethod();
- verify(mock, never()).someMethod();
  - verifyZeroInteractions(mockTwo);
- verify(mock, atLeastOnce()).someMethod();

Повторные вызовы

- verify(mock, times(2)).someMethod();
- verify(mock, never()).someMethod();
  - verifyZeroInteractions(mockTwo);
- verify(mock, atLeastOnce()).someMethod();
- verify(mock, atLeast(2)).someMethod();

Повторные вызовы

- verify(mock, times(2)).someMethod();
- verify(mock, never()).someMethod();
  - verifyZeroInteractions(mockTwo);
- verify(mock, atLeastOnce()).someMethod();
- verify(mock, atLeast(2)).someMethod();
- verify(mock, atMost(5)).someMethod();

Повторные вызовы

- verify(mock, times(2)).someMethod();
- verify(mock, never()).someMethod();
  - verifyZeroInteractions(mockTwo);
- verify(mock, atLeastOnce()).someMethod();
- verify(mock, atLeast(2)).someMethod();
- verify(mock, atMost(5)).someMethod();
- InOrder inOrder = inOrder(mock);
  inOrder.verify(mock).someMethod("was called first");
  inOrder.verify(mock).someMethod("was called second");

spy(...)

```
List list = new LinkedList();
List spy = spy(list);

when(spy.size()).thenReturn(100);

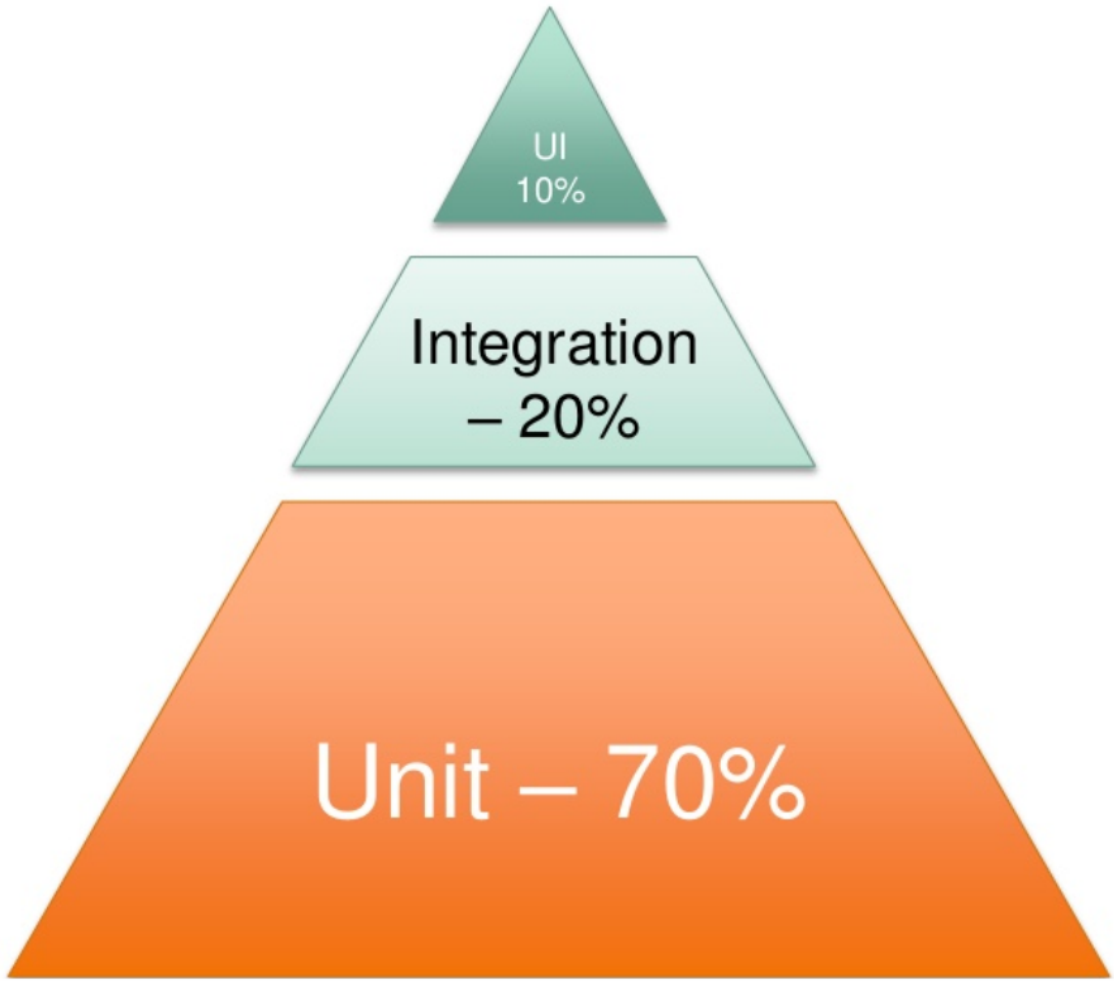
spy.add("one");
spy.add("two");

//prints "one" - the first element of a list
System.out.println(spy.get(0));

//size() method was stubbed - 100 is printed
System.out.println(spy.size());

verify(spy).add("one");
verify(spy).add("two");
```

Hamcrest

- assertThat(someString, is(not(equalTo(someOtherString))));

Hamcrest

- assertThat(someString, is(not(equalTo(someOtherString))));
- assertThat(list, everyItem(greaterThan(1)));

Hamcrest

- assertThat(someString, is(not(equalTo(someOtherString))));
- assertThat(list, everyItem(greaterThan(1)));
- assertThat(cat.getKittens(), hasItem(someKitten));

Hamcrest

- assertThat(someString, is(not(equalTo(someOtherString))));
- assertThat(list, everyItem(greaterThan(1)));
- assertThat(cat.getKittens(), hasItem(someKitten));
- assertThat("test", anyOf(is("testing"), containsString("est")));

Hamcrest

- `assertThat(someString, is(not(equalTo(someOtherString))));`
- `assertThat(list, everyItem(greaterThan(1)));`
- `assertThat(cat.getKittens(), hasItem(someKitten));`
- `assertThat("test", anyOf(is("testing"), containsString("est")));`
- ```
  assertThat(x,
      allOf(greaterThan(0), lessThanOrEqualTo(10)));
  ```
  ```

Примерное соотношение объёма тестов в проекте



Задание на остаток пары

- Реализовать стековый калькулятор, принимающий на вход строку с арифметическим выражением в постфиксной записи и возвращающий результат его вычисления
  - Должны поддерживаться операции $+$, $-$, $*$, $/$
  - Можно использовать библиотечный стек
- Протестировать калькулятор без стека с помощью библиотеки Mockito