TP 9: Test & Test Doubles

Programación Orientada a Objetos II
Comisión 2
2º Cuatrimestre de 2024

Nicolás Salvanescki

TDD y Test de Unidad

- Mantener de forma exhaustiva una suite de tests significa que no queden obsoletos ante modificaciones en el código. Esto puede lograrse actualizando los tests y/o no dependiendo directamente de las clases que no se estén testeando pero se necesiten utilizar en los tests (mocking).
- Como mencionaba anteriormente, los tests deben ser unitarios, es decir, que prueben los casos, tanto positivos como borde, de los mensajes de un SUT (System Under Test) o clase que estemos testeando. Dichos tests **NO DEBEN** utilizarse para probar también los DOC (Depended-on Component) o componentes de los que depende el SUT para ser testeado. Para solucionar esto se utilizan los Test Doubles, objetos falsos que responden de igual manera que los DOCs originales y permiten **aislar** el SUT de los DOC.
- Comunicar la intención del test es ser suficientemente descriptivo con el nombre o los comentarios para explicar de que caso de test se trata. ¿Se trata de un caso donde se pruebe una excepción?, ¿Es un caso borde?, etc.

Mockito

2. Para verificar que el objeto mock recibió una secuencia de mensajes en un orden preestablecido se usa InOrder. Por ejemplo:

```
List<String> mockList = mock(List.class);

mockList.add("primero");
mockList.add("segundo");
mockList.add("tercero");

InOrder inOrder = inOrder(mockList);
inOrder.verify(mockList).add("primero");
inOrder.verify(mockList).add("segundo");
    inOrder.verify(mockList).add("tercero");
```

3. Sin importar el orden o la obligatoriedad se usa simplemente verify

```
List<String> mockList = mock(List.class);

mockList.add("primero");
mockList.add("segundo");

verify(mockList).add("primero");
verify(mockList).add("segundo");
```

4. Si, se hace de la siguiente manera:

```
Clase mockObject = mock(Clase.class);

when(mockObject.method1().method2()).thenReturn(someValue);

assertEquals(someValue, mockObject.method1().method2());
```

5. Se hace utilizando el método verify(), que recibe como parámetro el mock y luego se llama el método. Además, puede recibir como segundo parámetro el modo de verificación. Entre estos destacan atLeastOne(), atLeast(int), atMost(int), times(int), timeout(long) y never().

Test Doubles

1. Los tests doubles son objetos falsos de una clase existente que se crean con el objetivo de testear el SUT sin depender de los DOC.

2. Los tipos de test doubles son:

   - Dummy: Son objetos que pasan por allí pero nunca se utilizan realmente. Por lo general, sólo son utilizadas para rellenar listas de parámetros.
   - Stub: Proporciona respuestas pre-programadas a las llamadas realizadas durante el testing.

```
MyService mockService = mock(MyService.class);
when(mockService.getData()).thenReturn("datos predefinidos");
```

   - Mock: Es un Stub pero, además, tiene la capacidad de verificarse.

```
MyService mockService = mock(MyService.class);

when(mockService.getData()).thenReturn("datos predefinidos");

verify(mockService).getData();
```

   - Spy: Es un Mock pero que llama realmente a los métodos del objeto real.

```
MyService realService = new MyService();
MyService spyService = spy(realService);

when(spyService.getData()).thenReturn("datos predefinidos");
```