P06- Dependencias externas 2: mocks

Revisión de las soluciones entregadas

Hemos revisado las entregas que habéis subido a Bitbucket de la práctica 6. Os indicamos algunos errores comunes que hemos observado y añadimos también explicaciones sobre las soluciones de cada ejercicio.

No obstante, si durante la realización o corrección de algún ejercicio os surgen otras dudas concretas, podéis enviarlas al foro sin son de carácter general o enviar una tutoría si se trata de alguna cuestión particular de vuestra solución.

Errores frecuentes que debéis evitar

- Hay que invocar a *replay()*, para cada doble, ANTES de invocar a la unidad a probar
- Hay que invocar a *verify()*, para cada doble, DESPUÉS de invocar a la unidad a probar
- No hay que crear ninguna clase "Testable" para sobreescribir métodos locales que son dependencias de nuestra sut, sino que hay que utilizar para ello un mock parcial
- Si no es imprescindible refactorizar, no hay que hacerlo. Ni en esta práctica ni en la anterior. En un examen no se admitirán refactorizaciones innecesarias.
- Para declarar una expectativa de un método void que lanza un excepción, hay que usar *expectLastCall()*
- Tenéis que evitar que el método anotado con @Test lance alguna excepción, y también "envolver" las sentencias que pueden lanzar excepciones dentro del driver con un try...catch. En su lugar DEBEIS usar una sentencia *assertDoesNotThrow()*
- Si no se especifica lo contrario, si pedimos una verificación basada en el comportamiento, HAY QUE VERIFICAR también el orden de llamadas ENTRE los dobles.

  Si UN doble se ha creado como un StrictMock, se verificará el orden de las llamadas "A ESE" doble. Pero si en el driver tenemos varios dobles, HAY QUE USAR UN IMocksControl, si queremos que se verifiquen también las invocaciones entre los dobles.

  En el caso de que uno de los dobles sea un "partial mock", la forma de incluir ese doble para que sea controlado por el objeto IMocksControl es la que mostramos en siguiente código:

```
//creamos los dobles con un IMocksControl
IMocksControl ctrl;
ctrl = EasyMock.createStrictControl();
Clase1 mock1 = EasyMock.partialMockBuilder(Clase1.class)
                       .addMockedMethods("metodo1","metodo2")
                       .createMock(ctrl);
Clase2 mock2 = ctrl.createMock(Clase2.class);
Clase3 mock3 = ctrl.createMock(Clase3.class);

//programamos las expectativas
ctrl.replay();
//invocamos a nuestro SUT
ctrl.verify();
```

- En todos los ejercicios de esta práctica, DEBÉIS usar un IMockControl.

Indicaciones sobre las soluciones de los ejercicios

EJERCICIO 1:

- Debéis crear un mock parcial y un mock para la clase Calendario. El doble de Calendario se inyecta con el expect() del método getCalendario()

EJERCICIO 2:

- Debéis crear un mock parcial y un mock de la clase ClienteWebService que es inyectado directamente actualizando el atributo público correspondiente

EJERCICIO 3:

- Debéis crear un mock parcial, un mock de la interfaz IService y otro mock de la clase Calendario.
- El mock de Calendario se inyecta directamente actualizando el atributo protected.
- Para inyectar el mock IService es necesario refactorizar, eliminando de la sut la sentencia "`Service servicio = new Servicio();`"

  El enunciado restringe la refactorización a añadir un método de factoría local o bien crear una clase factoría. Ejemplos de ambas opciones las tenéis en las transparencias de teoría y en las soluciones de la práctica de stubs.
- Recordad que el mock Calendario al ser strict, importa el orden de las invocaciones al método es_festivo() al declarar los correspondientes expect()

EJERCICIO 4:

- Debéis crear un mock parcial, un mock de la clase FactoriaBOs y otro mock de la interfaz IOperacionBO
- El mock de IOperacionBO se inyecta con un expect() del método getOperacionBO() del mock de FactoriaBOs
- Para inyectar el mock FactoriaBOs es necesario refactorizar, eliminando de la sut la sentencia: FactoriaBOs fd = new FactoriaBOs();

  El enunciado restringe las alternativas de refactorización a añadir un método de factoría local o bien crear una clase factoría. Es recomendable que elijáis la alternativa que no hayáis usado en el ejercicio anterior, para practicar ambas formas.
- En el test del caso de prueba C1 no hay que declarar expect() para los mocks de las clases FactoriaBOs y IOperacionBO
- En la implementación de los drives utilizando verificación basada en el estado, hay que utilizar: createNiceMock(), anyString(), anyObject(), andStubReturn(), expectLastCall().asStub(), expectLastCall().andStubThrow() y no invocar a verify()