Post Dobles

Un doble es un objeto que se hace pasar por el original ☺. Este objeto puede ser “Indutrial” o “Artesanal”.

Los dobles “Industriales” son las herramientas que nos proporcionan aquellos frameworks o librerías para crear, gestionar y administrar los dobles que necesitamos. Los más conocidos Xunit, EasyMock, RhinoMock y otros mucho. Cada lenguaje tiene uno o varios frameworks con los que nos podemos sentir cómodos y son sencillos de usar.

Los dobles “Artasenales” son aquellos objetos que nos creamos nosotros mismos (hechos a mano) para que se hagan pasar por otro objeto e imiten su funcionalidad sin implementarla.

Según la funcionalidad que imiten los dobles pueden ser:

Stub: Son aquellos objetos que no comprueban la funcionalidad del objeto que estamos probando únicamente devuelve el resutlado. Otra definición que leí en un foro es que se utilizan para los test que comprueban los cambios de estado. Este tipo de objeto no se usa para validar los Assert.

Mocks: Son aquellos objetos que comprueban la funcionalidad del objeto que estamos probando y nos devuelve el resultado según su lógica. También se utilizan para probar el funcionamiento de un objeto. Se usa para validar los Assert.

Fakes: Son aquellos objetos que nos creamos nosotros (pueden ser Mocks o stubs) para que se hagan pasar por un objeto que necesitamos.

Como a programar se aprende programando vamos a un ejemplo.

Cargar un fichero en una tabla en bbdd con los campos

ID no null entero

Titulo no null varchar(50)

Director no null varchar(50)

Cantidad no null entero

Precio no null float(2,2)

El fichero tiene los campos separados por |

Enviar un mail con las películas de un director pasado como parámetro.

Ciclo 1: Comprobar que existe el fichero de entrada

Cuando creamos el test nos damos cuenta que necesitamos una clase fichero con el propósito de gestionar nuestro fichero de entrada, en este caso que nos diga si existe. Como el fichero de entrada es un objeto externo necesitaremos crear un mock o un stub. Podemos utilizar una librería o framework o podemos crearnos nuestro propio doble. Como vamos a comprobar si existe o no usamos un stub que nos devolverá cuando lo llamemos true.

Si usamos un doble artifical en este caso con la librería RhinoMock.

        [TestMethod]

        public void Existe()

        {

            Rhino.Mocks.MockRepository mock = new Rhino.Mocks.MockRepository();

            FicheroEntrada ficheroEntrada = mock.Stub<FicheroEntrada>();

            Assert.IsTrue(ficheroEntrada.Existe("ficheroEntrada"));

        }

Clase FicheroEntrada

 public bool Existe(string nombreFichero)

        {

            return File.Exists(nombreFichero);

        }

Como el ficheo de entrada no existe el test se queda en rojo, como el objeto fichero de entrada es externo y lo hemos sustituido por un stub, vamos a hacer que el test se ponga verde añadiendo al stub el resultado que queremos cuando se llame al método Existe.

 [TestMethod]

        public void Existe()

        {

            Rhino.Mocks.MockRepository mock = new Rhino.Mocks.MockRepository();

            IFicheroEntrada ficheroEntrada = mock.Stub<IFicheroEntrada>();

            Rhino.Mocks.Expect.Call(ficheroEntrada.Existe("")).IgnoreArguments().Return(true);

            mock.Replay(ficheroEntrada);

            Assert.IsTrue(ficheroEntrada.Existe("ficheroEntrada"));

            mock.Verify(ficheroEntrada);

        }

Modificaciones al insertar el resultado del stub.

RhinoMockTrabaja con Interface por lo que hemos tenido que extraer una interface de la clase FicheroEntrada para trabajar sobre ella.

Definimos verificamos las expectativas que esperamos cuando se llame a un método del stub

Con esto le estamos diciendo que cuando nuestro test llame al método Existe() de la clase FicheroEntrada este debe devolver True independientemente de lo que le pasemos.

Despues de refactorizar podemos añadir el nombre del fichero a la Interface y nos queda el test como

        [TestMethod]

        public void Existe()

        {

            Rhino.Mocks.MockRepository mock = new Rhino.Mocks.MockRepository();

            IFicheroEntrada ficheroEntrada = mock.Stub<IFicheroEntrada>();

            Rhino.Mocks.Expect.Call(ficheroEntrada.Existe("")).IgnoreArguments().Return(true);

            mock.Replay(ficheroEntrada);

            Assert.IsTrue(ficheroEntrada.Existe(ficheroEntrada.nombreFicheroEntrada));

            mock.Verify(ficheroEntrada);

        }

Fijaros que al ponerle IgnoreParameter le podemos pasar cualquier cosa le da lo mismo y devolverá siempre true

Cicclo 2: Comprobar que el fichero de entrada tiene registros.

[TestMethod]

        public void Leer\_Registros\_Del\_Fichero()

        {

            Rhino.Mocks.MockRepository mock = new Rhino.Mocks.MockRepository();

            FicheroEntrada ficheroEntrada = mock.DynamicMock<FicheroEntrada>();

            Assert.IsFalse(ficheroEntrada.esVacio);

        }