ACTIVIDAD GUIADA 2

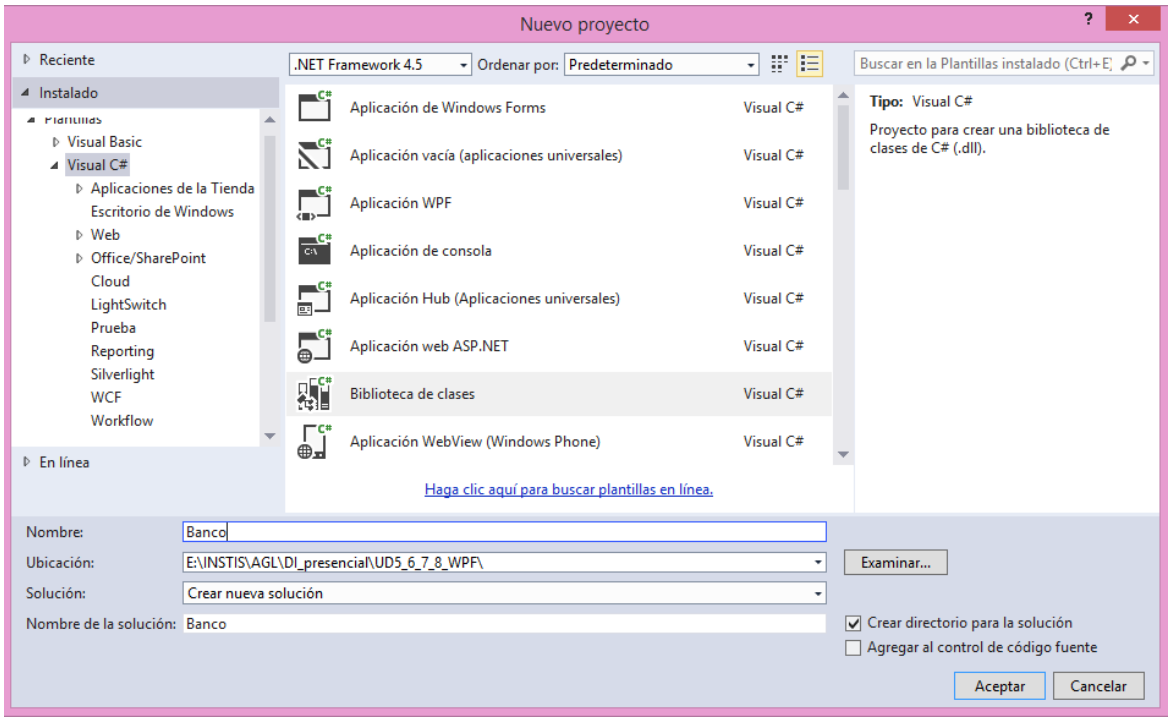
Pruebas unitarias en Visual Studio

Contenido

Preparando la clase a probar 1
Creación del proyecto de pruebas 3
Creación de los métodos de prueba 4
Ejecución de las pruebas 5
EJERCICIOS 5

Preparando la clase a probar

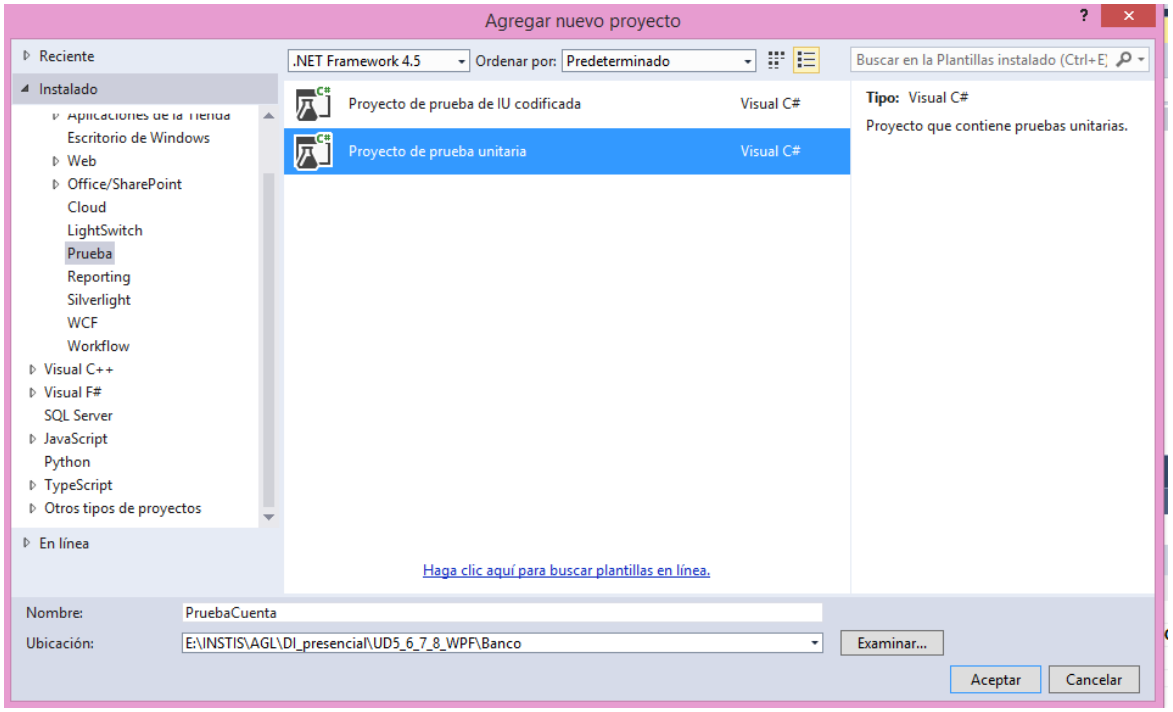
Creamos un Nuevo Proyecto de tipo "***Biblioteca de clases***", llamado Banco, donde crearemos una clase CuentaBanco.cs con el código siguiente que representará una cuenta bancaria:



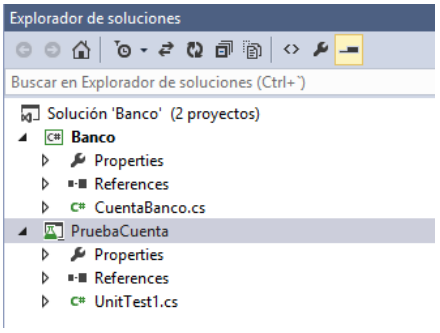
```
namespace Banco
{
    public class CuentaBanco
    {
        private string nombre;
        private double balance;
        private bool congelada = false;

        public CuentaBanco(string cliente, double balance)
        {
            this.nombre = cliente;
            this.balance = balance;
        }

        public string NombreCliente{get { return nombre; } }

        public double Balance {get { return balance; } }

        public void Debito(double cantidad)
        {
            if (congelada)
                throw new Exception("La cuenta está congelada");
            if (cantidad > balance)
                throw new ArgumentOutOfRangeException("cantidad");
            if (cantidad < 0)
                throw new ArgumentOutOfRangeException("cantidad");

            balance += cantidad; // codigo incorrecto de forma intencionada
        }

        public void Credito(double cantidad)
        {
            if (congelada)
                throw new Exception("Cuenta congelada");
            if (cantidad < 0)
                throw new ArgumentOutOfRangeException("cantidad");

            balance += cantidad;
        }

        private void CongelarCuenta()
        {
            congelada = true;
        }

        private void DescongelarCuenta()
        {
            congelada = false;
        }

        public static void Main()
        {
            CuentaBanco cb = new CuentaBanco("Sr. Gonzalez", 11.99);
            cb.Credito(5.77);
            cb.Debito(11.22);
            Console.WriteLine("El balance actual es is ${0}", cb.Balance);
        }
    }
}
```

Creación del proyecto de pruebas

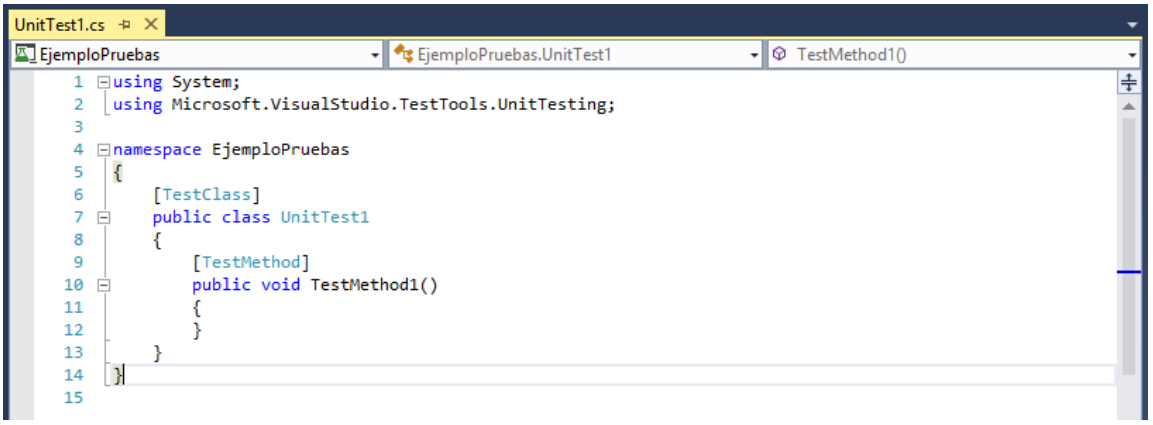
Veamos el funcionamiento de una prueba unitaria dentro del entorno Visual Studio. Crearemos un Proyecto específico de tipo ***Prueba*** llamado "*PruebaCuenta*" como se ve en la siguiente figura. Para crearlo dentro de la misma Solución, habrá que escoger la opción ***Agregar Nuevo Proyecto***.



Con lo que conseguiremos la siguiente estructura:

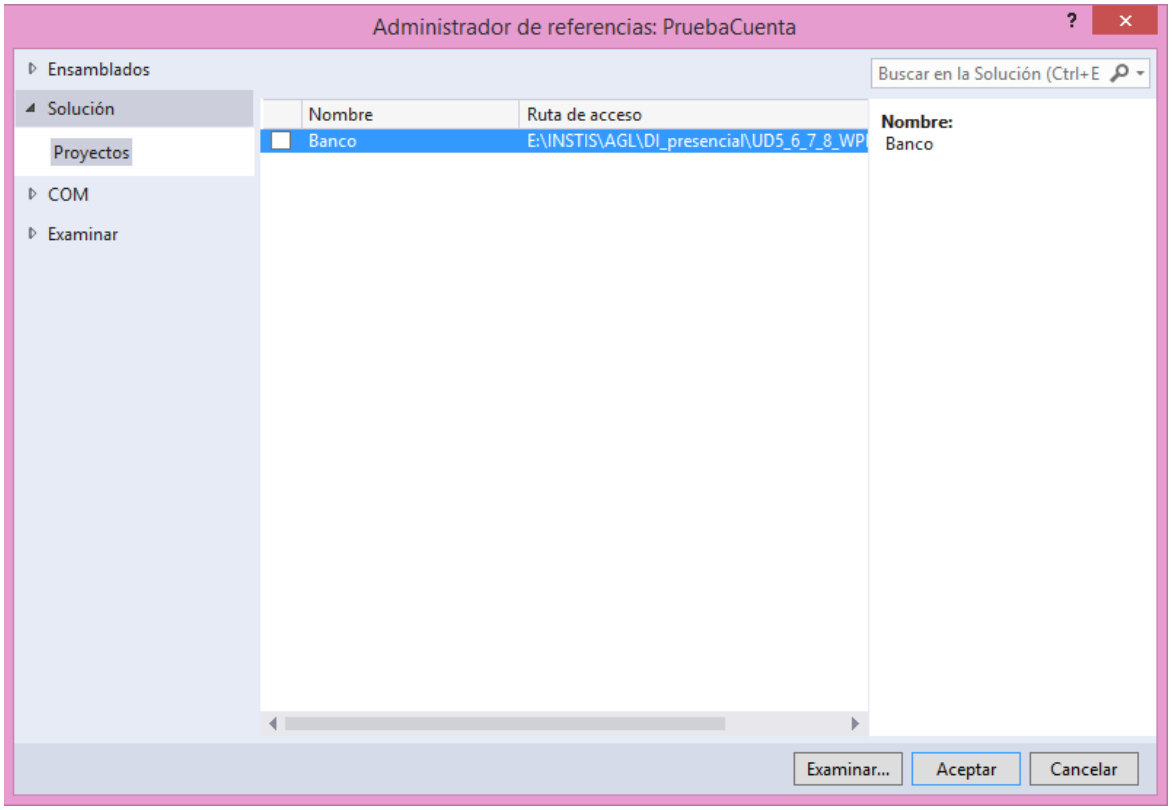


Visual Studio dispone de una plantilla predeterminada que nos crea una solución con una clase de prueba ***UnitTest1.cs*** (nombre por defecto) con el siguiente código:



Como se puede ver se crea una clase con el atributo **[TestClass]** que indica que posee métodos de prueba, y cada uno de los métodos que podemos evaluar se encuentran identificados por el atributo **[TestMethod].**

A continuación será necesario agregar una **referencia** desde el Proyecto de pruebas hasta el Proyecto a probar. Con el botón derecho sobre el Proyecto de pruebas, escogemos la opción Agregar referencia y seleccionamos "Banco":



También es útil añadir la sentencia `using CuentaBanco` al Proyecto de pruebas, para no tener que dar la ruta completa cuando queramos usar sus objetos.

Creación de los métodos de prueba

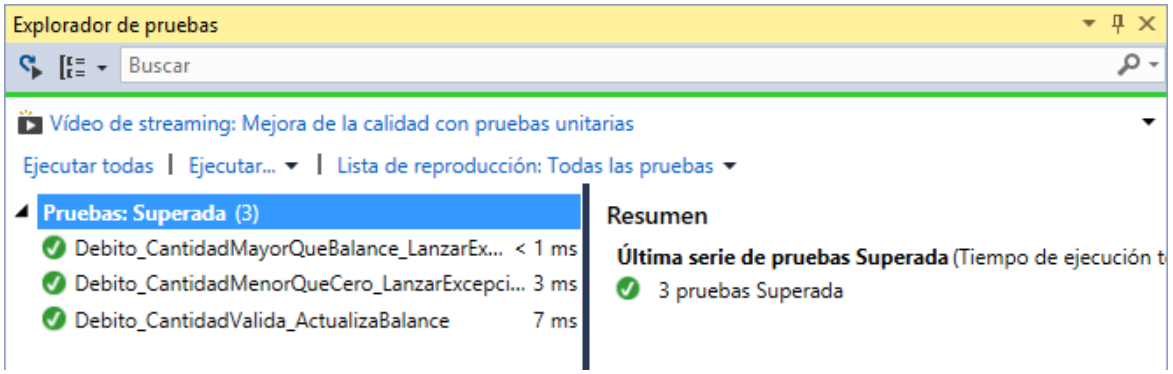
Como vimos con JUnit, un aspecto esencial en las pruebas es la inclusión de asertos para indicar a la prueba el resultado esperado y si no se cumple mostrar un mensaje por pantalla.

En el ejemplo nos centraremos en probar el método **Débito** que se llama cuando se quiere retirar dinero de la cuenta. Al analizar el método en pruebas, se determina que hay al menos tres comportamientos que deben comprobarse:

- El método produce *ArgumentOutOfRangeException* si la cantidad de crédito es mayor que el saldo.
- También produce *ArgumentOutOfRangeException* si la cantidad de crédito es menor que cero.
- Si se cumplen las comprobaciones 1.) y 2.), el método resta la cantidad del saldo de cuenta.

Ejecución de las pruebas

Para ejecutar la prueba lo más práctico es abrir el Explorador de pruebas (PRUEBA/Ventanas/Explorador de pruebas), o bien ir directamente a la opción ejecutar del menu PRUEBA. Una vez ejecutada, podemos ver si ha sido correcta, y su salida:



EJERCICIOS

1) Comprobar que una cantidad válida (menor que el saldo de la cuenta y mayor que cero) retira la cantidad correcta de la cuenta:
    a. Hacer un método de prueba que cree una cuenta de banco con un saldo inicial y se le retire una cantidad de dinero válida con el método Debito.
    b. El método para comparar dos cantidades es `Assert.AreEqual`.
    c. Corregir el código si es necesario y volver a ejecutar las pruebas.
2) Comprobar que se lanza la excepción correspondiente si la cantidad a retirar es menor que 0.
    a. Hacer un método de prueba que cree una cuenta de banco con un saldo inicial y se le retire una cantidad de dinero negativa con el método Debito.
    b. Para comprobar que se ha lanzado la excepción se puede hacer uso del atributo "ExpectedException", a través de la siguiente sintaxis:

```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))
```

    Este atributo hace que la prueba dé error a menos que se produzca una excepción del tipo indicado.
    c. Probar el método retirando cantidades positivas y negativas.
3) Comprobar que se lanza la excepción correspondiente si la cantidad a retirar es mayor que el saldo.
    a. Hacer un método de prueba que cree una cuenta de banco con un saldo inicial y se le retire una cantidad de dinero menor que el balance con el método Debito.

    La forma de proceder es similar al apartado anterior.

4) De la ejecución de las pruebas anteriores se deduce que el hecho de que ambas condiciones (mayor que el balance y menor que cero) devuelven una excepción del mismo tipo, y no es posible distinguir cuál de las dos ha infringido la condición.

   Generar más información cuando el método inicie una excepción ayudaría a todos los componentes relacionados, pero el atributo *ExpectedException* no puede proporcionar esta información.

   Consultado la MSDN Library se puede averiguar que que existe un constructor de la excepción más completo *ArgumentOutOfRangeException (String, Object, String*) que incluye el nombre del argumento, su valor y un mensaje definido por el usuario. Se puede refactorizar el método en pruebas para utilizar este constructor.

   a. Añadir dos constantes

   b. Modificar el método Debit

   c. Refactorizar el método de prueba

```
public const string CantidadExcedeBalance = "La cantidad es mayor que el balance";
public const string CantidadNegativa = "La cantidad a retirar es menor que cero";
```

```
if (cantidad > balance)
                throw new ArgumentOutOfRangeException(CantidadExcedeBalance);

if (cantidad < 0)
                throw new ArgumentOutOfRangeException(CantidadNegativa);
```