UNIVERSIDAD PRIVADA-DE-TACNA

INGENIERIA DE SISTEMAS

TITULO:

Trabajo Final de Unidad

CURSO:

BASE DE DATOS II

DOCENTE(ING):

Patrick Cuadros Quiroga

Integrantes:

Orlando Antonio Acosta Ortiz	(2015052775)
Orestes Ramirez Ticona	(2015053236)
Nilson Laura Atencio	(2015053846)
Roberto Zegarra Reyes	(2010036175)
Richard Cruz Escalante	(2013047247)
Wilfredo Vilca Chambilla	(2006028540)

Índice

1. PROBLEMATICA

El siguiente trabajo se desarrolla con el Sistema de Cajero Automatico, el cual busca que el cualquier cajero del Banco Pichincha pueda realizar acciones de forma mas automatizada y con toda seguridad:

- La Gestion de Cajero, Retiros, Depositos, Consultas
- Gestión de las usuarios que cuenten con una cuenta en el banco PICHINCHA.
- La venta de distintos tipos de planes.
- La realización de reportes sobre las transacciones realizadas del día, del mes, y sobre los clientes correspondientes a cada local.
- El sistema permitirá realizar la autentificación de los usuarios.

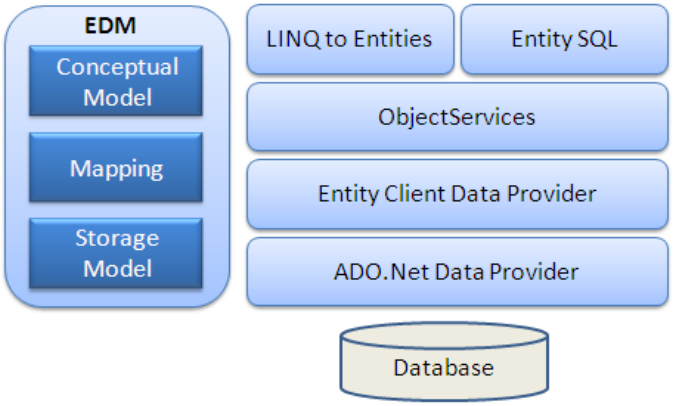
La motivación principal de este proyecto es que los cajeros del Banco Pichincha tengan una automatizacion mas eficaz, lo que le permitirá realizar sus tareas sin inconvenientes.

2. MARCO TEORICO

2.1. Entity Framework

Entity Framework es un marco de ORM de código abierto para aplicaciones .NET admitidas por Microsoft. Permite a los desarrolladores trabajar con datos utilizando objetos de clases específicas del dominio sin centrarse en las tablas y columnas de la base de datos subyacente donde se almacenan estos datos. Con el Entity Framework, los desarrolladores pueden trabajar en un nivel más alto de abstracción cuando tratan con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código en comparación con las aplicaciones tradicionales.[2]

Su arquitectura:



2.2. API REST

Ha pasado más de una década desde que Roy Fielding, un científico informático estadounidense y uno de los autores principales de la especificación HTTP, introdujo Representational State Transfer (REST) como un estilo de arquitectura. A lo largo de los años, REST ha ganado impulso gracias a su popularidad para la creación de servicios web.
Al no tener estado, y al crear identificadores únicos para permitir el almacenamiento en caché, la creación de capas y la capacidad de lectura, las API REST hacen uso de los verbos HTTP existentes (GET, POST, PUT y DELETE) para crear, actualizar y eliminar nuevos recursos. El término REST se usa con frecuencia para describir cualquier URL que devuelva JSON en lugar de HTML.

2.3. Consultas LINQ

Permite consultar cualquier objeto enumerable en ADO.NET mediante el uso del modelo de programación de Language-Integrated Query (LINQ) [1] para recuperar datos de la base de datos subyacente. El proveedor de la base de datos traducirá estas consultas LINQ al lenguaje de consulta específico de la base de datos (por ejemplo, SQL para una base de datos relacional).
Hay 3 tecnologias ADO.NET LINQ distintas:

- LINQ to DataSet: proporciona una capacidad de consulta mas rica y optimizada sobre DataSet.

- LINQ to SQL: permite consultar directamente los esquemas de las base de datos de SQL Server.
- LINQ to Entities: permite consultar Entity Data Model.

2.4. Pruebas Unitarias

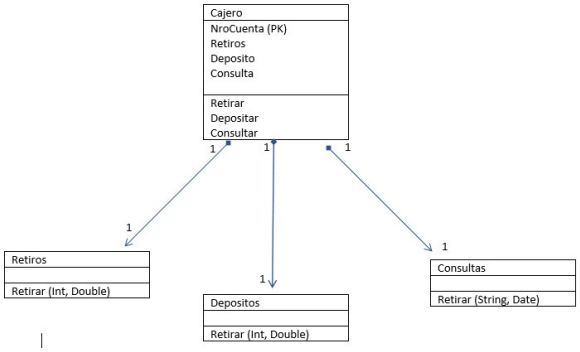
Las pruebas unitarias son la mejor forma de probaar el codigo de una aplicación a lo largo de su desarrollo. Estas pruebas permiten asegurar que losmétodos devuelven resultados correctos, teniendo en cuenta los argumentos que se le pasan y, además, se pueden utilizar para hacer Test Driven Development. Se trata de una técnica en la que se escriben antes que las clases y los métodos.[4]
No obstante, tras realizar la integración con otros módulos deberá revisarse de nuevo la interfaz.

3. DESARROLLO

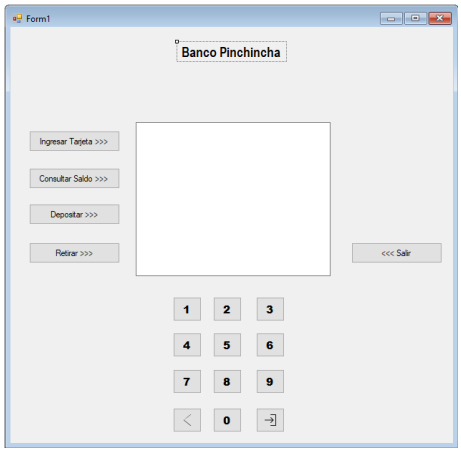
3.1. Desarrollo del Sistema

1. Pasos de desarrollo de la aplicacion

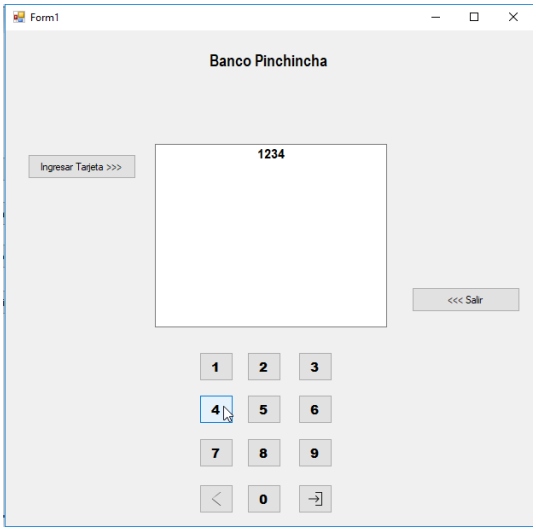
   a) Paso 1: Diagrama de Clases de solución Cajero



   b) Paso2: Crear un Windows form para desorrallar nuestra aplicación de escritorio:



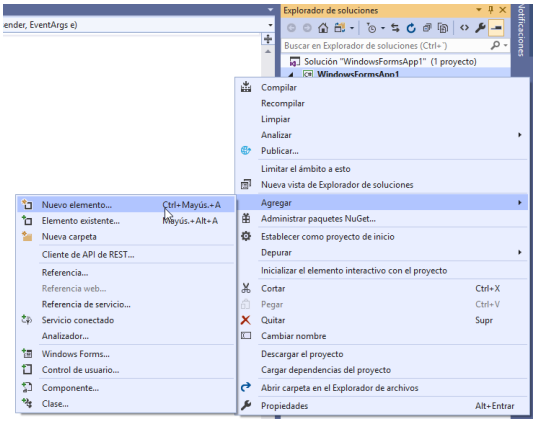
   c) Paso 3: Codificar los botones según lo planeado para nuestra aplicación, por ejemplo para esta aplicación hemos decidido virtualizar un cajero automático.



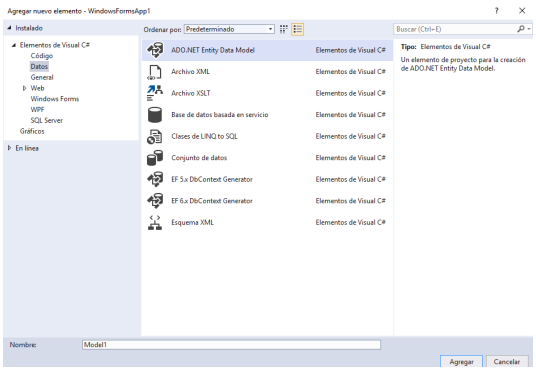
d) Paso 4: Muestra de algo de código de los botones:

```
1 referencia
private void Button1_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "1";
}

1 referencia
private void BtnNro2_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "2";
}

1 referencia
private void BtnNro3_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "3";
}

1 referencia
private void BtnNro4_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "4";
}
```

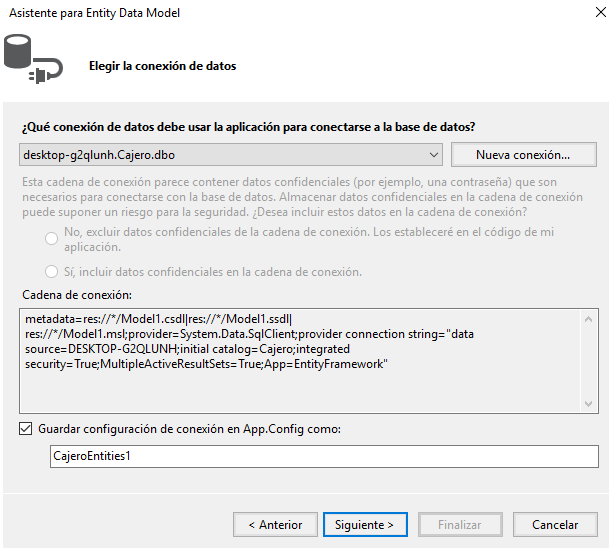
e) Paso 5: aplicar un ORM a nuestra aplicación: aplicaremos el ORM Entity framework(define que es y para que sirve entity framework)



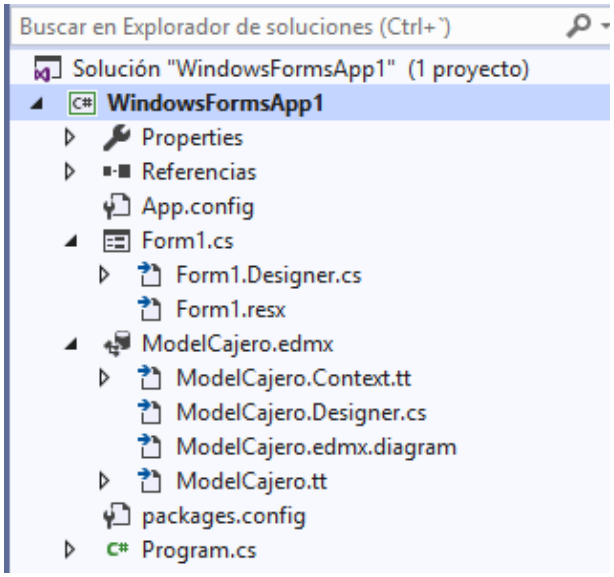
f) Paso 6: Agregaremos el modelo ADO.NET entity data model



g) Paso 7: ELuego configuraremos con la base de datos que se encuentra en Nuestro servidor SQL



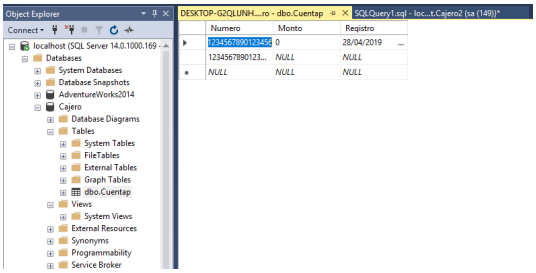
h) Paso 8: una vez configurado podremos comprobar que ya podemos interactuar con la base de datos a través Entity Framework.



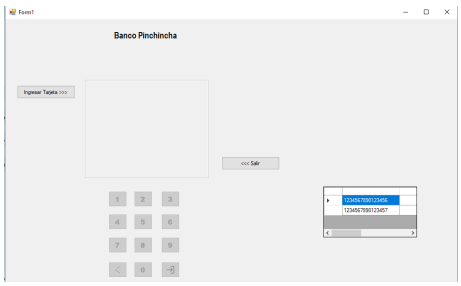
- Despues escribrir la siguiente consulta.

```
1 referencia
public void llamardatos()
{
    using (CajeroEntities db = new CajeroEntities())
    {

        var list = db.Cuentap;
        foreach (var oCuenta in list)

        {
            DgvDatos.ColumnCount = 4;
            DgvDatos.Rows.Add(oCuenta.Numero);

        }
    }

    }
```

i) Paso 9: Demostraremos el de entity framework con un Login. Para lograr esto liste los datos de Tarjeta que se encuentran en la base de datos en un DataGridView (Por motivos de Seguridad este DataGridView no será visible para el Usuario) con el siguiente código:



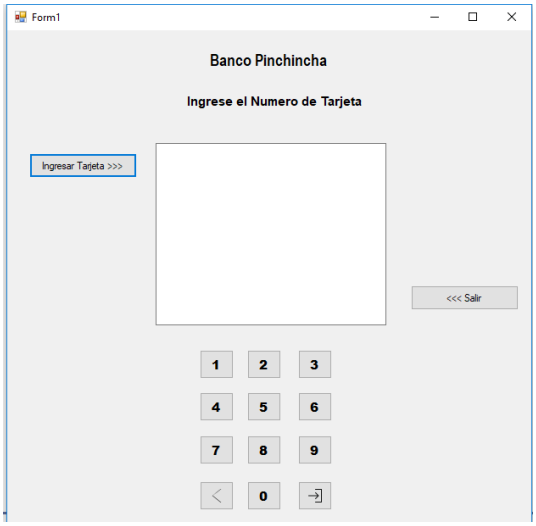
j) Paso 10: Demostraremos el de entity framework con un Login. Para lograr esto liste los datos de Tarjeta que se encuentran en la base de datos en un DataGridView (Por motivos de Seguridad este DataGridView no será visible para el Usuario) con el siguiente código:



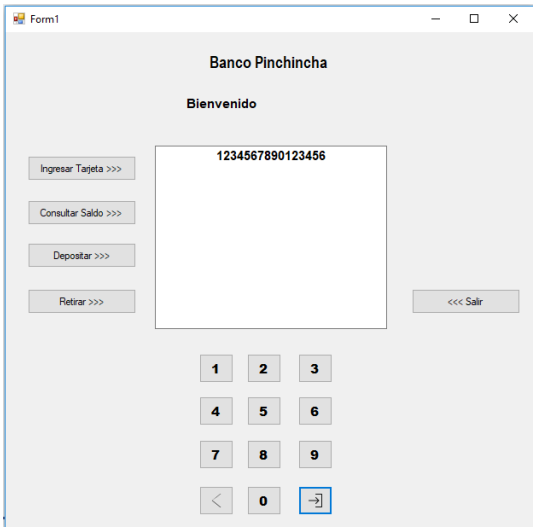
k) Paso 11: Una vez obtenidos los datos podremos confirmar si el usuario existe o no en la base de datos: Codigo:

```
1 referencia
public void Logear()
{
    using (CajeroEntities db = new CajeroEntities())
    {


        NroCuenta = TxtPantalla.Text;

        for (int i = 0; i < DgvDatos.RowCount; i++)
        {
            string dato;
            dato = Convert.ToString(DgvDatos.Rows[i].Cells[0].Value);

            if (NroCuenta == dato)
            {
                LblGuia.Text = "Bienvenido";
                MostrarBotones();
                break;

            }
            else
            {
                LblGuia.Text = "Usted No esta Registrado";
                OcultarBotones();
            }
        }

    }
}
```

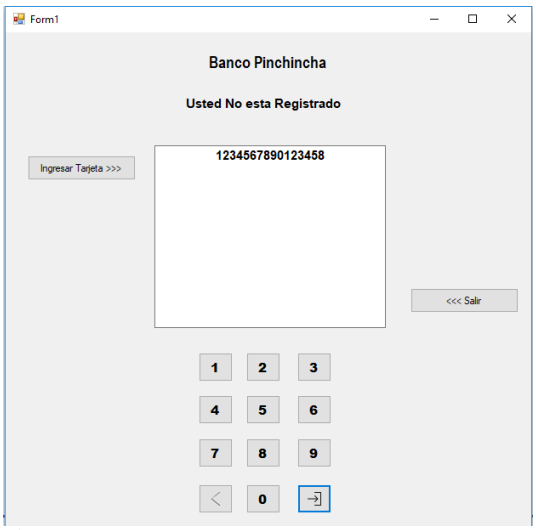
l) Paso 12: Se muestra el formulario principal del cajero



m) Paso 13: Si existe el formulario

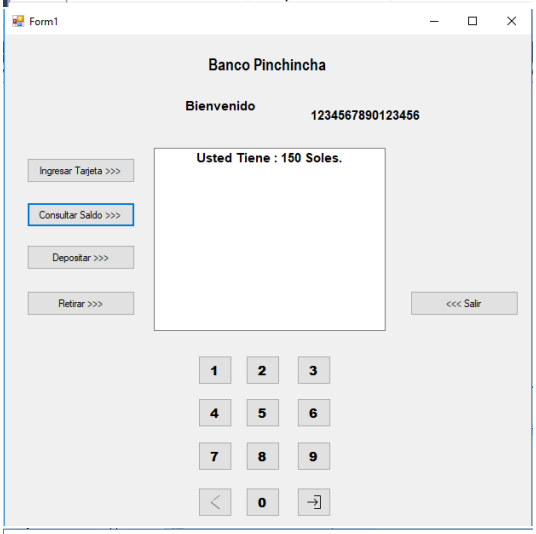


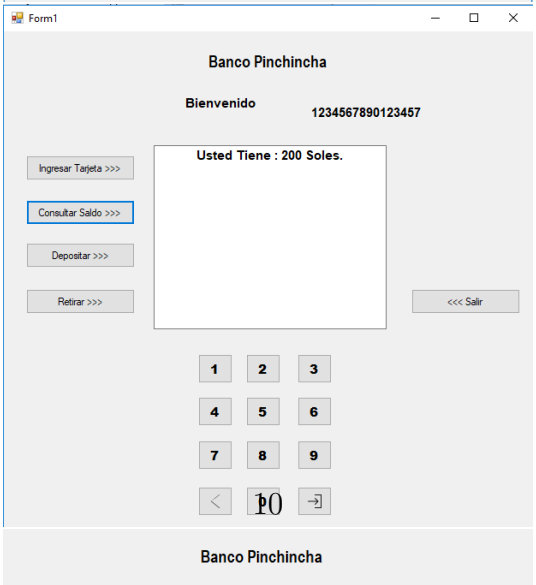
n) Paso 14: Si en caso que no existe




```
1 referencia
public void Consultar()
{
    using (CajeroEntities db = new CajeroEntities())
    {

        for (int i = 0; i < DgvDatos.RowCount; i++)

        {
            string dato;
            dato = Convert.ToString(DgvDatos.Rows[i].Cells[0].Value);
            if (NroCuentaAnc.Text == dato)
            {
                TxtPantalla.Text ="Usted Tiene : " + Convert.ToString(DgvDatos.Rows[i].Cells[1].Value)+" Soles.";

            }
        }
    }

}
```

	Numero	Monto	Registro
	1234567890123456	150	28/04/2019 ...
	1234567890123457	200	NULL
▶*	NULL	NULL	NULL





ñ) Paso 14: Y así hemos demostradro la funcionalidad con un ORM. Ha quedado demostrado la simplicidad del código que hemos usado.

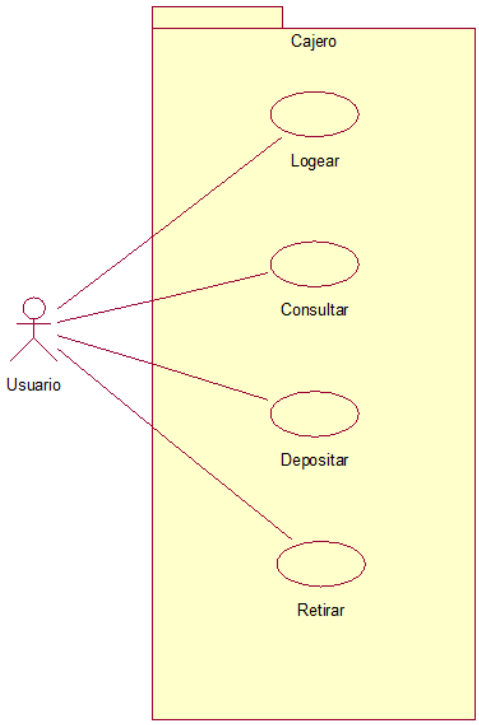
DESKTOP-G2QLUNH....ro - dbo.Cuentap | SQLQuery1.sql - loc...t.Cajero2 (sa (149))*

	Numero	Monto	Registro
	1234567890123456	150	28/04/2019 ...
	1234567890123457	0	28/04/2019 ...
▸*	NULL	NULL	NULL

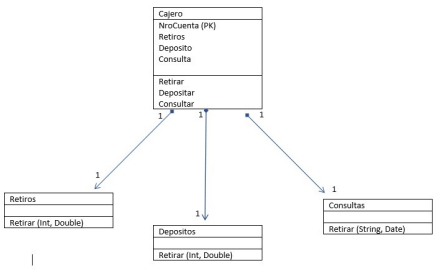
3.2. Analisis

1. Escribiendo consultas con el operador PIVOT

   a) Analisis de desarrollo del Sistema - Caso de uso del Sistema



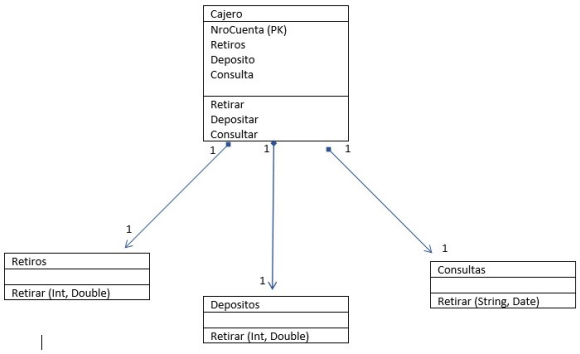
- Diagrama de Entidad Relacion



3.3. Diseño

1. Diagrama de Clases, Modelo Entidad Relación

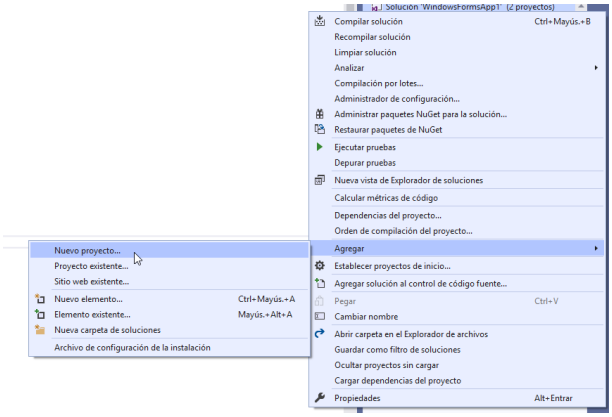
   a) El diagrama de entidad Relacion fue desarrollado con el fin de que la base de datos tenga una estrutura relacional optima



3.4. Pruebas

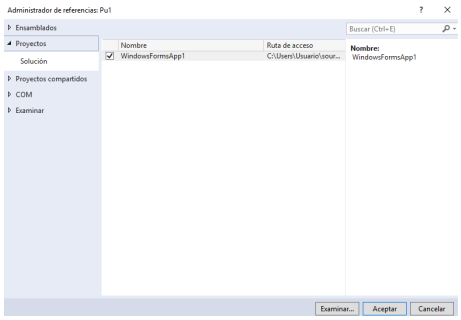
1. Pruebas Unitarias

   a) Paso 1: Pruebas unitarias: haremos pruebas unitarias para los métodos deposito, retiro.
   - Agregaremos un test de pruebas unitarias

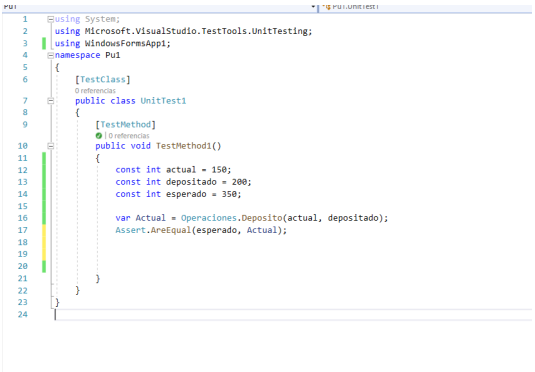


   b) Paso 2: Seguidamente el nuevo y tipo de proyecto.

c) Paso 3: Luego referenciaremos el proyecto principal

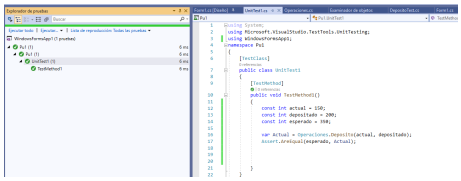


d) Paso 4: Luego probaremos los métodos :

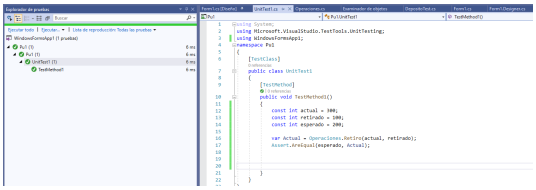


```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using WindowsFormsApp1;
namespace Pu1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
            const int actual = 150;
            const int depositado = 200;
            const int esperado = 350;

            var Actual = Operaciones.Deposito(actual, depositado);
            Assert.AreEqual(esperado, Actual);


        }
    }
}
```

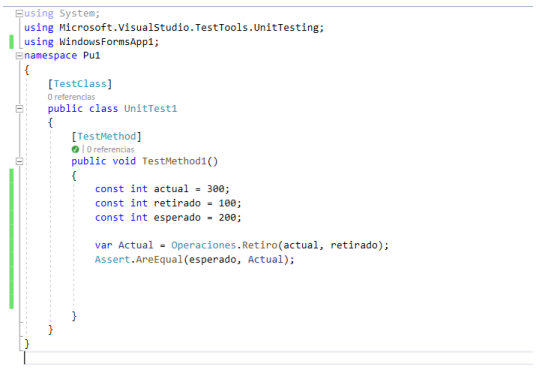
e) Paso 5: Esperamos mientras carga la pruba que se va desarrollar



f) Paso 6: Retiros



g) Paso 7: Finalmente aplicamos el codigo que se ejecurtar



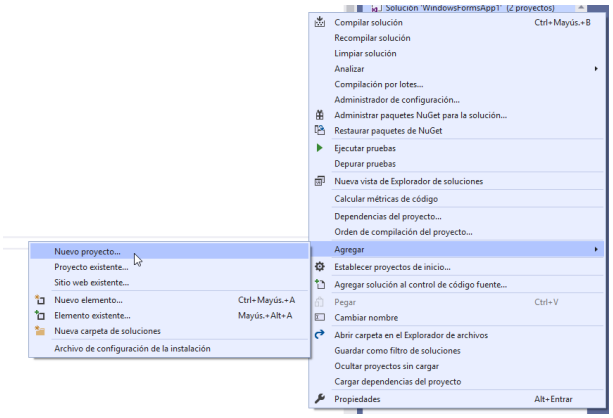
```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using WindowsFormsApp1;
namespace Pu1
{
    [TestClass]
    0 referencias
    public class UnitTest1
    {
        [TestMethod]
        0 referencias
        public void TestMethod1()
        {
            const int actual = 300;
            const int retirado = 100;
            const int esperado = 200;

            var Actual = Operaciones.Retiro(actual, retirado);
            Assert.AreEqual(esperado, Actual);



        }
    }
}
```

4. ANALISIS E INTERPRETACION DE RESULTADOS

-Al compilar el proyecto de prueba, las pruebas aparecen en el Explorador de pruebas. Si el Explorador de pruebas no está visible, elija Prueba en el menú de Visual Studio, elija Ventanas y, después, Explorador de pruebas. -Se puede elegir Ejecutar todas para ejecutar todas las pruebas o bien Ejecutar para elegir un subconjunto de pruebas que se desea ejecutar. Después de ejecutar un conjunto de pruebas, aparecerá un resumen de la serie de pruebas en la parte inferior de la ventana Explorador de pruebas. Seleccione una prueba para ver los detalles de esa prueba en el panel inferior.

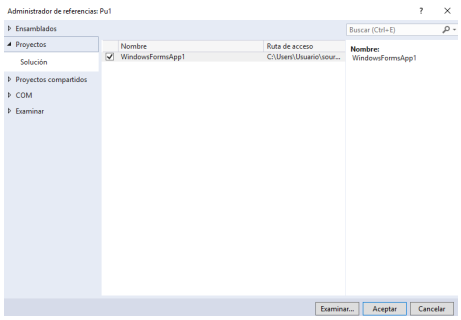
1. Pruebas Unitarias

   a) Paso 1: Pruebas unitarias: haremos pruebas unitarias para los métodos deposito, retiro.
   - Agregaremos un test de pruebas unitarias

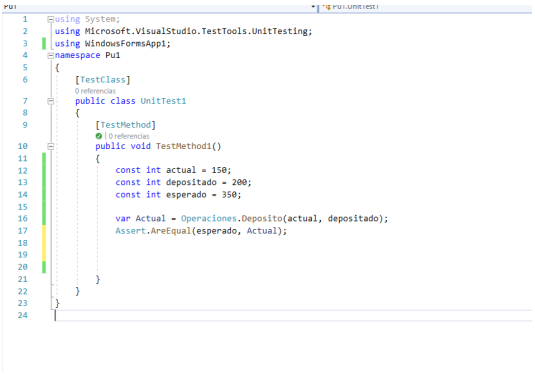


   b) Paso 2: Seguidamente el nuevo y tipo de proyecto.

   c) Paso 3: Luego referenciaremos el proyecto principal

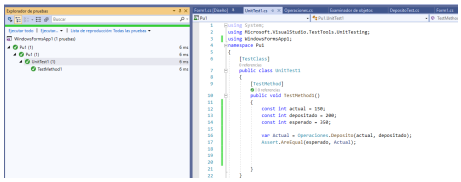


d) Paso 4: Luego probaremos los métodos :

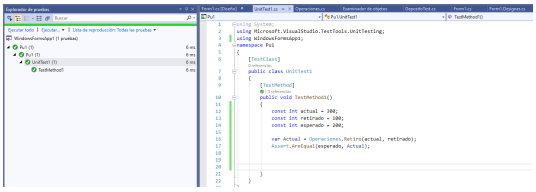


```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using WindowsFormsApp1;
namespace Pu1
{
    [TestClass]
    0 referencias
    public class UnitTest1
    {
        [TestMethod]
        0 referencias
        public void TestMethod1()
        {
            const int actual = 150;
            const int depositado = 200;
            const int esperado = 350;

            var Actual = Operaciones.Deposito(actual, depositado);
            Assert.AreEqual(esperado, Actual);


        }
    }
}
```

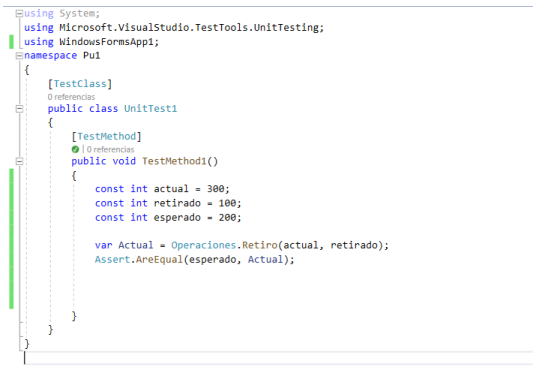
e) Paso 5: Esperamos mientras carga la pruba que se va desarrollar



f) Paso 6: Retiros



g) Paso 7: Finalmente aplicamos el codigo que se ejecurtar



```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using WindowsFormsApp1;
namespace Pu1
{
    [TestClass]
    0 referencias
    public class UnitTest1
    {
        [TestMethod]
        | 0 referencias
        public void TestMethod1()
        {
            const int actual = 300;
            const int retirado = 100;
            const int esperado = 200;

            var Actual = Operaciones.Retiro(actual, retirado);
            Assert.AreEqual(esperado, Actual);



        }
    }
}
```

5. CONCLUSIONES

[1] Torres, M. (2012). PROGRAMACION ORIENTADA A OBJETOS CON VISUAL BASIC 2012. Editorial Macro. Pág. 370[[1]

[2] Elman, J. y Lavin, M. (2014). Django ligero: utilizando REST, WebSockets y Backbone. Editor .ºReilly Media, Inc. Pág. 61

[3] Hugon, J. (2014). C# 5: desarrolle aplicaciones Windows con Visual Studio 2013. Ediciones ENI. Pág. 269