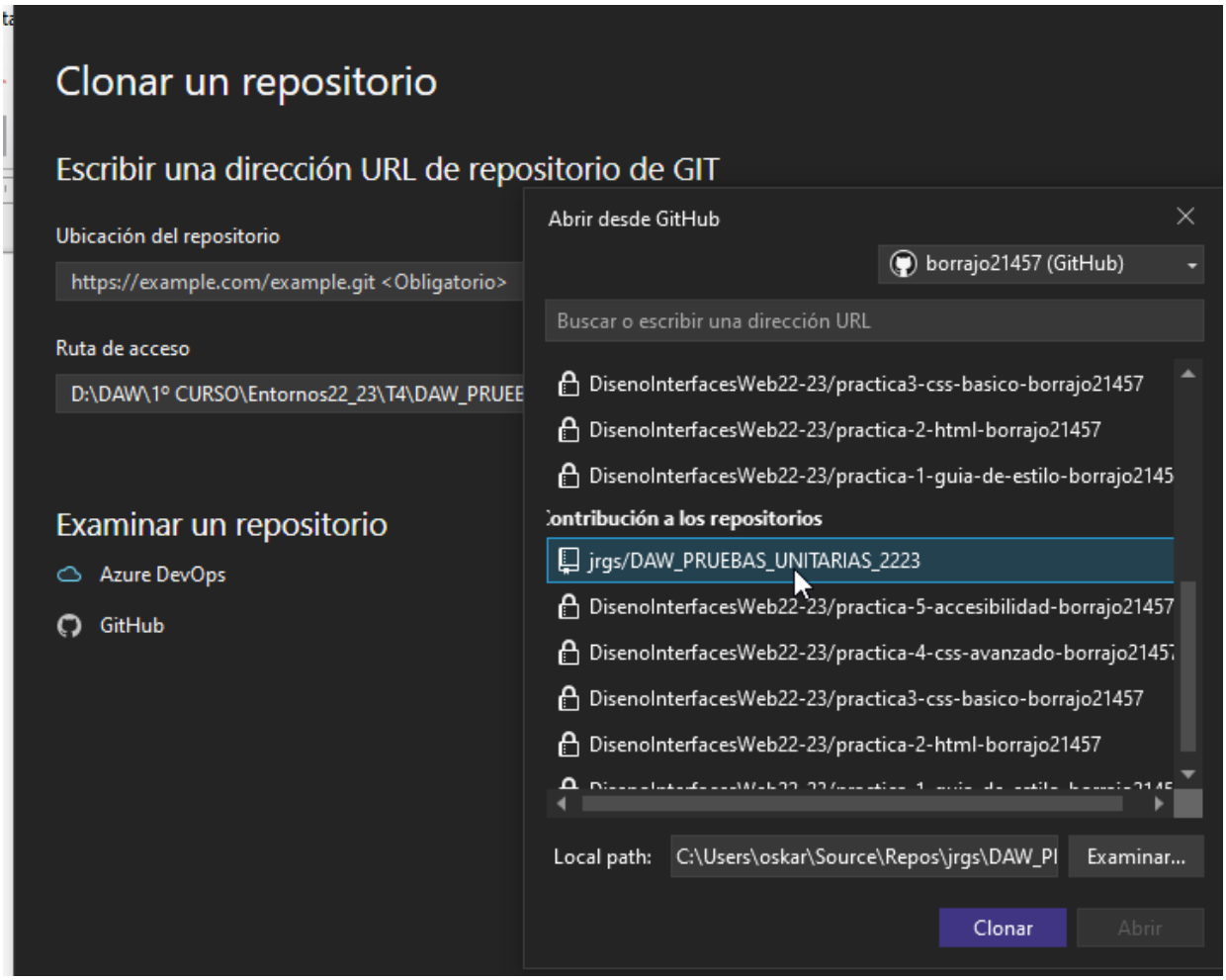
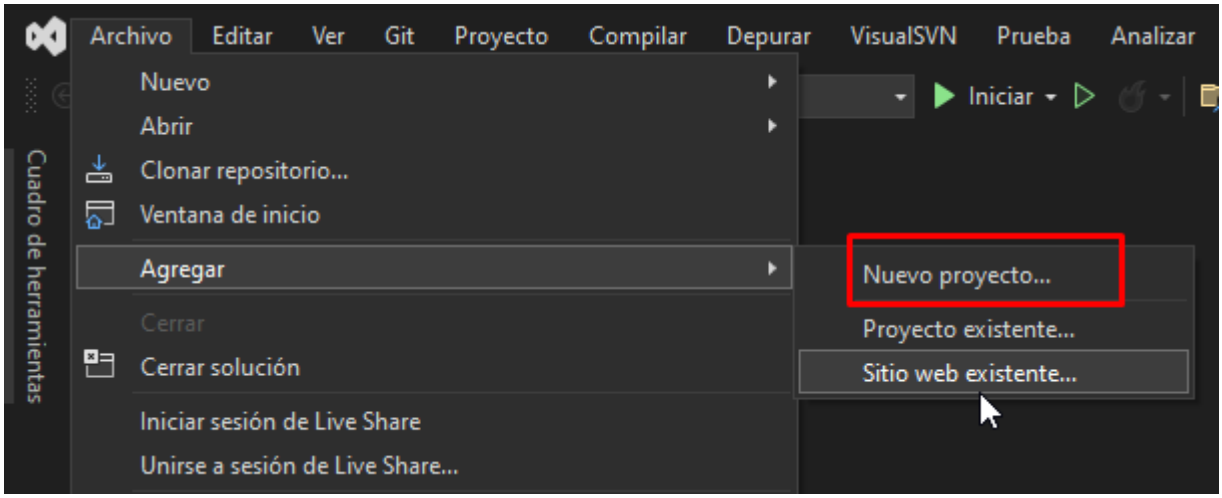
1. Conectate a GitHub y realiza un fork y clonamos el mismo.

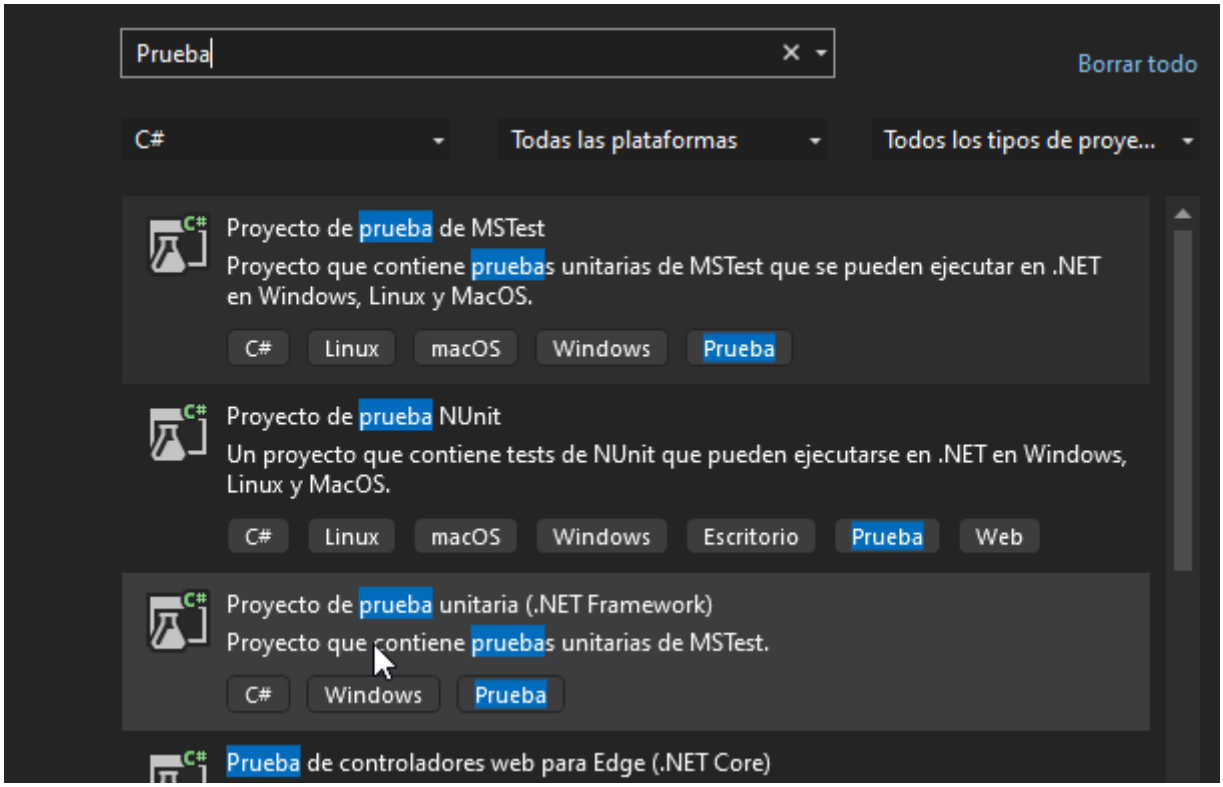


2- Compilar el proyecto

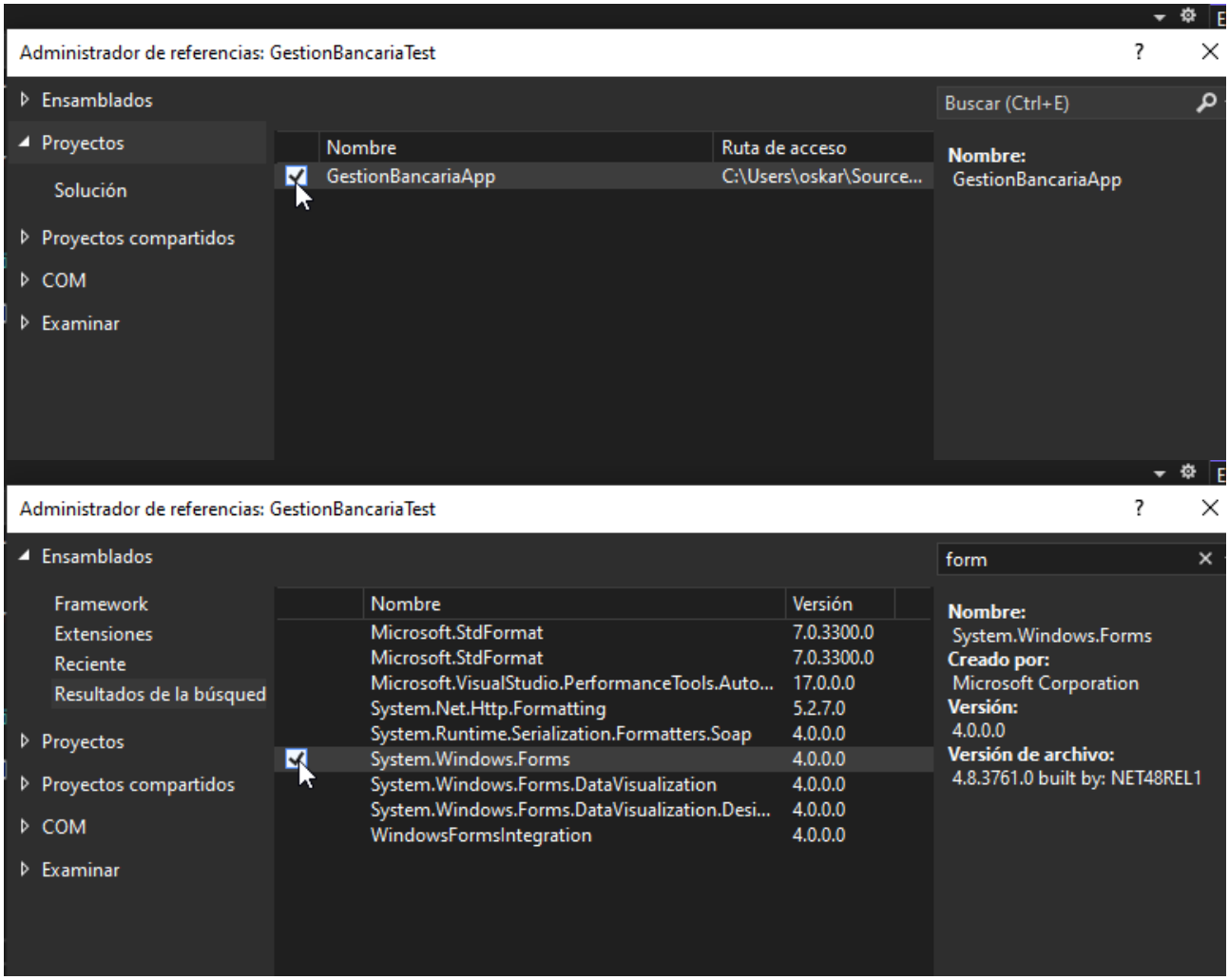
3-Crear un proyecto de prueba unitaria

3-1 Agregamos el proyecto de prueba unitaria



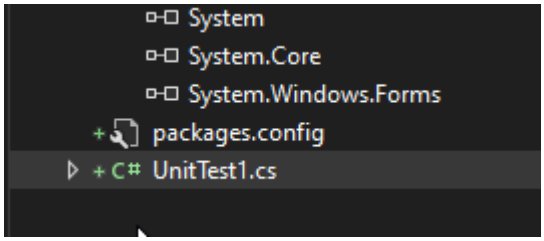


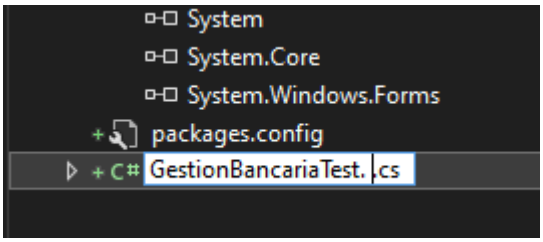
3-2 Agregamos las referencias al mismo



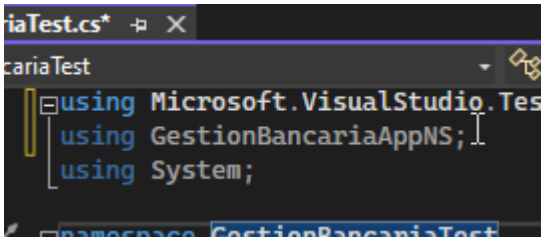
4. Crear la clase de prueba

4-1 Renombramos para reutilizar Unit Test1 a GestionBancariaTest.





4-2 Agregar una instrucción using al proyecto en prueba



Pasamos al ver las Clases equivalencia , valores frontera y diseño de método de prueba

RealizarReintegro(double cantidad)

1. Reintegro > saldo = error saldo insuficiente
2. Reintegro < 0 = error Cantidad no valida
3. Reintegro <= saldo = Valida

RealizarIngreso(double cantidad)

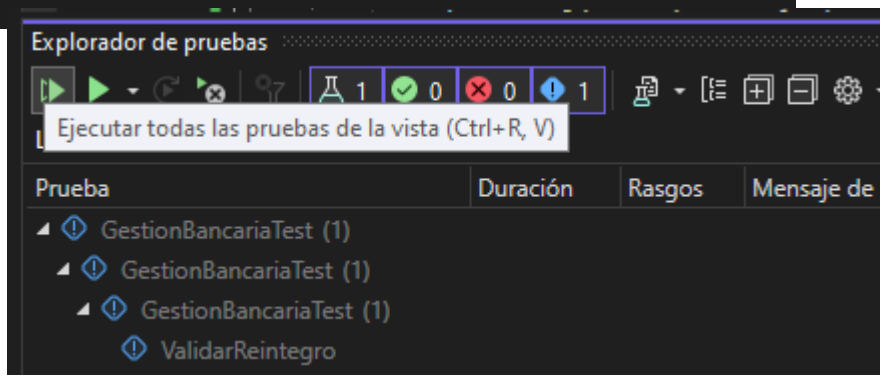
1. Ingreso <= 0 = error Cantidad no valida
2. Ingreso > 0 = Valida

	Clases equivalencia	Saldo	Reintegros	Resultado esperado	Prueba
Reintegro valido	1	1000	250	750	Valida
Retirada negativa	Cantidades negativas	1000	-500	1000	**Error C no valida**
Saldo insuficiente	Reintegro > saldo	1000	1001	1000	**Error S insuficiente**
			Ingresos		
Ingreso valido	2	1000	1	1001	Valida
Ingreso negativo	Cantidades negativas	1000	-1	1000 **Error C no valida**	**Error C no valida**

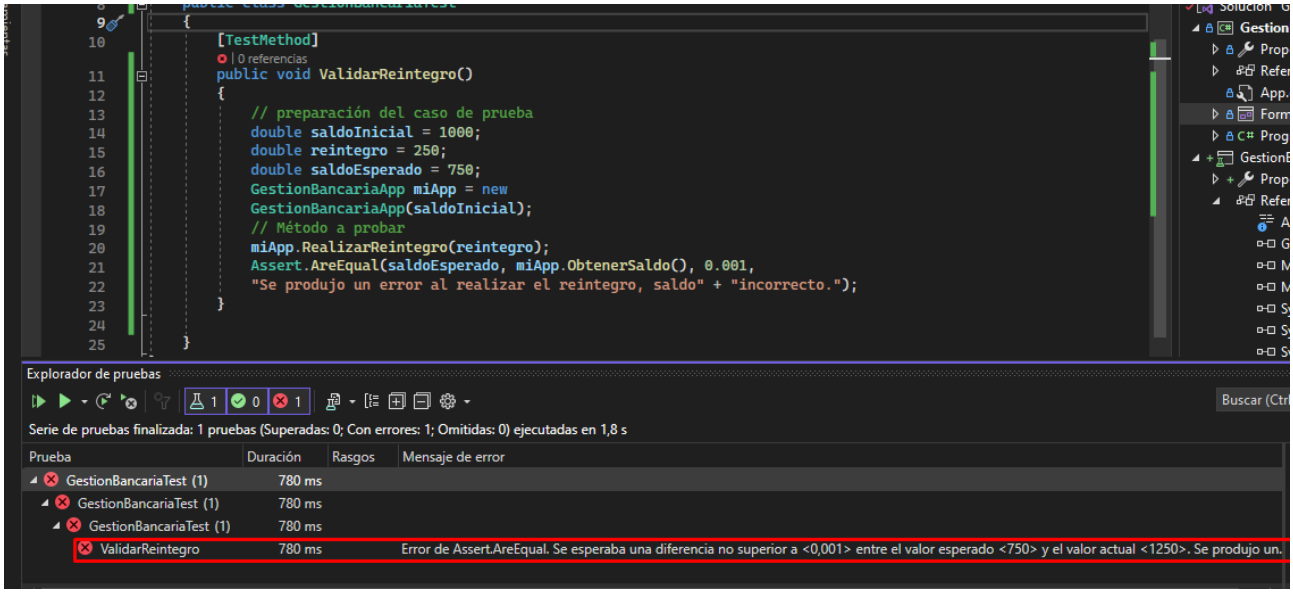
5. Crear el primer método de prueba

Tras implementar el test de prueba procedemos a su compilación y ejecución

```
[TestClass]
0 referencias
public class GestionBancariaTest
{
    [TestMethod]
    0 referencias
    public void ValidarReintegro()
    {
        // preparación del caso de prueba
        double saldoInicial = 1000;
        double reintegro = 250;
        double saldoEsperado = 750;
        GestionBancariaApp miApp = new
        GestionBancariaApp(saldoInicial);
        // Método a probar
        miApp.RealizarReintegro(reintegro);
        Assert.AreEqual(saldoEsperado, miApp.ObtenerSaldo(), 0.001,
        "Se produjo un error al realizar el reintegro, saldo" + "incorrecto.");
    }

}
```



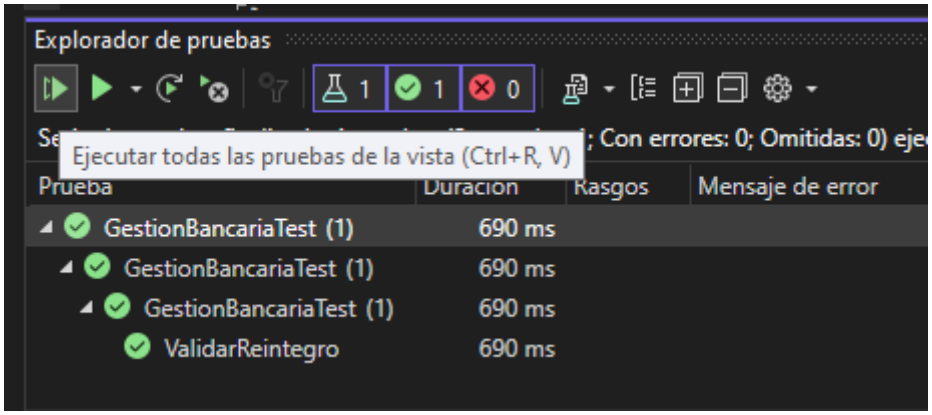
Y tras analizar los resultados dado que el prueba ha fallado dado que la cantidad reintegro se incrementa del saldo de cuenta en lugar de decrementarse dado que se esperaban 750 y la prueba arroja como resultado 1250



Corregimos el error en el codigo

```
public int RealizarReintegro(double cantidad
{
    if (cantidad <= 0)
        return ERR_CANTIDAD_NO_VALIDA;
    if (saldo < cantidad)
        return ERR_SALDO_INSUFICIENTE;

    saldo -= cantidad;
    return 0;
}
```

Volvemos a ejecutar la prueba

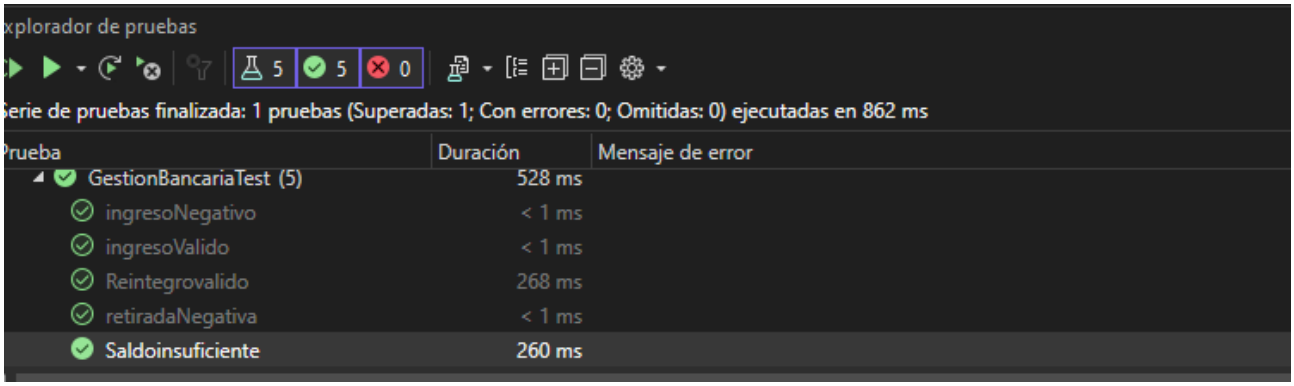


Estudia el método RealizarReintegro.

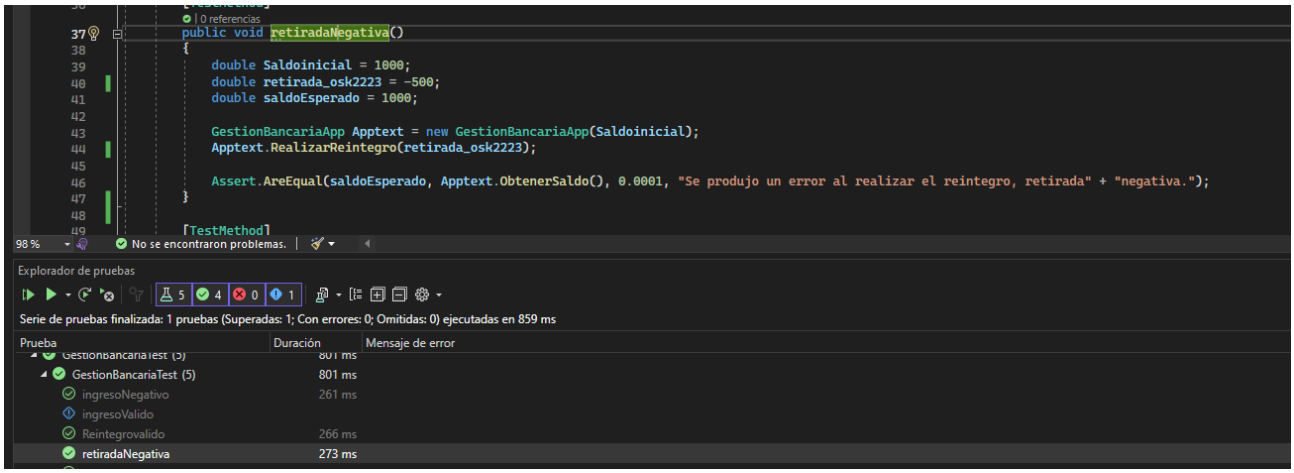
1=El método produce ERR_SALDO_INSUFICIENTE si la cantidad a retirar es mayor que el saldo.

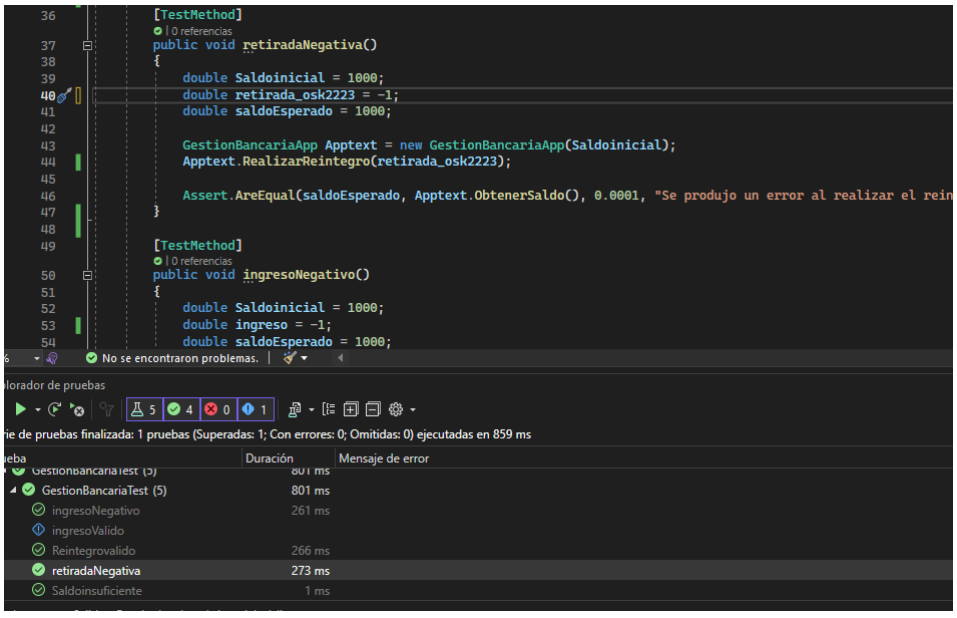
```
[TestMethod]
0 referencias
public void Saldoinsuficiente()
{
    double Saldoinicial = 1000;
    double retirada_OSK2223 = 1001;
    double saldoEsperado = 1000;

    GestionBancariaApp Apptext = new GestionBancariaApp(Saldoinicial);
    Apptext.RealizarReintegro(retirada_OSK2223);

    Assert.AreEqual(saldoEsperado, Apptext.ObtenerSaldo(), "Se produjo un error al realizar el
}
```

Y ejecutamos el test



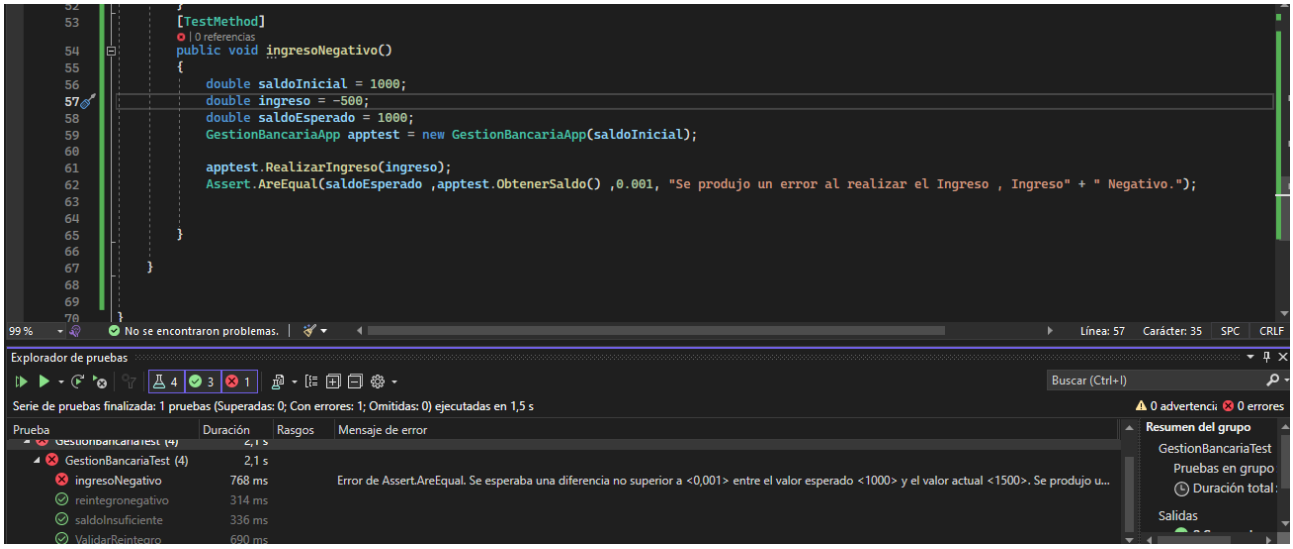
2 ERR_CANTIDAD_NO_VALIDA si la cantidad a retirar es menor que cero.



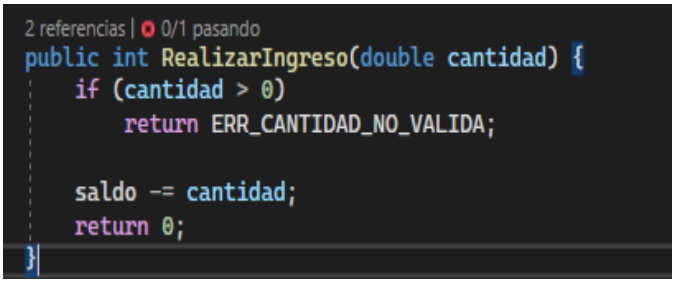


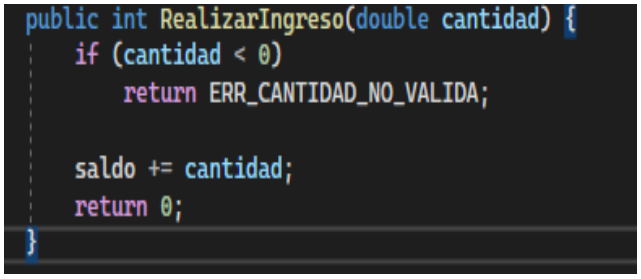
Estudia el método RealizarIngreso

1-- Cantidad a ingresar negativa

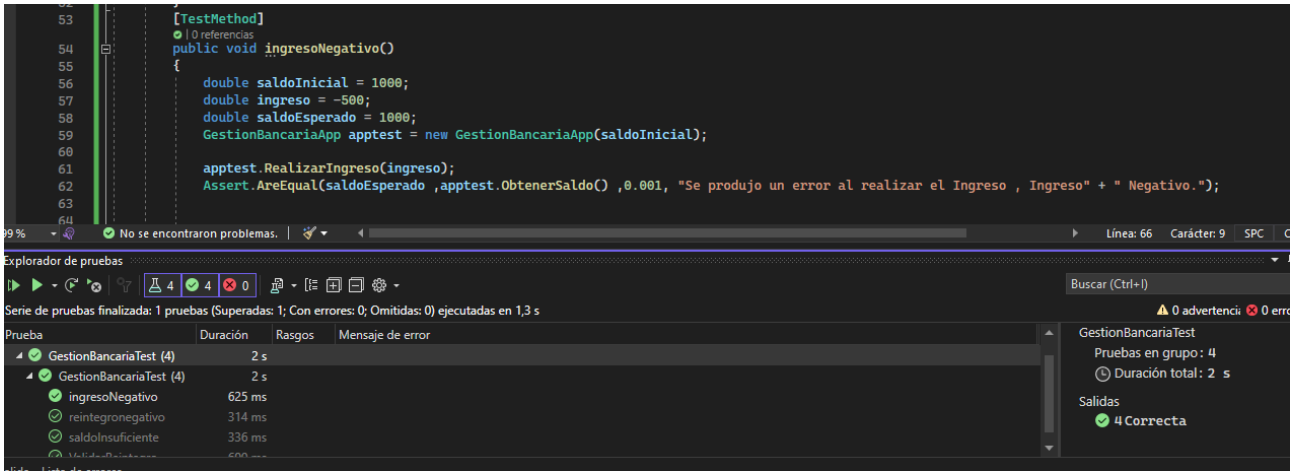


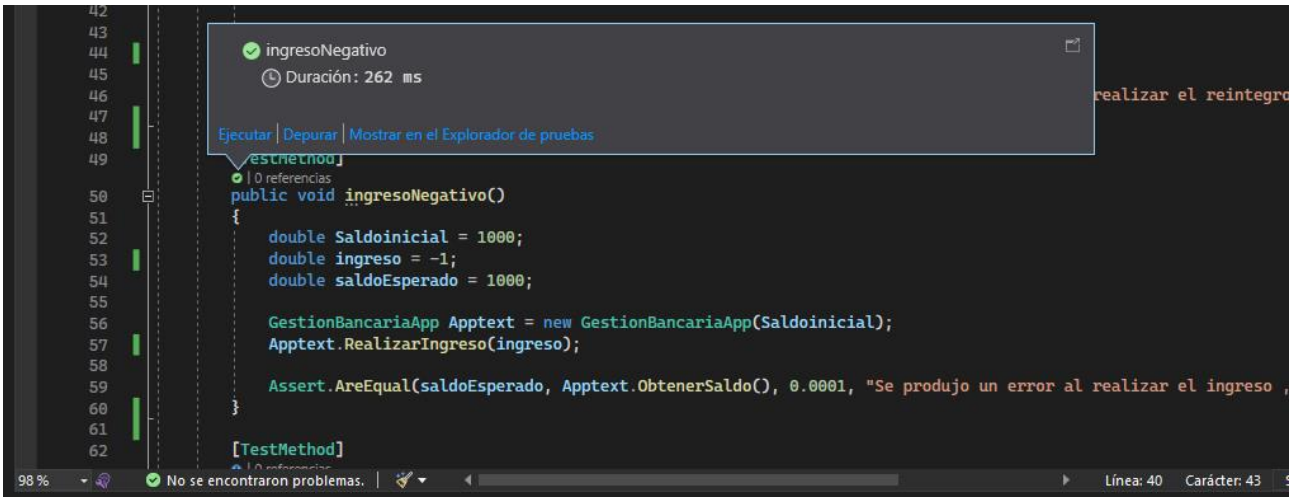
Al haber fallado el test y tras comprobar que la cantidad a ingresar ,aun siendo negativa ,se suma al saldo , indica que hemos encontrado otro error en la función RealizarIngreso y pasamos a revisar el código .



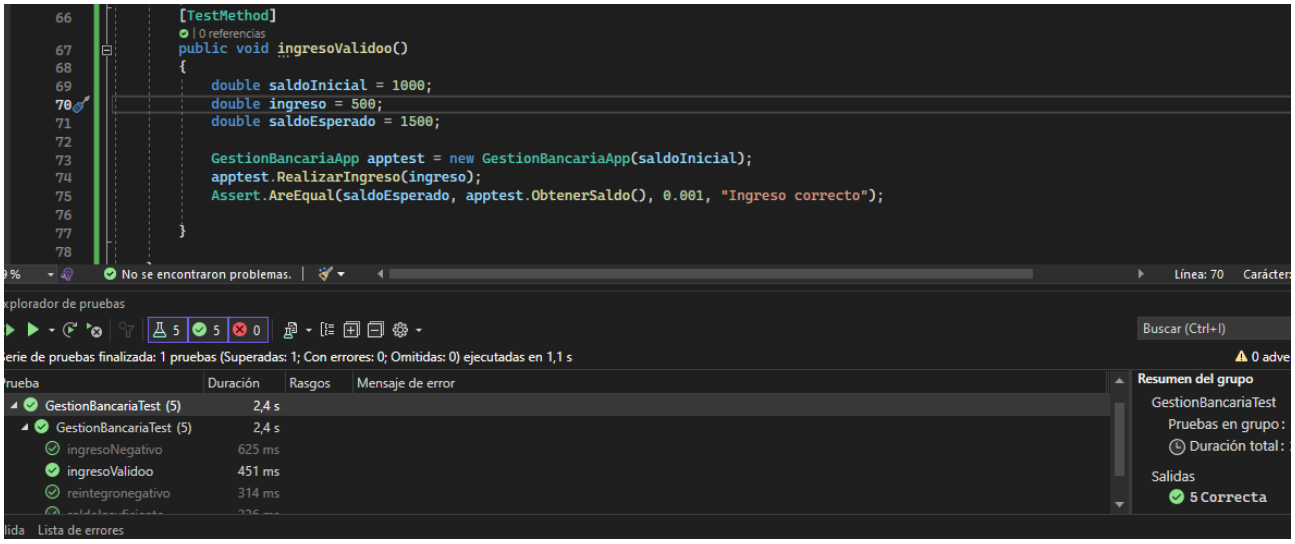


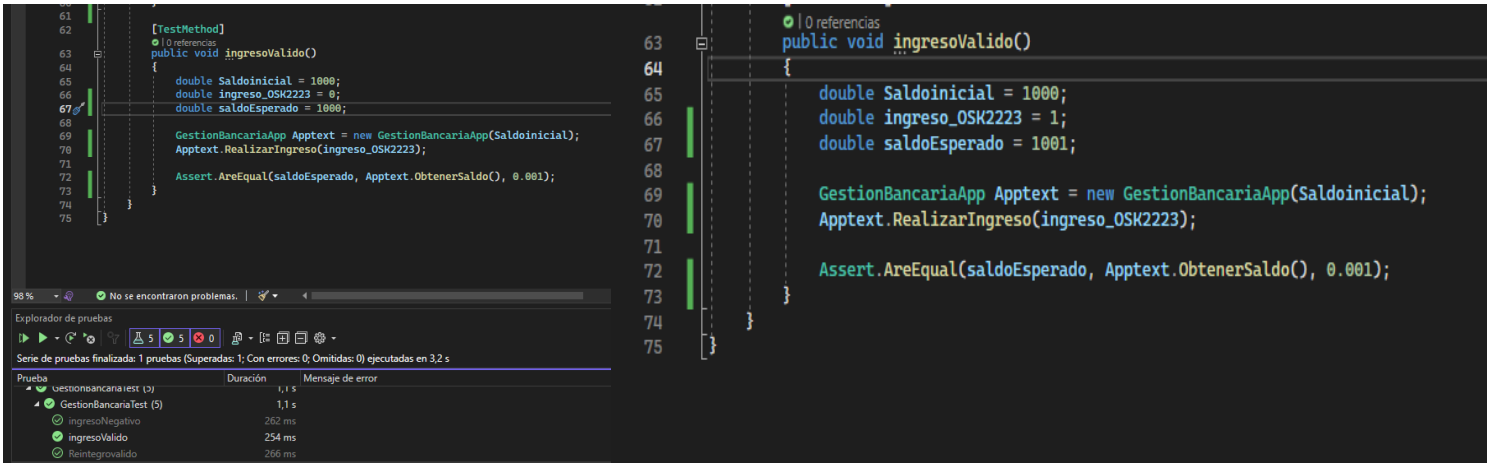
Tras rectificar los errores corremos otra vez lo test.





2-- Ingreso valido.





8. Pruebas que esperan excepciones

8-1modificar los métodos para incluir la generación de las siguientes excepciones

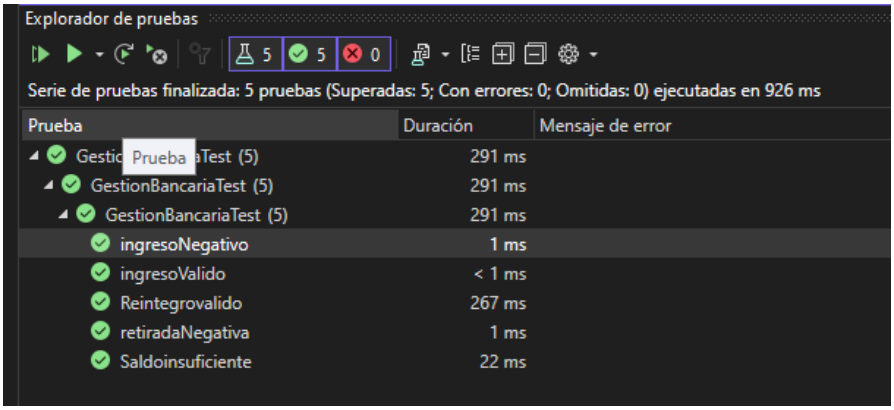
```
7 referencias | 5/5 pasando
public double ObtenerSaldo() { return saldo; }

4 referencias | 3/3 pasando
public int RealizarReintegro(double cantidad)
{
    if (cantidad <= 0)
        throw new ArgumentOutOfRangeException("La cantidad indicada no es válida.");

    if (saldo < cantidad)
        throw new ArgumentOutOfRangeException("Saldo Insuficiente");

    saldo -= cantidad;
    return 0;
}

3 referencias | 2/2 pasando
public int RealizarIngreso(double cantidad) {
    if (cantidad < 0)
        throw new ArgumentOutOfRangeException("La cantidad indicada no es válida.");

    saldo += cantidad;
    return 0;
}
```

9--Cree los métodos de prueba necesarios para controlar el resto de errores (excepciones

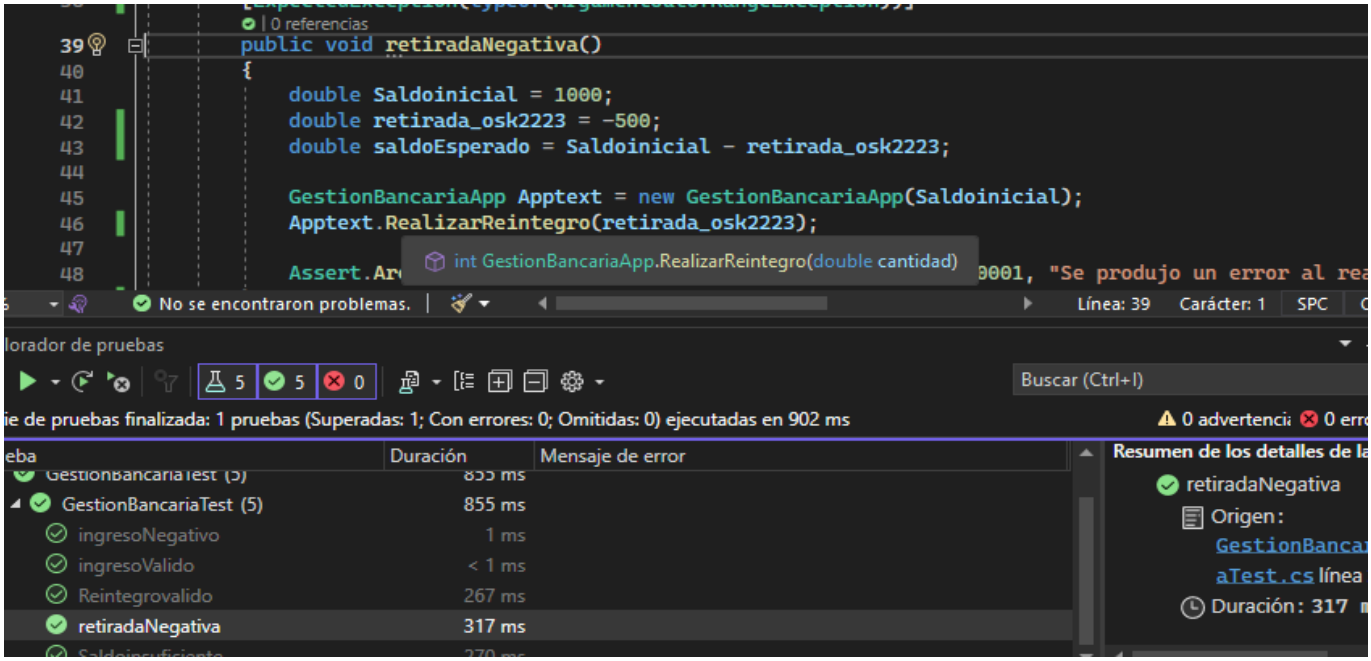
```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
| 0 referencias
public void Saldoinsuficiente()
{
    double Saldoinicial = 1000;
    double retirada_OSK2223 = 1001;
    double saldoEsperado = 1000;

    GestionBancariaApp Apptext = new GestionBancariaApp(Saldoinicial);
    Apptext.RealizarReintegro(retirada_OSK2223);

    Assert.AreEqual(saldoEsperado, Apptext.ObtenerSaldo(), "Se produjo un error al realizar el reintegro, saldo" + " insuficiente.");
}
```

```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
| 0 referencias
public void retiradaNegativa()
{
    double Saldoinicial = 1000;
    double retirada_osk2223 = -500;
    double saldoEsperado = 1000;

    GestionBancariaApp Apptext = new GestionBancariaApp(Saldoinicial);
    Apptext.RealizarReintegro(retirada_osk2223);

    Assert.AreEqual(saldoEsperado, Apptext.ObtenerSaldo(), 0.0001, "Se produjo un error al realizar el reintegro, retirada" + "negativa.");
}
```

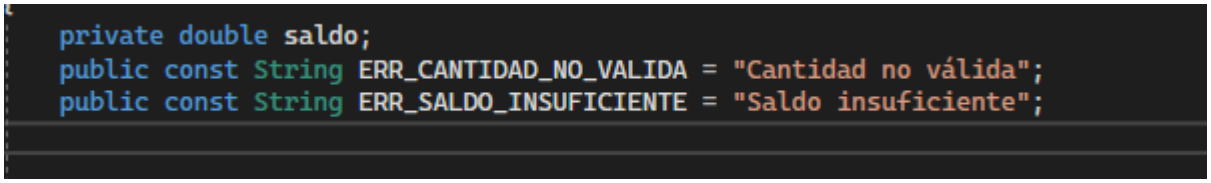
```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
| 0 referencias
public void ingresoNegativo()
{
    double Saldoinicial = 1000;
    double ingreso = -1;
    double saldoEsperado = 1000;

    GestionBancariaApp Apptext = new GestionBancariaApp(Saldoinicial);
    Apptext.RealizarIngreso(ingreso);

    Assert.AreEqual(saldoEsperado, Apptext.ObtenerSaldo(), 0.0001, "Se produjo un error al realizar el ingreso , ingreso" + " negativo.");
}
```

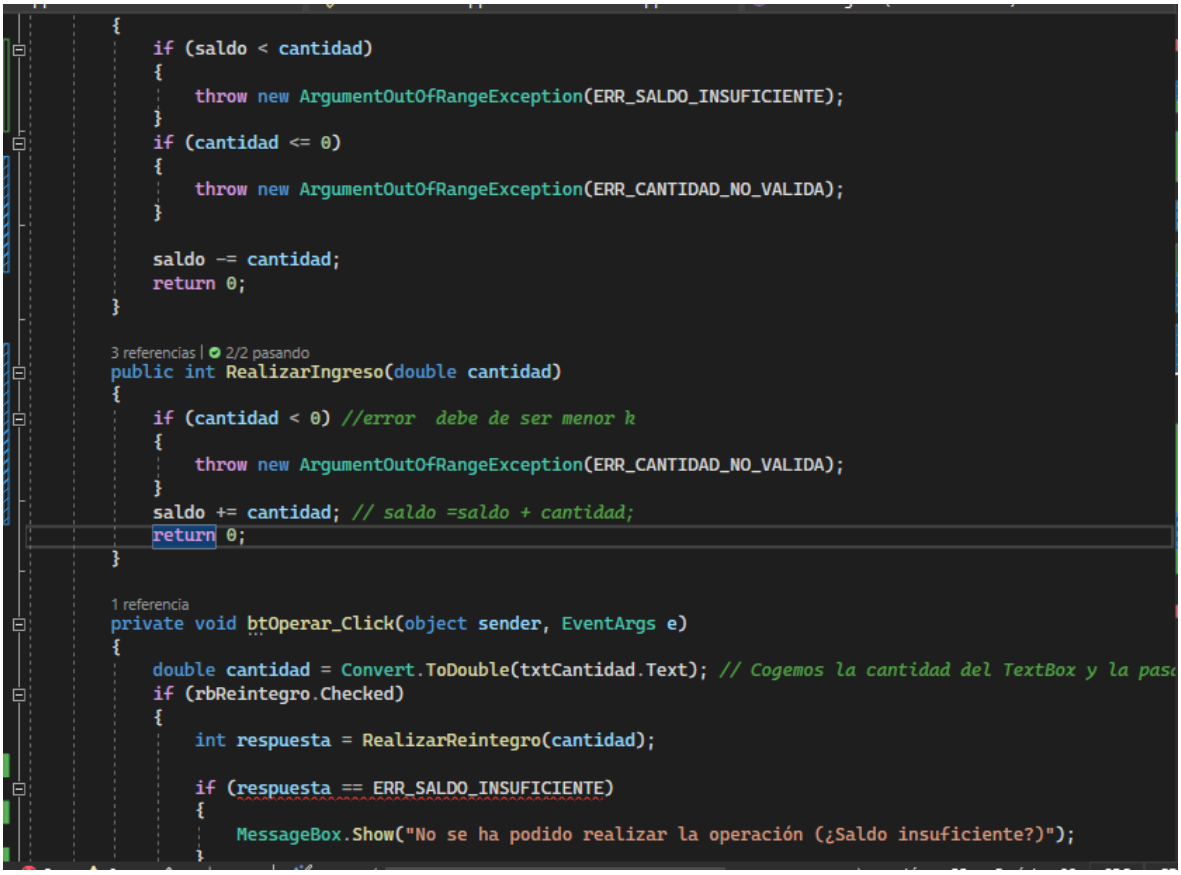


10. Mejorar el código de prueba dado que en la implementación del método RetiradaNegativa este no detecta es error de una cantidad negativa , sumándola al saldo esperado ;



Redefiniremos nuestras constantes de error y la forma de generar las excepciones





```
        {
            if (saldo < cantidad)
            {
                throw new ArgumentOutOfRangeException(ERR_SALDO_INSUFICIENTE);
            }
            if (cantidad <= 0)
            {
                throw new ArgumentOutOfRangeException(ERR_CANTIDAD_NO_VALIDA);
            }

            saldo -= cantidad;
            return 0;
        }

        3 referencias | 2/2 pasando
        public int RealizarIngreso(double cantidad)
        {
            if (cantidad < 0) //error  debe de ser menor k
            {
                throw new ArgumentOutOfRangeException(ERR_CANTIDAD_NO_VALIDA);
            }
            saldo += cantidad; // saldo =saldo + cantidad;
            return 0;
        }

        1 referencia
        private void btOperar_Click(object sender, EventArgs e)
        {
            double cantidad = Convert.ToDouble(txtCantidad.Text); // Cogemos la cantidad del TextBox y la pas
            if (rbReintegro.Checked)
            {
                int respuesta = RealizarReintegro(cantidad);

                if (respuesta == ERR_SALDO_INSUFICIENTE)
                {
                    MessageBox.Show("No se ha podido realizar la operación (¿Saldo insuficiente?)");
```

Reescribir el código de prueba

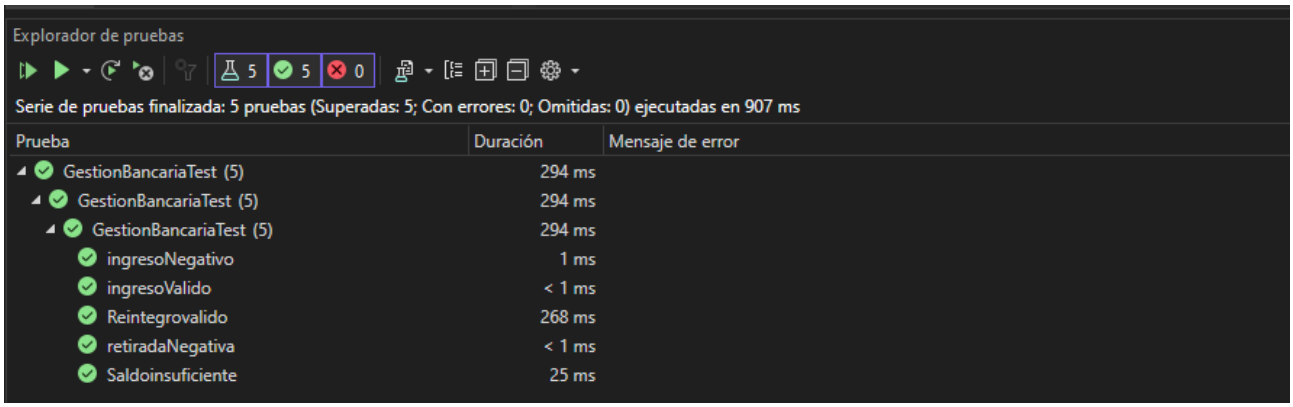
```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
0 referencias
public void retiradaNegativa()
{
    double Saldoinicial = 1000;
    double retirada_osk2223 = -500;
    double saldoEsperado = Saldoinicial - retirada_osk2223;

    GestionBancariaApp Apptext = new GestionBancariaApp(Saldoinicial);
    try
    {
        Apptext.RealizarReintegro(retirada_osk2223);
    }
    catch (ArgumentOutOfRangeException excepcion)
    {
        StringAssert.Contains(excepcion.Message,GestionBancariaApp.ERR_CANTIDAD_NO_VALIDA);
        return;
    }

    Assert.Fail("Error. Se debía haber producido una excepción.");
}
```

```
        [TestMethod]
        [ExpectedException(typeof(ArgumentOutOfRangeException))]
        | 0 referencias
        public void Saldoinsuficiente()
        {
            double Saldoinicial = 1000;
            double retirada_OSK2223 = 1001;
            double saldoEsperado = Saldoinicial - retirada_OSK2223;

            GestionBancariaApp Apptext = new GestionBancariaApp(Saldoinicial);
            try
            {
                Apptext.RealizarReintegro(retirada_OSK2223);
            }
            catch (ArgumentOutOfRangeException escepcion)
            {
                StringAssert.Contains(escepcion.Message, GestionBancariaApp.ERR_SALDO_INSUFICIENTE);
                return;

            }

            Assert.Fail("Error. Se debía haber producido una excepción.");
        }
```

```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
| 0 referencias
public void ingresoNegativo()
{
    double Saldoinicial = 1000;
    double ingreso = -1;
    double saldoEsperado = 1000;

    GestionBancariaApp Apptext = new GestionBancariaApp(Saldoinicial);
    try
    {
        Apptext.RealizarIngreso(ingreso);
    }
    catch (ArgumentOutOfRangeException escepcion)
    {
        StringAssert.Contains(escepcion.Message, GestionBancariaApp.ERR_CANTIDAD_NO_VALIDA);
    }
    Assert.Fail("Error. Se debía haber producido una excepción.");
}

[TestMethod]
```

Redefinir, en el código de la aplicación

```
private void btOperar_Click(object sender, EventArgs e)
{
    double cantidad = Convert.ToDouble(txtCantidad.Text); // Cogemos la cantidad del TextBox y la pasamos a
    if (rbReintegro.Checked)
    {
        try
        {
            RealizarReintegro(cantidad);
            MessageBox.Show("Transacción realizada.");
        }
        catch (Exception error)
        {
            if (error.Message.Contains(ERR_SALDO_INSUFICIENTE))
                MessageBox.Show("No se ha podido realizar la operación (¿Saldo insuficiente ?)");
            else
             if (error.Message.Contains(ERR_CANTIDAD_NO_VALIDA))
                MessageBox.Show("Cantidad no válida, sólo se admiten cantidades  positivas.");
        }
    }
    else
    {
        try
        {
            RealizarIngreso(cantidad);
            MessageBox.Show("Transacción realizada.");
        }
        catch (Exception error)
        {
            if (error.Message.Contains(ERR_CANTIDAD_NO_VALIDA))
                MessageBox.Show("Cantidad no válida, sólo se admiten cantidades positivas.");
        }
    }
    txtSaldo.Text = ObtenerSaldo().ToString();
}
```



[borrajo21457/DAW_PRUEBAS_UNITARIAS_2223 (github.com)]