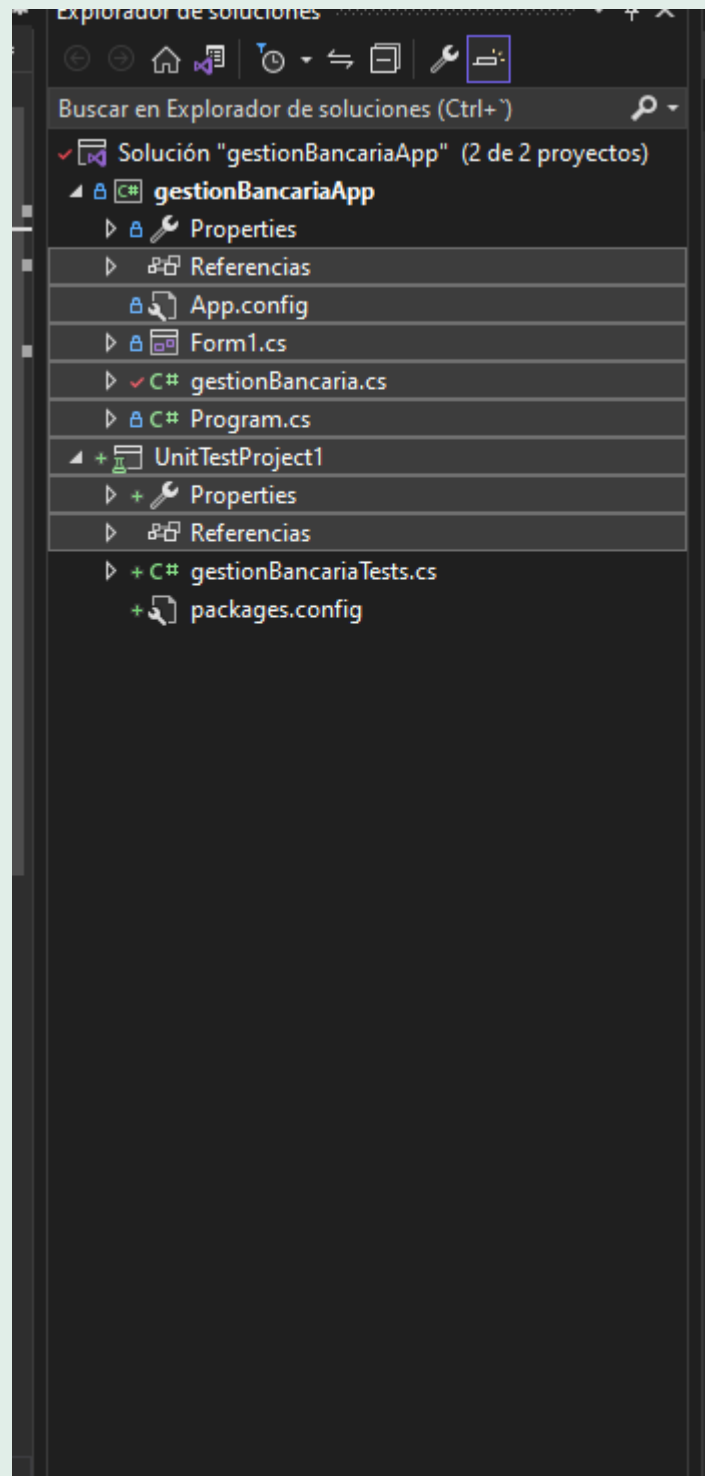


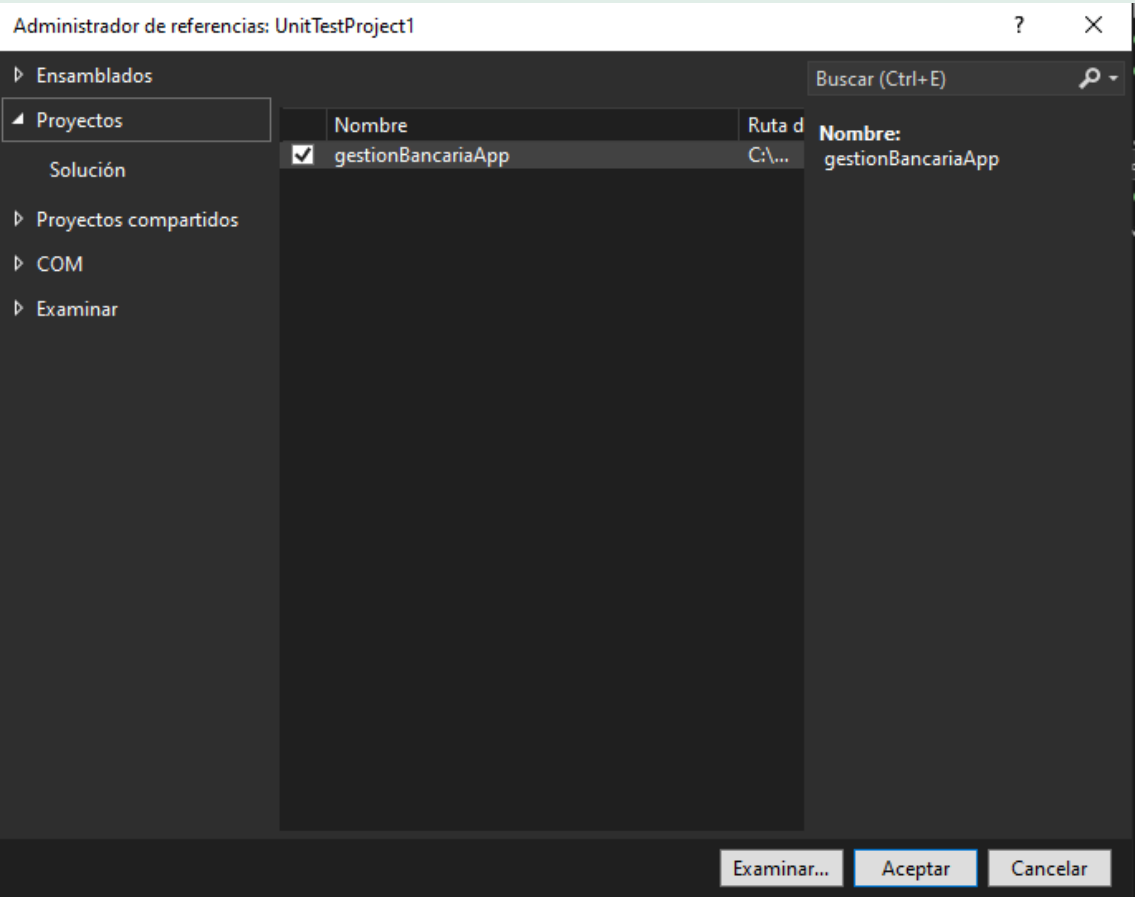
PRACTICA 4-3

ENTORNOS DE DESARROLLO

https://github.com/leocovess/DAM_PRUEBAS_UNITARIAS_2223

Prueba unitaria ya creada





Agregamos referencia gestiónBancariaApp

4. Creamos la clase de prueba

```

5      {
6          [TestClass]
7          0 referencias | 0 cambios | 0 autores, 0 cambios
8          public class gestionBancariaTests
9          {
10             [TestMethod]
11             0 referencias | 0 cambios | 0 autores, 0 cambios
12             public void validarMetodoIngreso()
13             {
14                 // preparación del caso de prueba

```

5. Creamos el primer método de prueba

```

[TestMethod]
0 referencias | 0 cambios | 0 autores, 0 cambios
public void validarMetodoIngreso()
{
    // preparación del caso de prueba
    double saldoInicial = 1000;
    double ingreso = 500;
    double saldoActual = 0;
    double saldoEsperado = 1500;

    gestionBancaria cuenta = new gestionBancaria(saldoInicial);

    // Método a probar
    cuenta.realizarIngreso(ingreso);

    // afirmación de la prueba (valor esperado Vs. Valor obtenido)
    saldoActual = cuenta.obtenerSaldo();

    Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
}

```

6. Compilar y ejecutar la prueba

En este caso nos da error y al depurar pruebas nos lleva a la función realizarIngreso()

General

No hay controles utilizables en este grupo. Arrastre un elemento a este texto y agréguelo al cuadro de herramientas.

```

7      {
8          [TestClass]
9          0 referencias | 0 cambios | 0 autores, 0 cambios
10         public class gestionBancariaTests
11         {
12             [TestMethod]
13             0 referencias | 0 cambios | 0 autores, 0 cambios
14             public void validarMetodoIngreso()
15             {
16                 // preparación del caso de prueba
17                 double saldoInicial = 1000;
18                 double ingreso = 500;
19                 double saldoActual = 0;
20                 double saldoEsperado = 1500;
21
22                 gestionBancaria cuenta = new gestionBancaria(saldoInicial);
23
24                 // Método a probar
25                 cuenta.realizarIngreso(ingreso);
26
27                 // afirmación de la prueba (valor esperado Vs. Valor obtenido)
28                 saldoActual = cuenta.obtenerSaldo();
29
30                 Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
31             }
32         }
33     }

```

88% 0 No se encontraron problemas: 0 Puntos de interrupción Salida X Línea 21 Carácter 45 SPC CRLF

Mostrar salida de: Compilación

Operación Compilar iniciada. Proyecto: gestionBancariaApp, configuración: Debug Any CPU -----

1> gestionBancariaApp -> C:\Users\leocovguz\Desktop\Entornos 4-3\gestionBancariaApp\bin\Debug\gestionBancariaApp.exe

2> ----- Operación Compilar iniciada: Proyecto: UnitTestProject1, configuración: Debug Any CPU -----

2> UnitTestProject1 -> C:\Users\leocovguz\Desktop\Entornos 4-3\UnitTestProject1\bin\Debug\UnitTestProject1.dll

===== Compilación: 2 correcto, 0 erróneo, 0 actualizado, 0 omitido =====

===== Compilar se inició en 4:24 PM y tomó 02,238 segundos =====

Explorador de pruebas

Serie de pruebas finalizada: 1 pruebas (Superadas: 0; Con errores: 1; Omitidas: 0) ejecutadas en 923 ms

Prueba	Duración	Rasgos	Mensaje de error
UnitTestProject1 (1)	294 ms		
UnitTestProject1 (1)	294 ms		
gestionBancariaTests (1)	294 ms		
validarMetodoIngreso	294 ms		Error de Assert.Are...

Resumen del grupo

UnitTestProject1

Pruebas en grupo: 1

○ Duración total: 294 ms

Salidas

✖ 1 Con error

Explorador de soluciones Cambios de GIT: Entornos

Comprobamos que el error esta en esa función

```
47     }
48
49     2 referencias | 0/1 pasando | 0 cambios | 0 autores, 0 cambios
50     public void realizarIngreso(double cantidad)
51     {
52         if (cantidad < 0)
53         {
54             //mostrarError(ERR_CANTIDAD_INDICADA_NEGATIVA);
55             throw new ArgumentOutOfRangeException("Cantidad negativa");
56         }
57         else
58         {
59             if (cantidad > 0)
60                 saldo -= cantidad; //
61         }
62     }
63 }
```

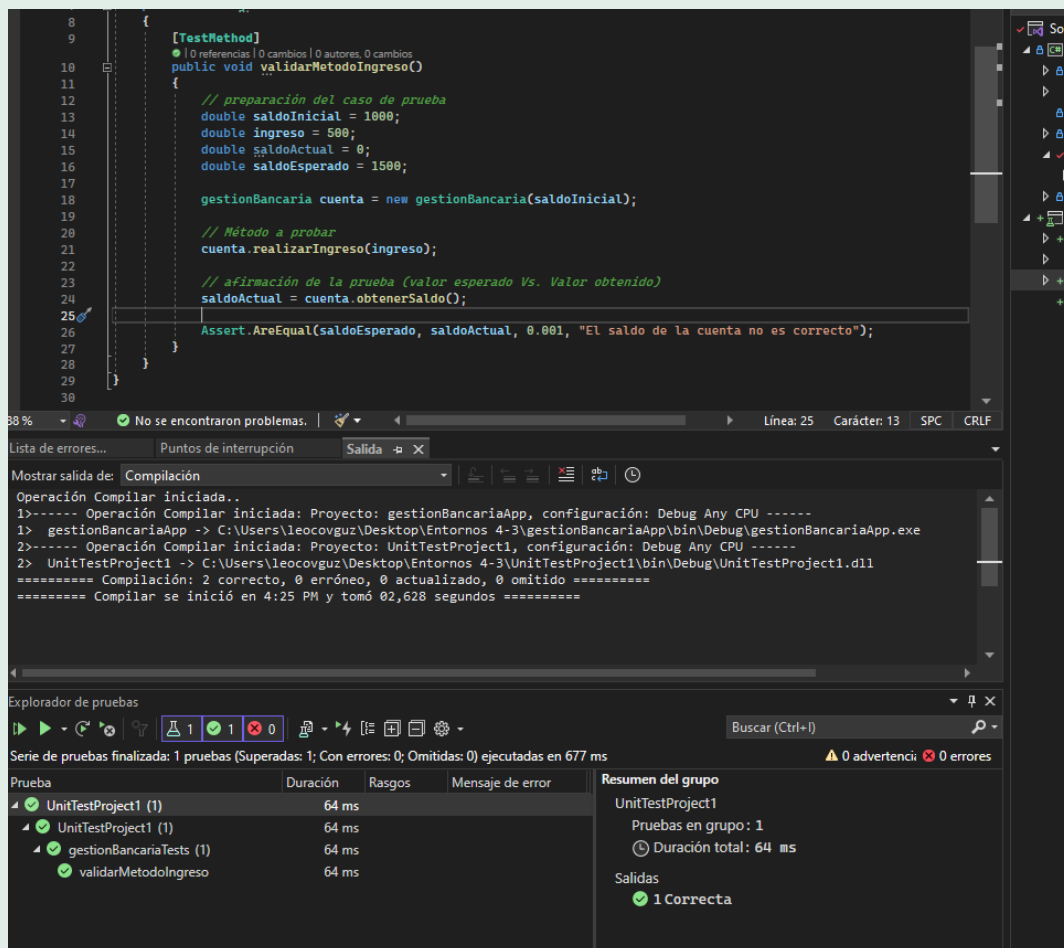
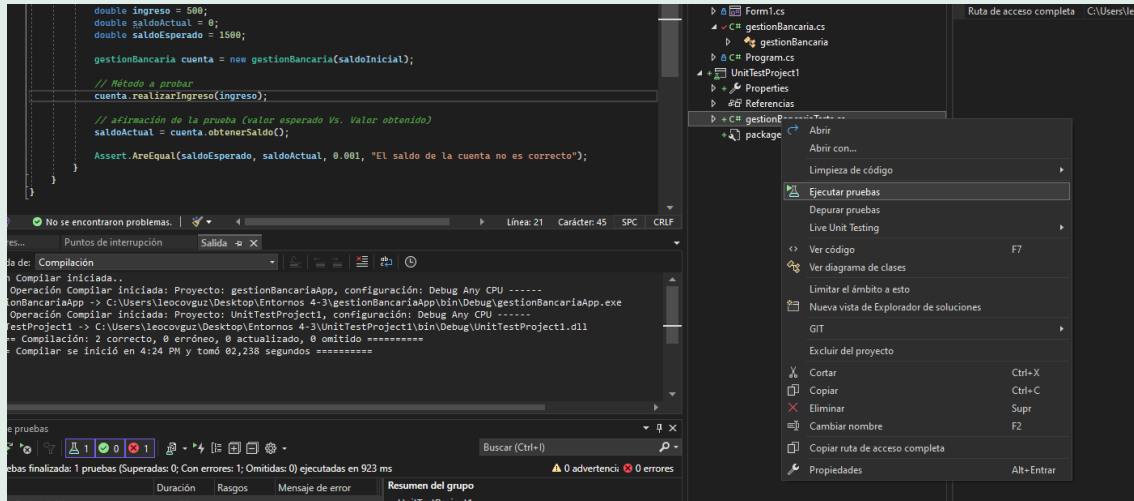
Tenia un fallo en la línea 60 en el signo negativo, pues lo cambiamos

```
public void realizarIngreso(double cantidad)
{
    if (cantidad < 0)
    {
        //mostrarError(ERR_CANTIDAD_INDICADA_NEGATIVA);
        throw new ArgumentOutOfRangeException("Cantidad negativa");
    }
    else
    {
        if (cantidad > 0)
            saldo += cantidad; //
    }
}
```

Lo sustituimos por un "+" ya que al ingresar sumas el ingreso con el saldo de la cuenta Bancaria. (Era un error intencionado según el PDF)

Volvemos a ejecutar la prueba y vemos que no da error

Click derecho en gestionBancariaTests, ejecutar pruebas y abajo saldrán el explorador de pruebas. En caso de que no salgo, pulsamos en la barra superior en Ver > click en 'Explorador de pruebas'



Punto 7 (clases de equivalencia y valores limites)

Clases de equivalencia → Ingreso

Caso 1: Que el ingreso sea > 0 (valor positivo) = true

```
[TestMethod]
0 | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void validarMetodoIngreso()
{
    // preparación del caso de prueba
    double saldoInicial = 1000;
    double ingreso = 500;
    double saldoActual = 0;
    double saldoEsperado = 1500;

    gestionBancaria cuenta = new gestionBancaria(saldoInicial);

    // Método a probar
    cuenta.realizarIngreso(ingreso);

    // afirmación de la prueba (valor esperado Vs. Valor obtenido)
    saldoActual = cuenta.obtenerSaldo();

    Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
}
```

Caso 2: Que el ingreso sea < 0 (valor negativo) = false

```
0 | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void validarMetodoIngresoNegativo()
{
    // preparación del caso de prueba
    double saldoInicial = 1000;
    double ingreso = -500;
    double saldoActual = 0;
    double saldoEsperado = 1500;

    gestionBancaria cuenta = new gestionBancaria(saldoInicial);

    // Método a probar
    cuenta.realizarIngreso(ingreso);

    // afirmación de la prueba (valor esperado Vs. Valor obtenido)
    saldoActual = cuenta.obtenerSaldo();

    Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
}
```

Caso 3: Que el saldo Inicial sea negativo = true

```
[TestMethod]
0 | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void validarMetodoIngresoSaldoInicialNegativo()
{
    // preparación del caso de prueba
    double saldoInicial = -1000;
    double ingreso = 500;
    double saldoActual = 0;
    double saldoEsperado = -500;

    gestionBancaria cuenta = new gestionBancaria(saldoInicial);

    // Método a probar
    cuenta.realizarIngreso(ingreso);

    // afirmación de la prueba (valor esperado Vs. Valor obtenido)
    saldoActual = cuenta.obtenerSaldo();

    Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
}
```

Caso 4: Que el ingreso sea 0 = false

```
[TestMethod]
0 | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void validarMetodoIngresoSiendo0()
{
    // preparación del caso de prueba
    double saldoInicial = 1000;
    double ingreso = 0;
    double saldoActual = 0;
    double saldoEsperado = 1000;

    gestionBancaria cuenta = new gestionBancaria(saldoInicial);

    // Método a probar
    cuenta.realizarIngreso(ingreso);

    // afirmación de la prueba (valor esperado Vs. Valor obtenido)
    saldoActual = cuenta.obtenerSaldo();

    Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
}
```

Caso 5: Que el ingreso sea NULL = false

Sin imagen

Clases de equivalencia → Reintegro

Caso 1: Que el reintegro sea < saldo Inicial = true

```
[TestMethod]
0 | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void validarMetodoReintegro()
{
    // preparación del caso de prueba
    double saldoInicial = 1000;
    double reintegro = 500;
    double saldoActual = 0;
    double saldoEsperado = 500;

    gestionBancaria cuenta = new gestionBancaria(saldoInicial);

    // Método a probar
    cuenta.realizarReintegro(reintegro);

    // afirmación de la prueba (valor esperado Vs. Valor obtenido)
    saldoActual = cuenta.obtenerSaldo();

    Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
}
```

Caso 2: Que el reintegro sea > saldo Inicial = false

```
[TestMethod]
0 | 0 referencias | 0 cambios | 0 autores, 0 cambios
public void validarMetodoReintegroNegativo()
{
    // preparación del caso de prueba
    double saldoInicial = 500;
    double reintegro = 1000;
    double saldoActual = 0;
    double saldoEsperado = -500;

    gestionBancaria cuenta = new gestionBancaria(saldoInicial);

    // Método a probar
    cuenta.realizarReintegro(reintegro);

    // afirmación de la prueba (valor esperado Vs. Valor obtenido)
    saldoActual = cuenta.obtenerSaldo();

    Assert.AreEqual(saldoEsperado, saldoActual, 0.001, "El saldo de la cuenta no es correcto");
}
```

Caso 3: Que el saldo Inicial sea negativo = false

Caso 4: Que el reintegro sea negativo = La cantidad indicada es negativa

Caso 5: Que el reintegro sea 0 = false

Caso 6: Que el reintegro sea NULL = false

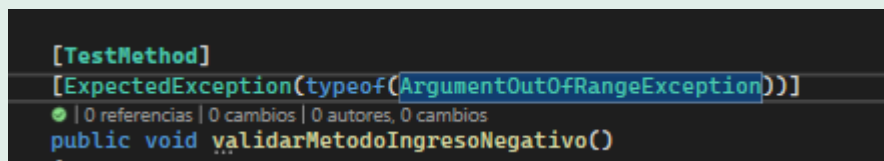
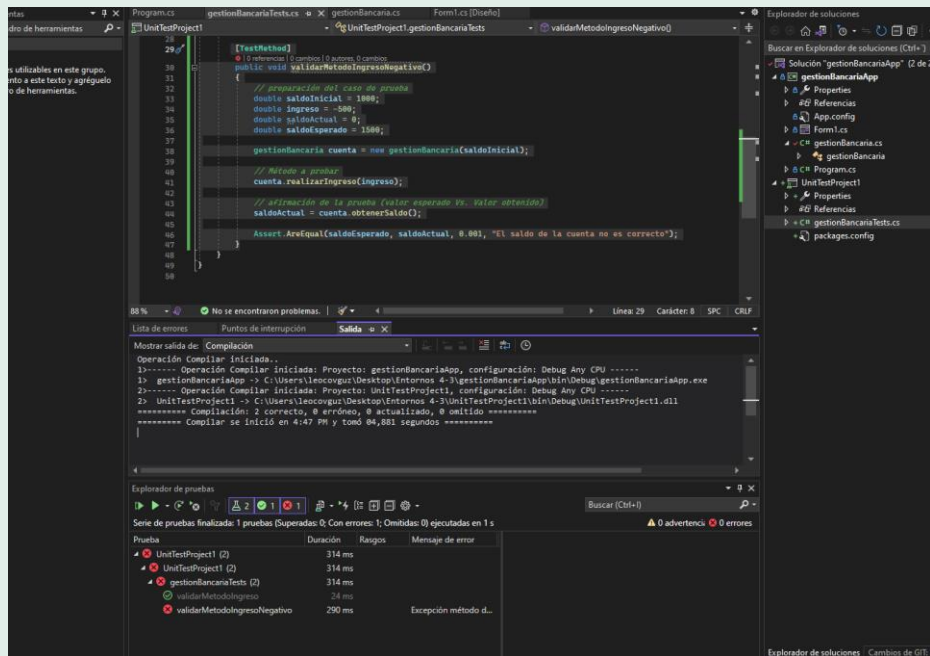
Valores Limite en el Ingreso es que el valor sea un numero positivo y en el Reintegro que el valor sea menor que el saldo Inicial que hay dentro de la cuenta bancaria

Punto 8 Pruebas que esperan excepciones

```
if (cantidad <= 0)
{
    //mostrarError(ERR_CANTIDAD_INDICADA_NEGATIVA);
    throw new ArgumentOutOfRangeException("La cantidad indicada es negativa ");
}
else
{
    if (cantidad > 0 && saldo > cantidad)
    {
        saldo -= cantidad;
    }
}
```

Excepciones cambiadas correctamente por
ArgumentOutOfRangeException

```
if (cantidad <= 0)
{
    //mostrarError(ERR_CANTIDAD_INDICADA_NEGATIVA);
    throw new ArgumentOutOfRangeException("La cantidad indicada es negativa ");
}
else
{
    if (cantidad > 0 && saldo > cantidad)
    {
        saldo -= cantidad;
    }
    else
    {
        //mostrarError(ERR_SALDO_INSUFICIENTE);
        throw new ArgumentOutOfRangeException("Saldo Insuficiente");
    }
}
```

Colocamos esa excepción para que se pueda realizar el que sea negativo el ingreso

