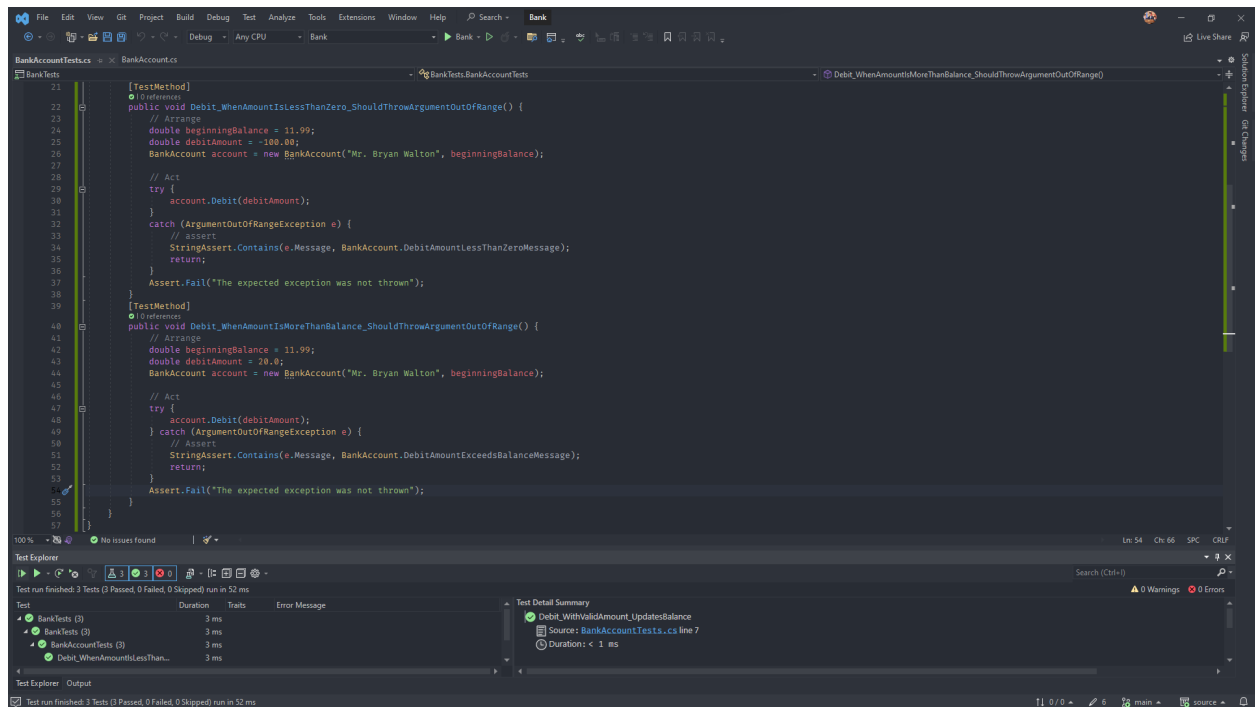


A) Basado en el tutorial del ejercicio de pruebas unitarias, en la última sección, Retest, rewrite, and reanalyze, explique muy brevemente ¿Por qué se necesita agregar la instrucción `Assert.Fail` a este caso de prueba?

Se necesita porque la idea es que esos **unit tests** entren en la expresión *catch* donde se revisa si se envió el mensaje correcto. Sucede que, si la función no lanza una excepción también pasaría el test porque terminaría sin ejecutar un *assert*. Por ello, agregamos un *assert* al final de la función y un *return* luego del *assert* dentro del *catch* para eliminar ese bug en el *unit test*.



```
[TestMethod]
public void Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRangeException() {
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = -100.00;
    BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);

    // Act
    try {
        account.Debit(debitAmount);
    }
    catch (ArgumentOutOfRangeException e) {
        // Assert
        StringAssert.Contains(e.Message, BankAccount.DebitAmountLessThanZeroMessage);
        return;
    }
    Assert.Fail("The expected exception was not thrown");
}

[TestMethod]
public void Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException() {
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = 20.0;
    BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);

    // Act
    try {
        account.Debit(debitAmount);
    }
    catch (ArgumentOutOfRangeException e) {
        // Assert
        StringAssert.Contains(e.Message, BankAccount.DebitAmountExceedsBalanceMessage);
        return;
    }
    Assert.Fail("The expected exception was not thrown");
}
```

B) Basado en la lectura de conceptos sobre pruebas unitarias, responda muy brevemente ¿cuál es el valor de las pruebas unitarias en el flujo de desarrollo de software?

El valor es que permiten evaluar en todo momento el código, revisando que las “unidades” tengan el comportamiento esperado y en caso contrario repararlo. También, sirve como un mecanismo de diseño pues se pueden establecer los test de manera previa al código, así la interfaz se impone desde el inicio y se sabe que debe dar la unidad de código que estamos produciendo. En general también produce código seguro y funcional.

**C) Basado en la lectura de conceptos sobre pruebas unitarias, responda muy brevemente ¿cuáles son las principales partes que componen una prueba unitaria?**

Las partes son AAA, **arrange**, **act**, **assert**. En **arrange** preparamos todo lo necesario para poder hacer nuestra prueba. Luego, en el **act** ejecutamos la acción sobre el escenario preparado en **arrange** y finalmente hacemos **assert**, donde revisamos que el resultado sea el correcto.

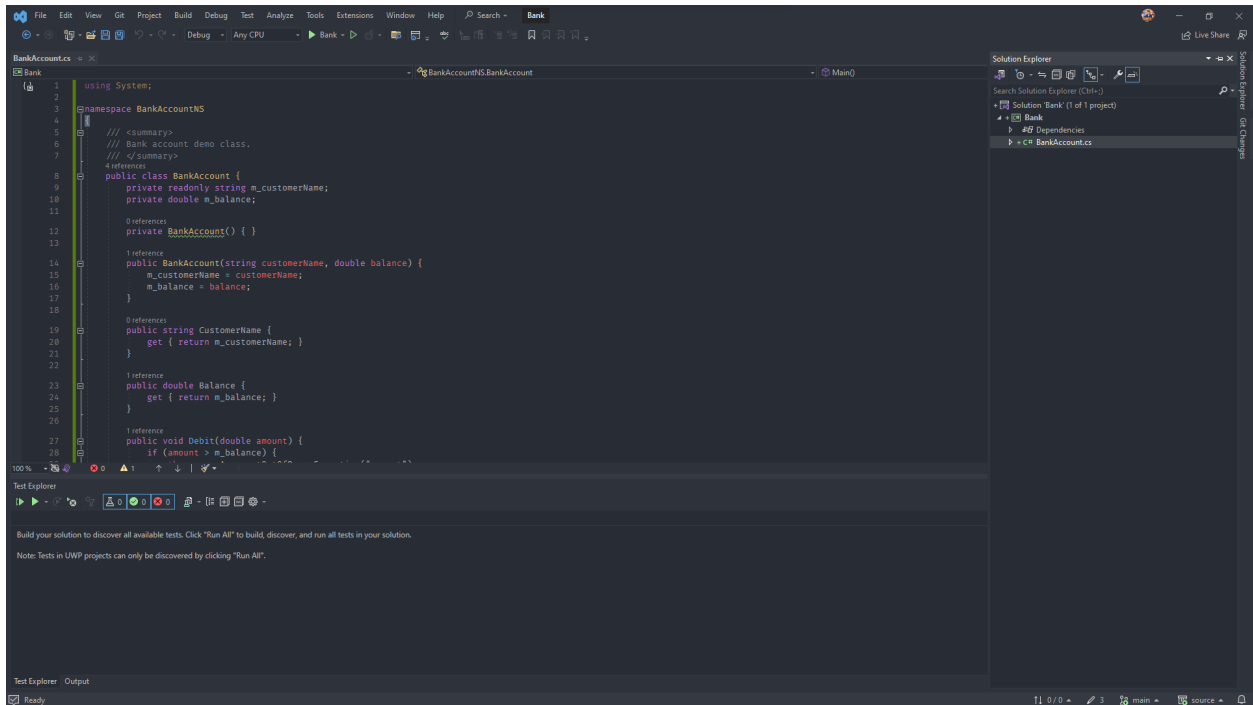
**D) Basado en la lectura de conceptos sobre pruebas unitarias, responda muy brevemente ¿Para qué sirve establecer un timeout a un caso de prueba?**

Sirve para dos cosas, para evaluar el desempeño de una unidad de código con un *test*, dicho en otras palabras para evaluar la complejidad de tiempo de una función por ejemplo. También en el caso de que la función falle por reciclarse y no termine, el *timer* es ideal para detectar este problema.

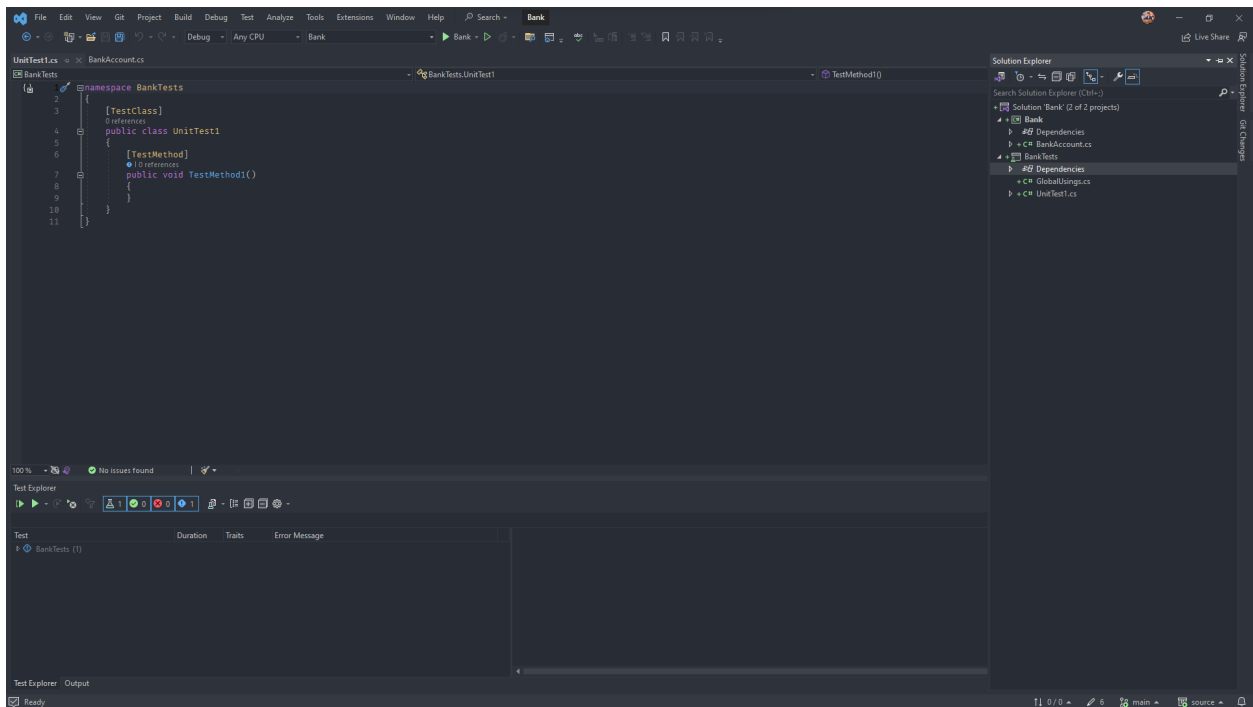
**Capturas del proceso del tutorial**

## A. Badilla Olivas - B80874 - lab 3 - Ingeniería de Software - ciclo II 2023 g1

Crear el proyecto Bank.

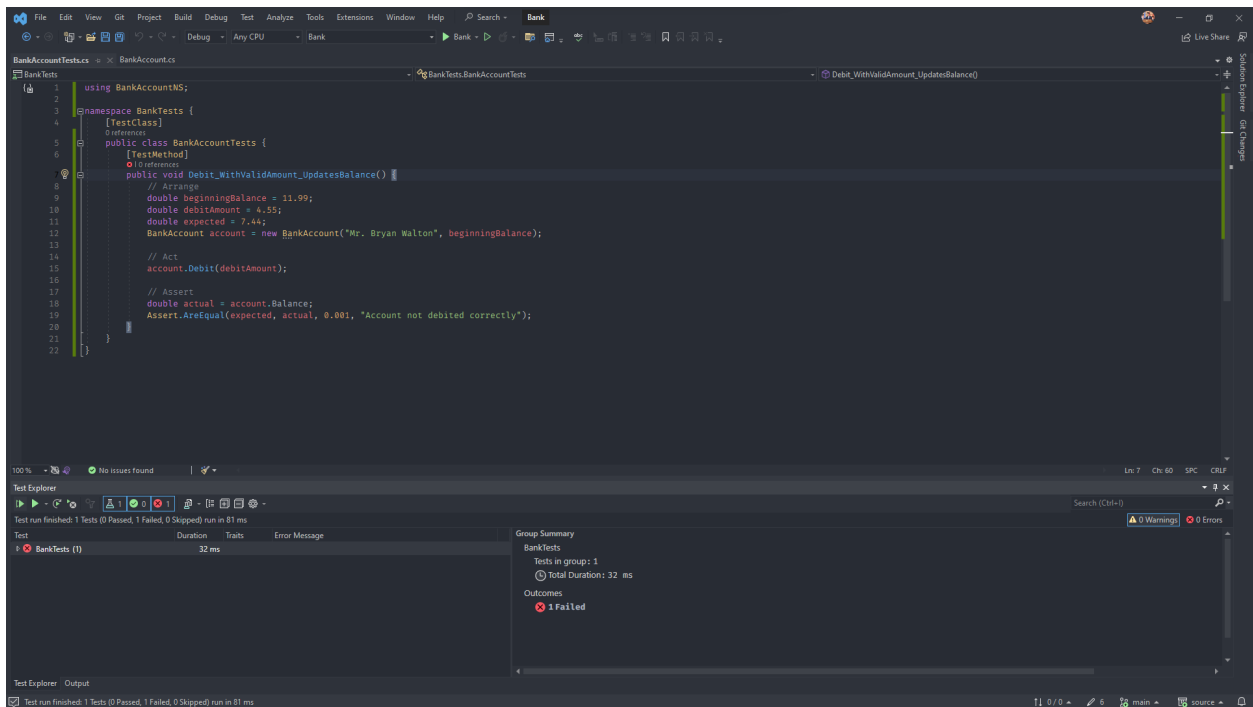


Agregar un proyecto de pruebas a la solución.

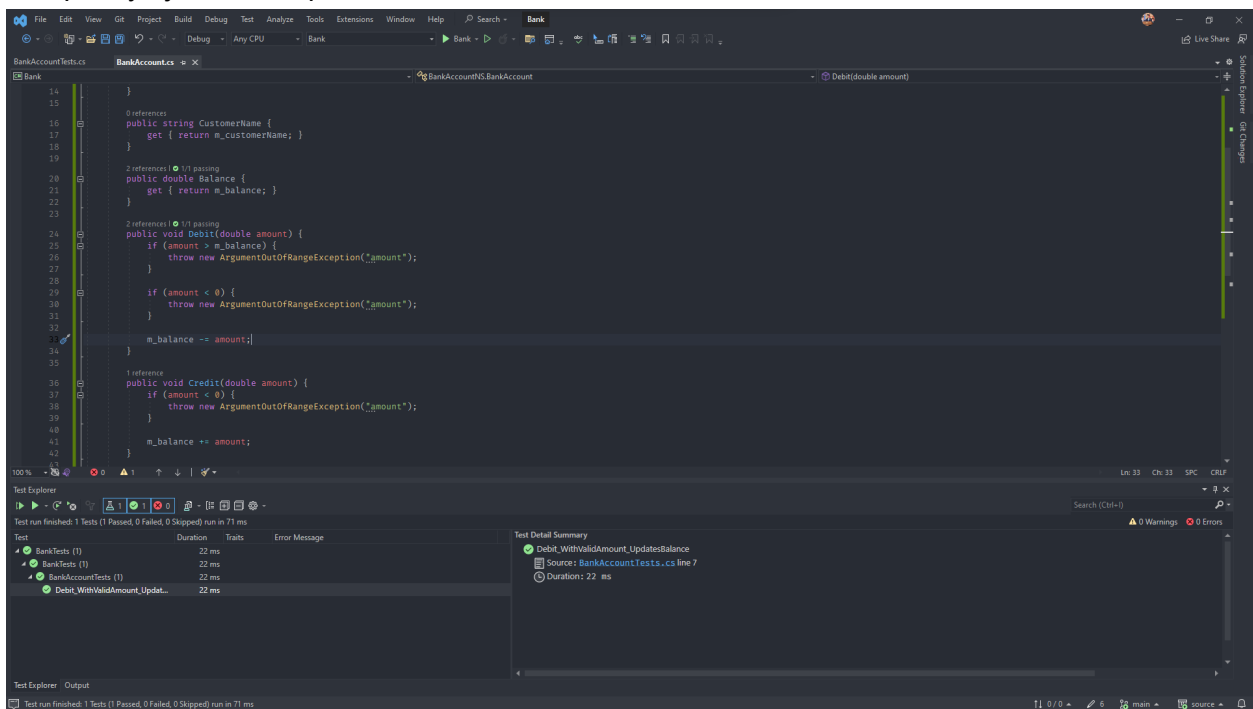


## A. Badilla Olivas - B80874 - lab 3 - Ingeniería de Software - ciclo II 2023 g1

Añadir un unit test para la función debit.

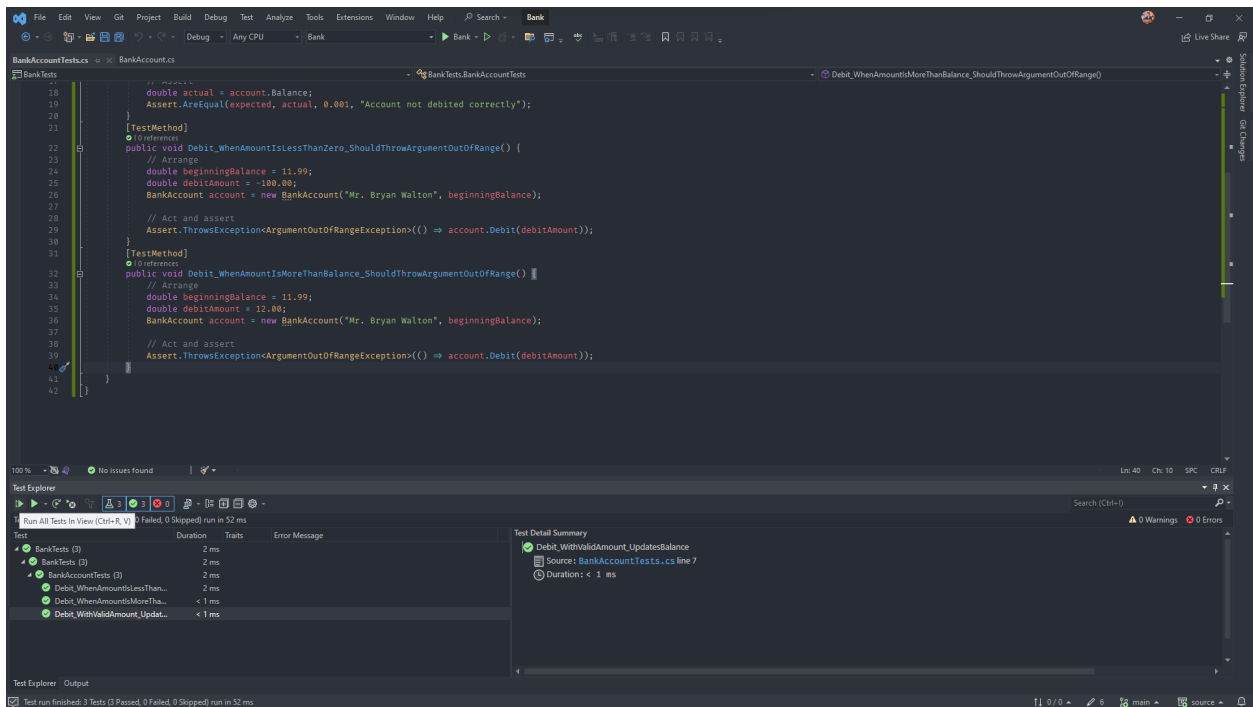


Compilar y ejecutar las pruebas.

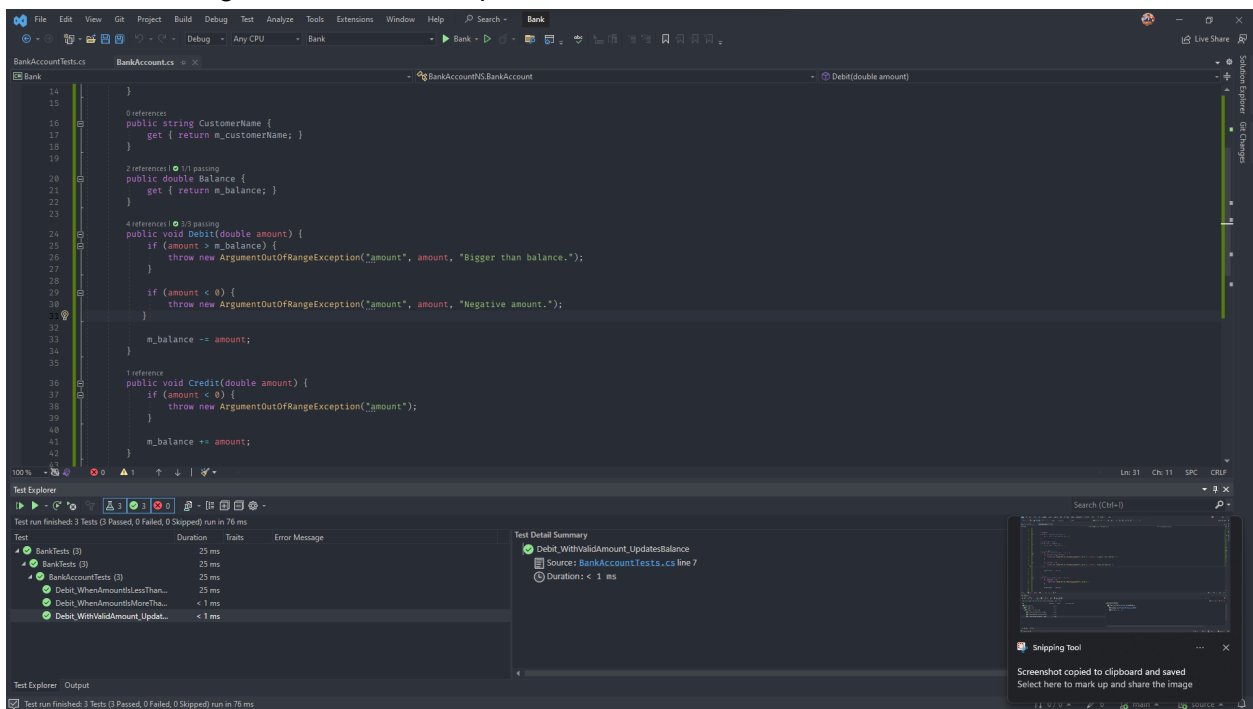


## A. Badilla Olivas - B80874 - lab 3 - Ingeniería de Software - ciclo II 2023 g1

Agregar más unit tests para dos posibles comportamientos esperados.



Haciendo más significativas las excepciones.



## Refactorizando excepciones.

