```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace BankAccountTest
{
    [TestClass]
    public class BankTest
    {
        [TestMethod]
        public void Debit_WithValideAmount_UpdateBalance()
        {
            //Arrange ( definie les varaible de depart et attendu )
            double beginningBalance = 11.99;
            double debitAmount = 4.55;
            double expected = 7.44;
            BankAccount account = new BankAccount("Mr Fabien GAUDRON", beginningBalance);

            //Act ( on fait l'action a tester )
            account.Debit(debitAmount);

            //Assert ( on compare les resultats obtenu et attendu )
            double actual = account.Balance;                                        // Récuperation de la valeur actuel
            Assert.AreEqual(expected, actual, " Account not debited correctly");    // Conparaison avec la valeur attendu
        }

        [TestMethod]
        public void Credit_WithValideAmount_UpdateBalance()
        {
            //Arrange ( definie les varaible de depart et attendu )
            double beginningBalance = 11.99;
            double creditAmount = 4.55;
            double expected = 16.54;
            BankAccount account = new BankAccount("Mr Fabien GAUDRON", beginningBalance);

            //Act ( on fait l'action a tester )
            account.Credit(creditAmount);

            //Assert ( on compare les resultats obtenu et attendu )
            double actual = account.Balance;                                        // Récuperation de la valeur actuel
            Assert.AreEqual(expected, actual, " Account not credited correctly");    // Conparaison avec la valeur attendu
        }
```

```
        [TestMethod]
        [ExpectedException(typeof(ArgumentOutOfRangeException))]        // On defini le type d'erreur attendu
        public void Debit_WhenAmountisLessThanZero_ShouldThrowArgumentOfRange()
        {
            //Arrange
            var account = new BankAccount("Mr Fabien GAUDRON", 50);
            var debitAmount = -30;

            //Act
            account.Debit(debitAmount);

            //Assert
            // pas besoin car on attend un type d'erreur ArgumentOfRange
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentOutOfRangeException))]        // On defini le type d'erreur attendu
        public void Debit_WhenAmountisGreaterThanBalance_ShouldThrowArgumentOfRange()
        {
            //Arrange
            var account = new BankAccount("Mr Fabien GAUDRON", 50);
            var debitAmount = 100;

            //Act
            account.Debit(debitAmount);

            //Assert
            // pas besoin car on attend un type d'erreur ArgumentOfRange
        }

        [TestMethod]
        [ExpectedException(typeof(Exception))]        // On defini le type d'erreur attendu
        public void Debit_WhenAccountisFrozen_ShouldThrowException()
        {
            //Arrange
            var account = new BankAccount("Mr Fabien GAUDRON", 50);
            var privateObj = new PrivateObject(account);        //Permet de recuperer une varaible private dans notre prgm
            privateObj.Invoke("FreezeAccount");                 // permet de faire appelle à une methode private dans notre
prgm

            //Act
            account.Debit(10);

            //Assert
            // pas besoin car on attend un type d'erreur ArgumentOfRange
        }
```

```
        [TestMethod]
        [ExpectedException(typeof(Exception))]        // On defini le type d'erreur attendu
        public void Credit_WhenAccountisFrozen_ShouldThrowException()
        {
            //Arrange
            var account = new BankAccount("Mr Fabien GAUDRON", 50);
            var privateObj = new PrivateObject(account);        //Permet de recuperer une varaible private dans notre prgm
            privateObj.Invoke("FreezeAccount");                 // permet de faire appelle à une methode private dans notre
prgm

            //Act
            account.Credit(10);

            //Assert
            // pas besoin car on attend un type d'erreur ArgumentOfRange
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentOutOfRangeException))]        // On defini le type d'erreur attendu
        public void Credit_WhenAmountisLessThanZero_ShouldThrowArgumentOfRange()
        {
            //Arrange
            var account = new BankAccount("Mr Fabien GAUDRON", 50);
            var creditAmount = -30;

            //Act
            account.Credit(creditAmount);

            //Assert
            // pas besoin car on attend un type d'erreur ArgumentOfRange
        }


        [TestMethod]
        public void CustomerNameTest()
        {
            //Arrange
            var account = new BankAccount("Mr Fabien GAUDRON", 50);
            var actual = account.CustomerName;
            var expected = "Mr Fabien GAUDRON";

            //Assert
            Assert.AreEqual(expected, actual, "CustomerName initialization is incorrect");
        }
```

```
[TestMethod]
public void FreezeAccountTest()
{
    //Arrange
    var account = new BankAccount("Mr Fabien GAUDRON", 50);
    var privateObj = new PrivateObject(account);
    privateObj.Invoke("FreezeAccount");
    var actual = (bool)privateObj.GetField("m_frozen");
    var expected = true;

    //Assert
    Assert.AreEqual(expected, actual);

}

[TestMethod]
public void unFreezeAccountTest()
{
    //Arrange
    var account = new BankAccount("Mr Fabien GAUDRON", 50);
    var privateObj = new PrivateObject(account);
    privateObj.Invoke("UnFreezeAccount");
    var actual = (bool)privateObj.GetField("m_frozen");
    var expected = false;

    //Assert
    Assert.AreEqual(expected, actual);

}


[TestMethod]
public void PublicConstructorTest()
{
    //Arrange
    var account = new BankAccount("Mr Fabien GAUDRON", 50);

    var actualName = account.CustomerName;
    var actualBalance = account.Balance;

    var expectedName = "Mr Fabien GAUDRON";
    var expectedBalance = 50;

    //Assert
    Assert.AreEqual(expectedName, actualName);
    Assert.AreEqual(expectedBalance, actualBalance);

}  } }
```