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

namespace BankAccountTestOptimized
{
    [TestClass]
    public class BankTestOptimized
    {
        BankAccount account;

        [TestInitialize]                                                // Permet de créer les objets tester par la suite
        public void SetUp()
        {
            account = new BankAccount("Mr Fabien GAUDRON", 50);
        }

        [TestCleanup]                                                   // On nettoie les objets utiliser par les tests
        public void CleanUp()
        {
            account = null;
        }

        [TestMethod]
        public void Debit_WithValideAmount_UpdateBalance()
        {
            account.Debit(10);
            double actual = account.Balance;                                        // Récuperation de la valeur actuel
            double expected = 40;
            Assert.AreEqual(expected, actual, " Account not debited correctly");    // Conparaison avec la valeur attendu
        }

        [TestMethod]
        public void Credit_WithValideAmount_UpdateBalance()
        {
            account.Credit(4.55);
            double actual = account.Balance;                                         // Récuperation de la valeur actuel
            double expected = 54.55;
            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()
        {
            var debitAmount = -30;
            account.Debit(debitAmount);        }
```

```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]        // On defini le type d'erreur attendu
public void Debit_WhenAmountisGreaterThanBalance_ShouldThrowArgumentOfRange()
{
    var debitAmount = 100;
    account.Debit(debitAmount);
}

[TestMethod]
[ExpectedException(typeof(Exception))]        // On defini le type d'erreur attendu
public void Debit_WhenAccountisFrozen_ShouldThrowException()
{
    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
    account.Debit(10);
}

[TestMethod]
[ExpectedException(typeof(Exception))]        // On defini le type d'erreur attendu
public void Credit_WhenAccountisFrozen_ShouldThrowException()
{
    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
    account.Credit(10);
}

[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]        // On defini le type d'erreur attendu
public void Credit_WhenAmountisLessThanZero_ShouldThrowArgumentOfRange()
{
    var creditAmount = -30;
    account.Credit(creditAmount);
}


[TestMethod]
public void CustomerNameTest()
{
    var actual = account.CustomerName;
    var expected = "Mr Fabien GAUDRON";
    Assert.AreEqual(expected, actual, "CustomerName initialization is incorrect");
}
```

```
    [TestMethod]
    public void FreezeAccountTest()
    {
        var privateObj = new PrivateObject(account);
        privateObj.Invoke("FreezeAccount");
        var actual = (bool)privateObj.GetField("m_frozen");
        //var expected = true;
        //Assert.AreEqual(expected, actual);
        Assert.IsTrue(actual);                          // Equivalent au deux lignes du haut
    }

    [TestMethod]
    public void unFreezeAccountTest()
    {
        var privateObj = new PrivateObject(account);
        privateObj.Invoke("UnFreezeAccount");
        var actual = (bool)privateObj.GetField("m_frozen");
        //var expected = false;
        //Assert.AreEqual(expected, actual);
        Assert.IsFalse(actual);                         // Equivalent au deux lignes du haut
    }


    [TestMethod]
    public void PublicConstructorTest()
    {
        var actualName = account.CustomerName;
        var actualBalance = account.Balance;
        var expectedName = "Mr Fabien GAUDRON";
        var expectedBalance = 50;
        Assert.AreEqual(expectedName, actualName);
        Assert.AreEqual(expectedBalance, actualBalance);
    }
}}
```