* 1. Пoправете класа, така че може да се задава само положителен баланс
  2. Напишете компонентни тестове, които да проверят работата на класа и да помогнат в откриването и отстраняването на евентуални проблеми. Тестовете да покриват всички методи на класа, включително и конструктора. За тестовете използвайте и **[SetUp]** метод.

1. В свойсвото се прави верификацията, затова там се добавя проверката.
2. public double Balance
3. {
4. get { return balance; }
5. set {
6. if (value <= 0)
7. {
8. throw new Exception("Negativ balance");
9. }
10. balance = value;
11. }
12. }
13. В сълюшъна добавете тестови проект, **с подходящо име**, като използвате шаблона **Nunit Test Project(.Net Core):**
14. Преименувайте **подходящо** си тестовия клас.
15. Добавете на референция към проекта, който ще се тества:
16. В тестовия проект добавете using директивата:

using Bank; //да се добави

1. Визуализирайте си **Test Explorer** от меню **Test**, за да можете да си стартирате тестовете.
2. Тестови клас – коментарите са за ваше удобство и не са задължителни

public class BankAccountTest

{

private double balance;

private double amount;

[SetUp]

public void TestInit()

{

balance = 100;

amount = 35;

}

[Test]

public void AcountInitializeWithPositiveValue()

{ //Arrange, Act

BankAccount acount = new BankAccount(balance);

//Assert

Assert.AreEqual(balance, acount.Balance);

}

[Test]

public void AcountInitializeWithNegativeValue()

{ //Не се тества така т.к. имаме изключение генерирано от метода

// BankAcount acount = new BankAcount(-2000);

// Assert.AreEqual(-2000, acount.Balance);

//Arrange, Act, Assert

Assert.Throws<System.Exception>(() => new BankAccount(-balance));

}

[Test]

public void AcountWithdrawAmountLessOrEqualBalans()

{

//Arrange

BankAccount acount = new BankAccount(balance);

// Act

acount.Withdraw(amount);

//Assert

Assert.AreEqual(acount.Balance, balance - amount);

}

[Test]

public void AcountWithdrawAmountMoreThenBalans()

{

//Arrange

BankAccount acount = new BankAccount(balance);

// Act и Assert - тестване на изключение

Assert.Throws<System.ArgumentException>(() => acount.Withdraw(balance + amount));

}

[Test]

public void AcountDepositAmountPositive()

{

//Arrange

BankAccount acount = new BankAccount(balance);

// Act

acount.Deposit(amount);

//Assert

Assert.AreEqual(acount.Balance, balance + amount);

}

[Test]

public void AcountDepositAmountNegative()

{

//Arrange

BankAccount acount = new BankAccount(balance);

amount = -135;

// Act и Assert

Assert.Throws<System.ArgumentException>(() => acount.Deposit(amount));

}

[Test]

public void AcountDepositPositivAmount()

{ //Arrange

BankAccount acount = new BankAccount(balance);

// Act

acount.Deposit(amount);

//Assert

Assert.AreEqual(acount.Balance, balance + amount);

}

[Test]

public void AcountDepositNegativeAmount()

{

//Arrange

amount = -135;

BankAccount acount = new BankAccount(balance);

// Act и Assert

Assert.Throws<System.ArgumentException>(() => acount.Deposit(amount));

}

}