**Những điều cần lưu ý:**

1. UnitTest là OOP
2. Chúng ta lập trình những function trước > chuột phải vào các hàm > chọn create unit test > VS tự động tạo các unit test > những unit test sẽ có ngoằn đỏ gạch dưới > để giải quyết, chúng ta phải build project > chuột phải vào project > chọn build (hoặc build solution cũng phục vụ cùng chức năng)
3. Để test các function thì trước những function chúng ta phải có [TestMethod]
4. [DataTestMethod] và [DataRow] được dùng để unit test multiple values

using Microsoft.VisualStudio.TestTools.UnitTesting;

using BankAcct;

using System;

Với TestMethod bình thường thì chúng ta chỉ nhập vào 1 dòng để kiểm tra function đó

DataTestMethod sẽ có nhiều dòng để chúng ta test liên tiếp nhiều dòng với những testcase khác nhau

ExpectedException được dùng để test exception của function, có những function sẽ throw Exception và ExpectedException sẽ được dùng để test những Exception đó mà không báo lỗi

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankAcct.Tests

{

[TestClass()]

public class BankAccountTests

{

[TestMethod()]

public void CreditTest()

{

//Assert.Fail();

}

[DataTestMethod]

[DataRow(100, -10, 110)]

[DataRow(100, -2000, 0)]

[ExpectedException(typeof(ArgumentOutOfRangeException))]

public void CreditTest\_Exception(double expected, double amount, double balance)

{

BankAccount ba = new BankAccount("Tester 1", balance);

ba.Credit(amount);

double act = ba.Balance;

Assert.AreEqual(expected, act);

}

[DataTestMethod]

[DataRow(50, 50, 0)]

[DataRow(100, 10, 90)]

public void CreditTest\_MV(double expected, double amount, double balance)

{

BankAccount ba = new BankAccount("Tester 1", balance);

ba.Credit(amount);

double act = ba.Balance;

Assert.AreEqual(expected, act);

}

[TestMethod()]

public void DebitTest()

{

}

[DataTestMethod]

[DataRow(100, 1000, 110)]

[DataRow(100, -2000, 0)]

[ExpectedException(typeof(ArgumentOutOfRangeException))]

public void DeditTest\_Exception(double expected, double amount, double balance)

{

BankAccount ba = new BankAccount("Tester 1", balance);

ba.Debit(amount);

double act = ba.Balance;

Assert.AreEqual(expected, act);

}

[DataTestMethod]

[DataRow(10, 90, 100)]

[DataRow(0, 50, 50)]

[DataRow(100, 100, 200)]

public void DebitTest\_MV(double expected, double amount, double balance)

{

BankAccount ba = new BankAccount("Tester 1", balance);

ba.Debit(amount);

Những DataTestMethod sẽ có nhiều dòng (chú ý những chỗ có MV – multiple values)

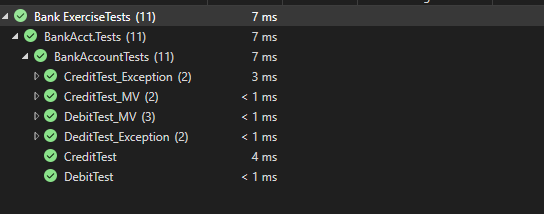
double act = ba.Balance;

Assert.AreEqual(expected, act);

}

}

}



using Microsoft.VisualStudio.TestTools.UnitTesting;

using UnitTest;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics.SymbolStore;

namespace UnitTest.Tests

{

[TestClass()]

public class StudentManagerTests

{

[TestMethod()]

public void addStudentTest()

{

StudentManager manager = new StudentManager();

bool result = manager.addStudent(new Student("HS01", "Le Gia Phu", 12, 8.8));

Assert.AreEqual(true, result);

}

[DataTestMethod]

[DataRow(true, "HS01", "Le Gia Phu", 12, 8.8)]

[DataRow(true, "HS02", "Le Gia Phu", 12, 8.8)]

[DataRow(true, "HS03", "Le Gia Phu", 12, 8.8)]

[DataRow(true, "HS01", "Le Gia Phu", 12, 8.8)]

public void addStudentTest\_MV(bool expected, String id, String name, int age, double score)

{

Student std = new Student(id, name, age, score);

StudentManager manager = new StudentManager();

bool act = manager.addStudent(std);

Assert.AreEqual(expected, act);

}

[TestMethod()]

[ExpectedException(typeof(ArgumentOutOfRangeException))]

public void getStudentAtTest()

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS01", "Le Gia Phu", 12, 8.8));

manager.addStudent(new Student("HS02", "Le Gia Phu", 12, 8.8));

manager.addStudent(new Student("HS03", "Le Gia Phu", 12, 8.8));

Student std = manager.getStudentAt(10);

}

[DataTestMethod]

[DataRow(true, 2)]

public void getStudentAtTest\_MV(bool expected, int position)

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS01", "Le Gia Phu", 12, 8.8));

manager.addStudent(new Student("HS02", "Le Gia Phu", 12, 8.8));

manager.addStudent(new Student("HS03", "Le Gia Phu", 12, 8.8));

Student std = manager.getStudentAt(position);

bool act = false;

if (std.ID.Length != 0)

{

act = true;

}

Assert.AreEqual(expected, act);

}

[TestMethod()]

public void findStudentByAgeTest()

{

//Assert.Fail();

}

[DataTestMethod]

[DataRow(true, 10)]

[DataRow(true, 11)]

[DataRow(true, 12)]

[DataRow(false, 18)]

public void findStudentByAgeTest\_MV(Object expected, int age)

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS01", "Le Gia Phu", 10, 8.8));

manager.addStudent(new Student("HS02", "Le Gia Phu", 11, 8.8));

manager.addStudent(new Student("HS03", "Le Gia Phu", 12, 8.8));

bool act = false;

Student sv = manager.findStudentByAge(age);

if (sv != null)

{

act = true;

}

Assert.AreEqual(expected, act);

}

[TestMethod()]

[ExpectedException(typeof(Exception))]

public void getAverageScoreTest()

{

StudentManager manager = new StudentManager();

double act = manager.getAverageScore();

}

[TestMethod]

public void getAverageScoreTest\_2()

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS01", "Le Gia Phu", 10, 8));

manager.addStudent(new Student("HS02", "Le Gia Phu", 11, 7));

manager.addStudent(new Student("HS03", "Le Gia Phu", 12, 9));

double expected = 8.0;

double act = manager.getAverageScore();

Assert.AreEqual(expected, act);

}

[TestMethod()]

public void findStudentWithMinScoreTestEmptyList()

{

StudentManager manager = new StudentManager();

bool expected = false;

bool act = false;

Student std = manager.findStudentWithMinScore();

if (std != null)

{

act = true;

}

Assert.AreEqual(expected, act);

//Assert.Fail();

}

[DataTestMethod]

[DataRow("HS05")]

public void findStudentWithMinScoreTest\_MV(String expected)

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS01", "Le Gia Phu", 10, 10));

manager.addStudent(new Student("HS02", "Le Gia Phu", 11, 9));

manager.addStudent(new Student("HS03", "Le Gia Phu", 12, 8));

manager.addStudent(new Student("HS04", "Le Gia Phu", 10, 7));

manager.addStudent(new Student("HS05", "Le Gia Phu", 11, 6));

manager.addStudent(new Student("HS06", "Le Gia Phu", 12, 6));

String act = manager.findStudentWithMinScore().ID.ToString().Trim();

Assert.AreEqual(expected, act);

//Assert.Fail();

}

[TestMethod()]

public void sortByAgeTest()

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS01", "Le Gia Phu", 10, 10));

manager.addStudent(new Student("HS02", "Le Gia Phu", 12, 9));

manager.addStudent(new Student("HS03", "Le Gia Phu", 13, 8));

manager.addStudent(new Student("HS04", "Le Gia Phu", 14, 7));

manager.addStudent(new Student("HS05", "Le Gia Phu", 15, 6));

manager.addStudent(new Student("HS06", "Le Gia Phu", 16, 6));

manager.sortByAge();

StudentManager manager2 = new StudentManager();

manager2.addStudent(new Student("HS06", "Le Gia Phu", 16, 6));

manager2.addStudent(new Student("HS05", "Le Gia Phu", 15, 6));

manager2.addStudent(new Student("HS04", "Le Gia Phu", 14, 7));

manager2.addStudent(new Student("HS03", "Le Gia Phu", 13, 8));

manager2.addStudent(new Student("HS02", "Le Gia Phu", 12, 9));

manager2.addStudent(new Student("HS01", "Le Gia Phu", 10, 10));

//should be true

bool expected = true;

bool act = true;

for (int i = 0; i < (int)manager.getStudentManagerLength(); i++)

{

String value1 = manager.getStudentAt(i).ID.ToString().Trim();

String value2 = manager2.getStudentAt(i).ID.ToString().Trim();

if (!value1.Equals(value2))

{

act = false;

break;

}

}

Assert.AreEqual(expected, act);

}

[TestMethod()]

public void sortByAgeTest2()

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS01", "Le Gia Phu", 10, 10));

manager.addStudent(new Student("HS02", "Le Gia Phu", 12, 9));

manager.addStudent(new Student("HS03", "Le Gia Phu", 13, 8));

manager.addStudent(new Student("HS04", "Le Gia Phu", 14, 7));

manager.addStudent(new Student("HS05", "Le Gia Phu", 15, 6));

manager.addStudent(new Student("HS06", "Le Gia Phu", 16, 6));

manager.sortByAge();

StudentManager manager2 = new StudentManager();

manager2.addStudent(new Student("HS05", "Le Gia Phu", 15, 6));

manager2.addStudent(new Student("HS04", "Le Gia Phu", 14, 7));

manager2.addStudent(new Student("HS03", "Le Gia Phu", 13, 8));

manager2.addStudent(new Student("HS02", "Le Gia Phu", 12, 9));

manager2.addStudent(new Student("HS01", "Le Gia Phu", 10, 10));

manager2.addStudent(new Student("HS06", "Le Gia Phu", 16, 6));

//should be false

bool expected = false;

bool act = true;

for (int i = 0; i < (int)manager.getStudentManagerLength(); i++)

{

String value1 = manager.getStudentAt(i).ID.ToString().Trim();

String value2 = manager2.getStudentAt(i).ID.ToString().Trim();

if (!value1.Equals(value2))

{

act = false;

break;

}

}

Assert.AreEqual(expected, act);

}

[TestMethod()]

public void filterByScoreTest()

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS04", "Le Gia Phu", 14, 7));

manager.addStudent(new Student("HS02", "Le Gia Phu", 12, 9));

manager.addStudent(new Student("HS01", "Le Gia Phu", 10, 10));

manager.addStudent(new Student("HS06", "Le Gia Phu", 16, 6));

manager.addStudent(new Student("HS03", "Le Gia Phu", 13, 8));

manager.addStudent(new Student("HS05", "Le Gia Phu", 15, 6));

bool expected = true;

bool act = false;

if (manager.filterByScore() != null)

{

act = true;

}

Assert.AreEqual(expected, act);

}

[TestMethod()]

public void filterByScoreTest2()

{

StudentManager manager = new StudentManager();

manager.addStudent(new Student("HS04", "Le Gia Phu", 14, 7));

manager.addStudent(new Student("HS06", "Le Gia Phu", 16, 6));

manager.addStudent(new Student("HS03", "Le Gia Phu", 13, 8));

manager.addStudent(new Student("HS05", "Le Gia Phu", 15, 6));

bool expected = false;

bool act = false;

if (manager.filterByScore() != null)

{

act = true;

}

Assert.AreEqual(expected, act);

}

}

}

