|  |
| --- |
|  |
| C# Types & Assemblies |
| Putting C# to the Test |
|  |
|  |
|  |

|  |
| --- |
|  |

# 

# C# Types and Assemblies

## Objectives

In this lab you’ll write unit tests and discover the behavior of different categories of objects. By the end you should:

* Understand the different between reference types and value types
* Pass parameters by value and reference
* Work with arrays

## Part 1 – Value Types

1. Open Visual Studio and select File -> **New Project**
2. Select **Test Project** from the Visual C# -> Test node. Name the project **TypeTests**. Place the project into the **before** directory for this lab.
3. Delete the UnitTest1.cs file from the new project.
4. Select File -> **Add New Project**
5. Select **Class Library** from the Visual C# -> Windows node. Name the project **TypeLogic**.
6. **Delete** the **Class1.cs** file from the TypeLogic project.
7. Right-click the **TypeTests** project and select **Add** Reference.
8. On the Projects tab, select the **TypeLogic** project and press OK.

At this point we have the project structure setup. The test project references the logic project and will be able to use public types defined in the logic project.

1. Add a new class to the **TypeLogic** project named **IntChanger**.
2. Mark the new class as public.
3. Add a Change method to the class as shown below.

public class IntChanger  
{  
    public void Change(int x)  
    {  
        x += 1;  
    }          
}

1. Right click the **TypeTests** project and select Add -> **New Test**. Select **Basic Unit Test** from the dialog box and name the test **ValueTypeTests**.
2. Add a using statement for TypeLogic;

using TypeLogic;

1. Change the name of TestMethod1 to **CanChangeIntParameter** and add the code below.

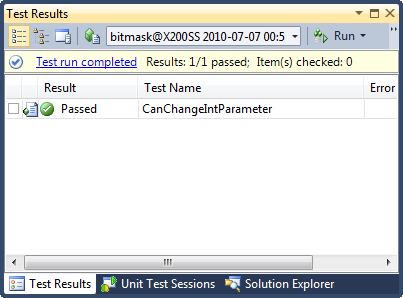
[TestMethod]  
public void CanChangeIntParameter()  
{  
    int x = 10;  
    IntChanger changer = new IntChanger();  
    changer.Change(x);  
  
    // Assert.IsTrue(x == 10);  
    // Assert.IsTrue(x == 11);  
}

1. **Uncomment** **one** of the Assert statements.

Which one do you think will be true?

1. Run the test (Test -> Run -> All Tests in Solution).

Did you make the right choice? Remember - parameters pass by *value* in C# and integer is a *value* type. If you are unsure of the result you are seeing then ask the instructor for clarification.



1. Return to the IntChanger class and add the following method.

public void Change(ref int x)  
{  
    x += 1;  
}

In this method we are passing the int *by ref.* The is an example of method overloading in C#. Two methods with the same name, but the signatures differ because C# distinguishes between pass by value and pass by ref.

1. Return to TypeTests and add the following test.

[TestMethod]  
public void CanChangeRefIntParameter()  
{  
    int x = 10;  
    IntChanger changer = new IntChanger();  
    changer.Change(ref x);  
  
    // Assert.IsTrue(x == 10);  
    // Assert.IsTrue(x == 11);  
}

1. **Uncomment** **one** of the Assert statements and run all test to check your answer.

## Part II – Reference Types

1. Right-click the TypeLogic project and select Add -> Class. Name the class ArrayChanger.
2. Make the class public and add the following method.

public void ChangeValues(int[] values)  
{  
    for(int i = 0; i < values.Length; i++)  
    {  
        values[i] += 1;  
    }  
}

1. Right click the **TypeTests** project and select Add -> **New Test**. Select **Basic Unit Test** from the dialog box and name the test **ReferenceTypeTests**.
2. Add a using statement for TypeLogic;

using TypeLogic;

1. Change the name of TestMethod1 to CanChangeArrayElement and add the code shown below.

[TestMethod]  
public void CanChangeArrayElement()  
{  
    int[] values = {1, 2, 3};  
    ArrayChanger changer = new ArrayChanger();  
    changer.ChangeValues(values);  
  
    // Assert.IsTrue(values[0] == 1);  
    // Assert.IsTrue(values[0] == 2);  
}

1. Uncomment one of the Assert statements and run all tests to verify your answer.
2. Return to the ArrayChanger class and add the following method.

public void ChangeValues(ref int[] values)  
{  
    for (int i = 0; i < values.Length; i++)  
    {  
        values[i] += 1;  
    }  
}

1. Now add the following test to ReferenceTypeTests.

[TestMethod]  
public void CanChangeRefArrayElement()  
{  
    int[] values = { 1, 2, 3 };  
    ArrayChanger changer = new ArrayChanger();  
    changer.ChangeValues(ref values);  
  
    // Assert.IsTrue(values[0] == 1);  
    // Assert.IsTrue(values[0] == 2);  
}

1. Uncomment one of the Assert statements and run all tests to verify your answer.
2. Add the following method to ArrayChanger.

public void ChangeArray(int[] values)  
{  
    values = new int[] {4, 5, 6};  
}

1. Now add the following test.

[TestMethod]  
public void CanChangeArray()  
{  
    int[] values = { 1, 2, 3 };  
    ArrayChanger changer = new ArrayChanger();  
    changer.ChangeArray(values);  
  
    // Assert.IsTrue(values[0] == 1);  
    // Assert.IsTrue(values[0] == 4);  
}

1. Pick your answer and run the tests.
2. Add the following method to the ArrayChanger class.

 public void ChangeArray(ref int[] values)  
 {  
     values = new int[] { 4, 5, 6 };  
 }

1. Now add the following test and choose your answer carefully.

[TestMethod]  
public void CanChangeRefArray()  
{  
    int[] values = { 1, 2, 3 };  
    ArrayChanger changer = new ArrayChanger();  
    changer.ChangeArray(ref values);  
  
    // Assert.IsTrue(values[0] == 1);  
    // Assert.IsTrue(values[0] == 4);  
}

## Conclusion

Congratulations! If you passed all the tests on the first try then you have a great grasp on the concept of references versus values.

For extra credit – change the parameters in the IntChanger methods to type object.

public void Change(object x)  
{  
    int value = (int)x;  
    value += 1;  
}      
  
public void Change(ref object x)  
{  
    int value = (int)x;  
    value += 1;  
}

You’ll need to fix some compiler errors and assess the impact of boxing on your existing tests…