

# Composition & Unit testing with MS test

*By Ira Zavushchak*

softserve

# Agenda

- ❖ Composition vs Inheritance?
- ❖ Examples
- ❖ Unit testing
- ❖ TDD
- ❖ Creating MS Test project
- ❖ Assert class
- ❖ Examples



# Composition vs Inheritance

- ❖ **Object Composition** is a way to combine simple objects or data types into more complex ones.
- ❖ **Inheritance** is a way to form new classes by using classes that have already been defined.
- ❖ **Composition** realizes "has a" relationship: A car "has an" engine, a person "has a" name, etc.
- ❖ **Inheritance** realizes "is a" relationship. A car "is a" vehicle, a student "is a" person, etc.

# Example 1. Point and Triangle

```
public struct Point
{
    public int X, Y;
    3 references
    public Point(int x, int y) {this.X = x; this.Y = y; }
    0 references
    public void Distance(Point p) { }
}
3 references
public class Triangle
{
    private Point p1, p2, p3;
    1 reference
    public Triangle(Point a, Point b, Point c)
    { p1 = a; p2 = b; p3 = c; }
    0 references
    public void Perimeter() { }
    0 references
    public void Square() { }
    0 references
    public void Print() { }
}
```

```
static void Main(string[] args)
{
    Point p1, p2, p3;
    p1 = new Point(1, 5);
    p2 = new Point(4, 1);
    p3 = new Point(1, 1);
    Triangle tr1 = new Triangle(p1, p2, p3);

    Console.ReadKey();
}
```

# Example 2. Slide, Presentation

SoftServe Confidential

```
public class Slide
{
    private string text = "";
    private bool isanimation = false;
    1 reference
    public bool IsAnimation { get { return isanimation; } }
    3 references
    public string Text { get { return text; } set { text = value; } }
}
```

2 references

```
public class Presentation
{
    private List<Slide> slides = new List<Slide>();
    1 reference
    public void Add(Slide s) { slides.Add(s); }
    0 references
    public bool Delete(Slide s) { return slides.Remove(s); }
    1 reference
    public void Show()
    {
        foreach (Slide s in slides)
            Console.WriteLine("Slide text:{0} animation {1}", s.Text, s.IsAnimation);
    }
    0 references
    public void Show(int number)
    { Console.WriteLine(slides[number].Text); }
}
```

```
static void Main(string[] args)
{
    Presentation p = new Presentation();
    Slide s = new Slide();
    s.Text = "FirstSlide";
    p.Add(s);
    p.Show();

    Console.ReadKey();
}
```

softserve

# Task-Homework 1

Create struct Point:

- fields x and y,
- method Distance() to calculate distance between points
- method ToString(), which return the Point in format "(x,y)"

Create class Triangle:

- fields vertex1, vertex2, vertex3 of type Point
- constructors
- methods Perimeter(), Square(), Print()

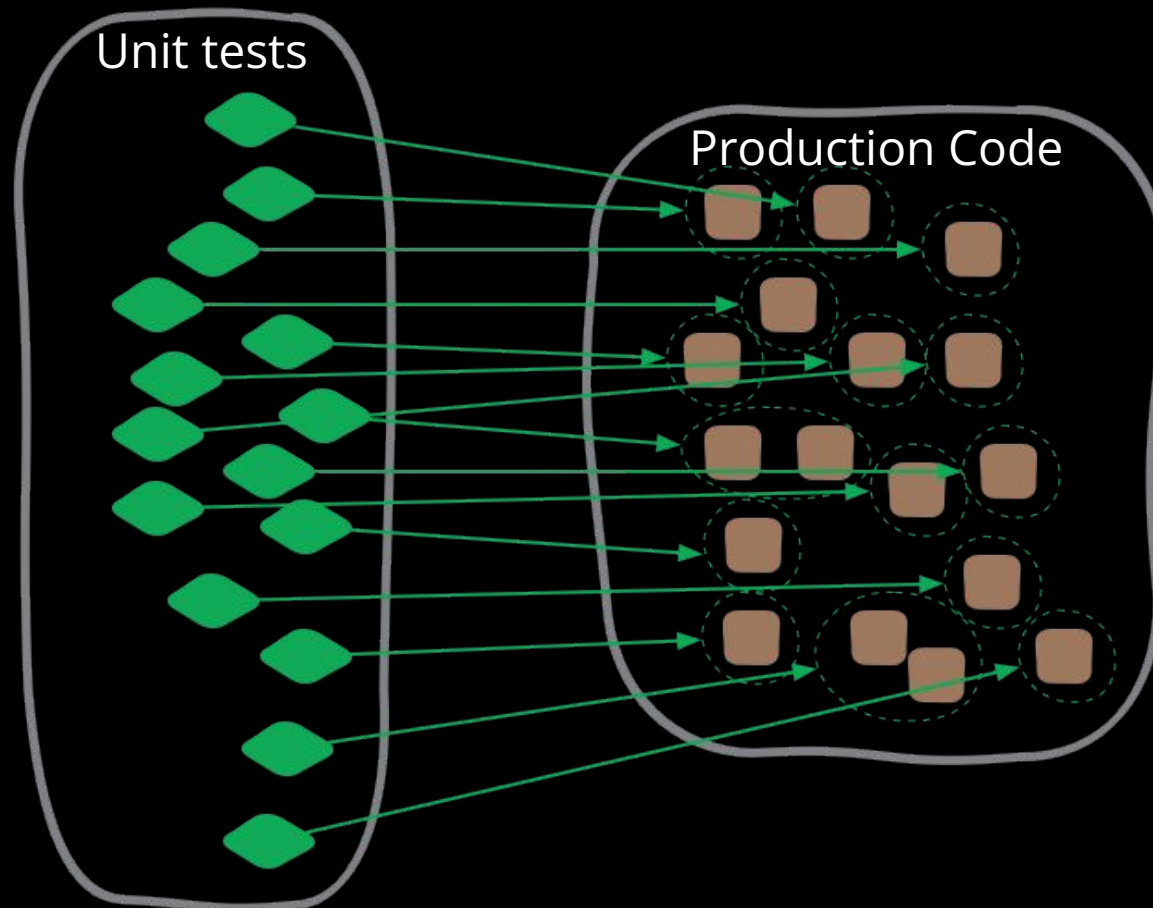
In the Main() create list of 3 triangles and write into console the information about these shapes.

\*Print the triangle with vertex which is the closest to origin (0,0)

# Unit testing with MS test

# Unit Testing

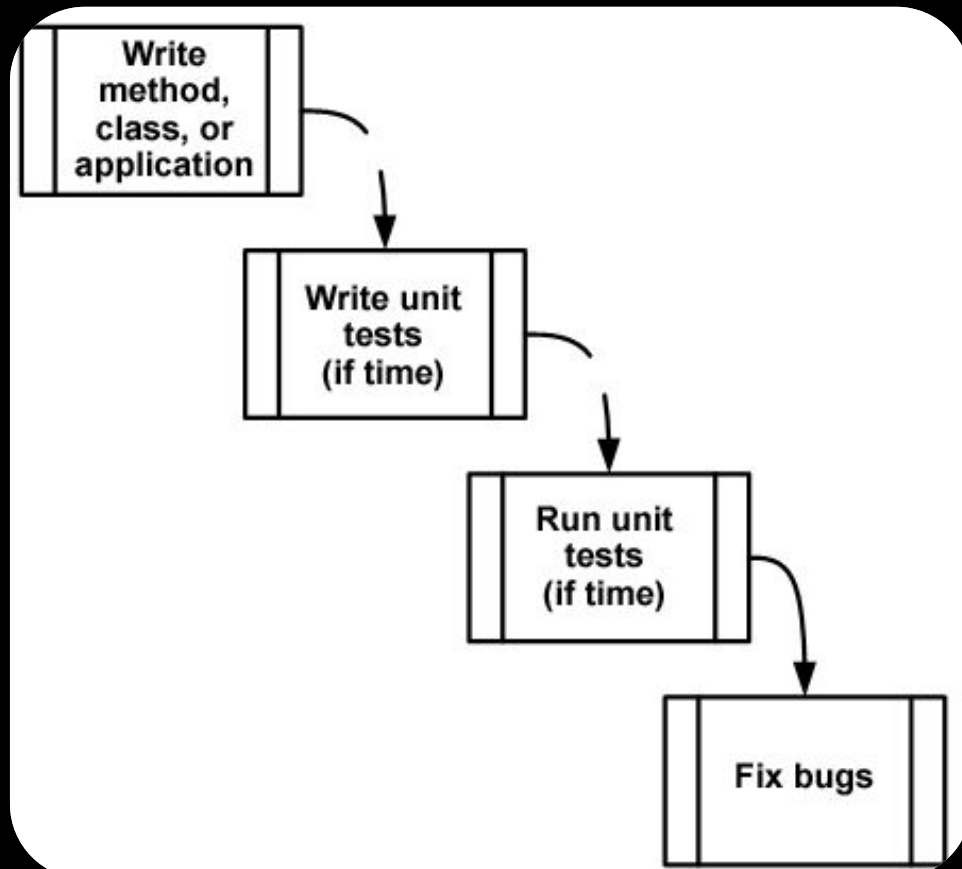
- ◆ **Unit testing** is a procedure used to validate that individual units of source code (methods, properties, classes) are working properly.



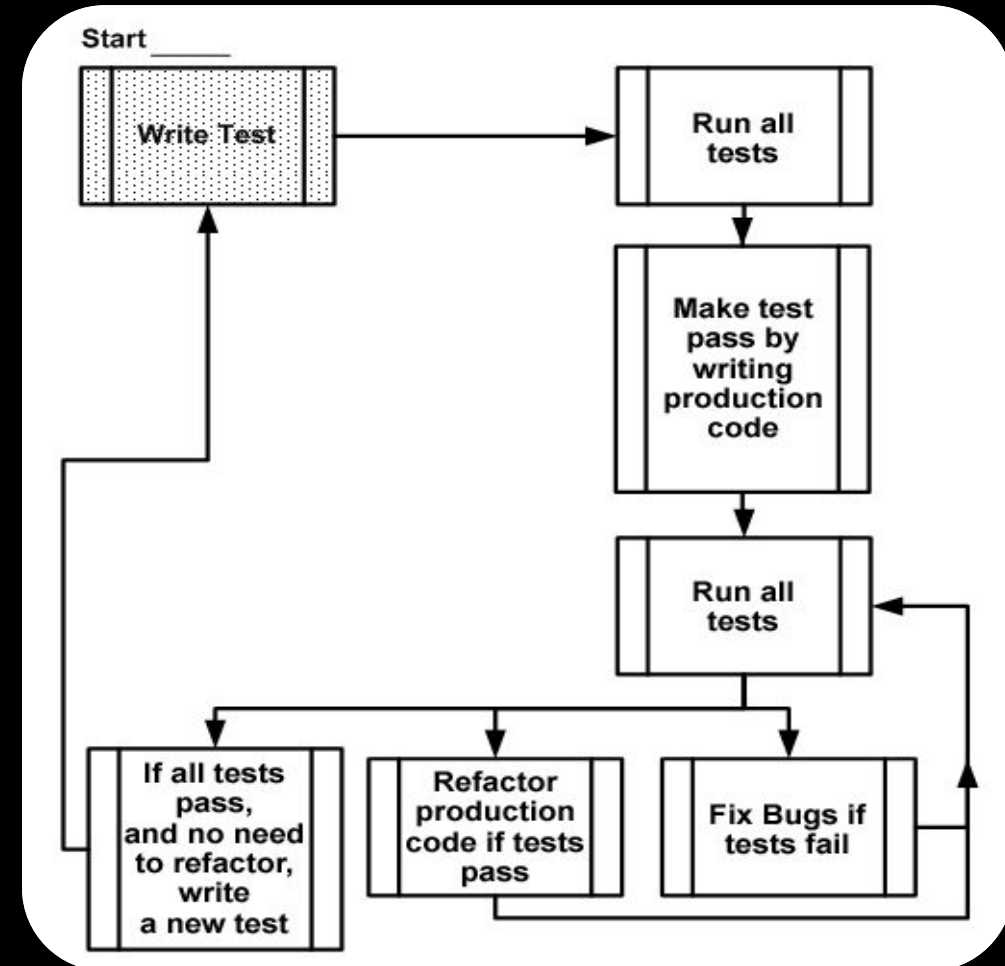


# Test-Driven Development (TDD)

❖ The traditional way of writing unit tests



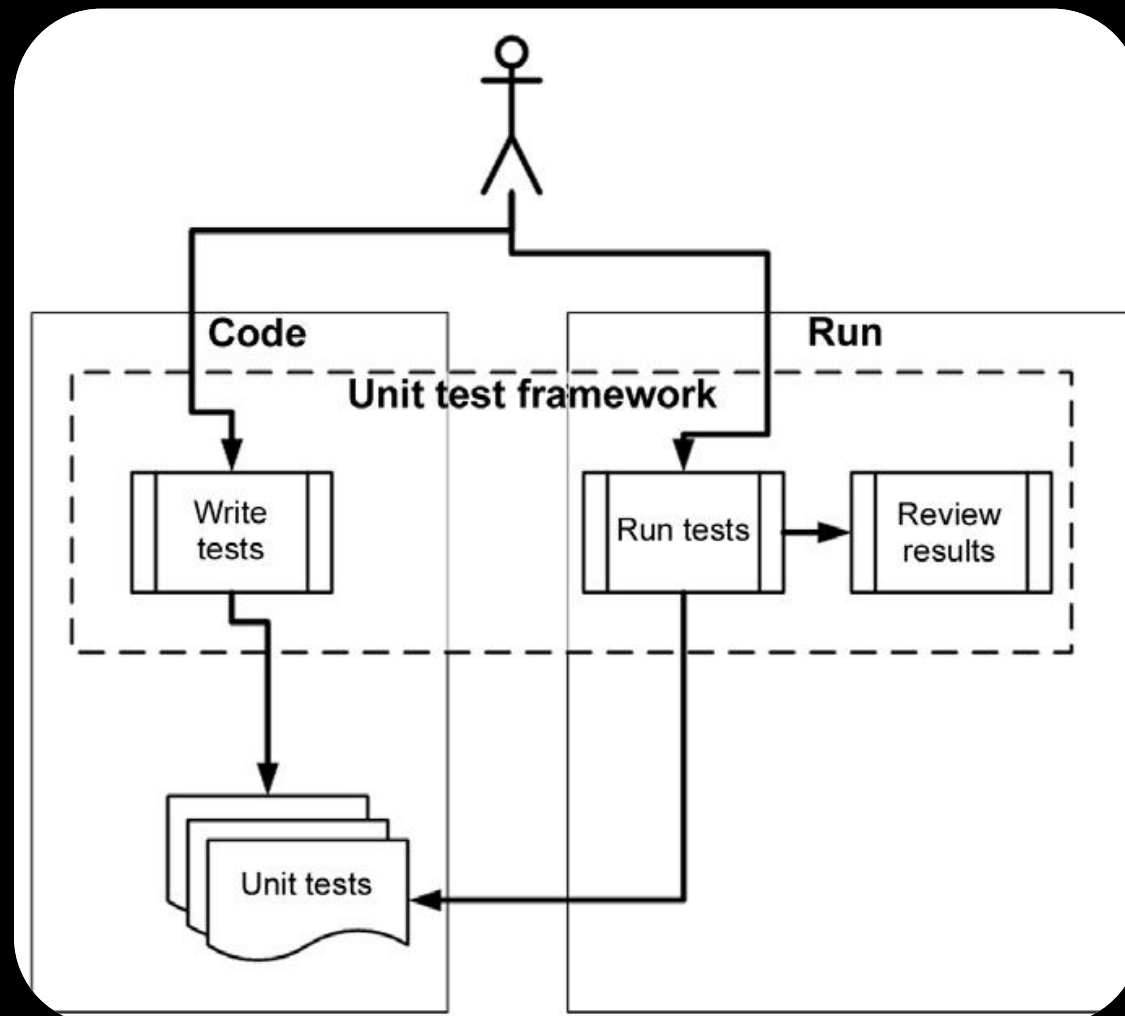
❖ Test-driven development



# Unit-testing frameworks

## ◆ UTF for .NET:

- ✓ MSTest
- ✓ NUnit
- ✓ XUnit
- ✓ ...



# Creating Unit Tests

```
[TestClass]
public class PointUnitTest
{
    [TestMethod]
    public void DistanceTest()
    {
        //Arrange
        Point p1 = new Point(0, 0);
        Point p2 = new Point(1, 0);
        double expected = 1;

        //Act
        double result = p1.Distance(p2);

        //Assert
        Assert.AreEqual(expected, result);
    }
}
```

# Kinds of Assert Classes

**Microsoft.VisualStudio.TestTools.UnitTesting** namespace:

## ❖ class **Assert**

In your test method, you can call any number of methods of the Assert class, such as `Assert.AreEqual()`. The Assert class has many methods to choose from, and many of those methods have several overloads.

## ❖ class **CollectionAssert**

Use the CollectionAssert class to compare collections of objects, and to verify the state of one or more collections.

## ❖ class **StringAssert**

Use the StringAssert class to compare strings. This class contains a variety of useful methods such as `StringAssert.Contains`, `StringAssert.Matches`, and `StringAssert.StartsWith`.

# public static class Assert


| Name                                     | Description   |
|--|---|
| <b>AreEqual(Object, Object)</b>          | Verifies that two specified objects are equal. The assertion fails if the objects are not equal.  |
| <b>AreNotEqual(Object, Object)</b>       | Verifies that two specified objects are not equal. The assertion fails if the objects are equal.  |
| <b>AreNotSame(Object, Object)</b>        | Verifies that two specified object variables refer to different objects. The assertion fails if they refer to the same object.                                    |
| <b>AreSame(Object, Object)</b>           | Verifies that two specified object variables refer to the same object. The assertion fails if they refer to different objects.                                    |
| <b>Fail()</b>                            | Fails the assertion without checking any conditions.  |
| <b>Inconclusive()</b>                    | Indicates that the assertion cannot be verified.  |
| <b>IsFalse(Boolean)</b>                  | Verifies that the specified condition is false. The assertion fails if the condition is true.   |
| <b>IsTrue(Boolean)</b>                   | Verifies that the specified condition is true. The assertion fails if the condition is false.   |
| <b>IsInstanceOfType(Object, Type)</b>    | Verifies that the specified object is an instance of the specified type. The assertion fails if the type is not found in the inheritance hierarchy of the object. |
| <b>IsNotInstanceOfType(Object, Type)</b> | Verifies that the specified object is not an instance of the specified type. The assertion fails if the type is found in the inheritance hierarchy of the object. |
| <b>IsNull(Object)</b>                    | Verifies that the specified object is null. The assertion fails if it is not null.  |
| <b>IsNotNull(Object)</b>                 | Verifies that the specified object is not null. The assertion fails if it is null.  |

# Example 1. Testing Bank project

```
public void Debit(double amount)
{
    if (m_frozen)
    {
        throw new Exception("Account frozen");
    }
    if (amount > m_balance)
    {
        throw new ArgumentOutOfRangeException("amount");
    }
    if (amount < 0)
    {
        throw new ArgumentOutOfRangeException("amount");
    }
    m_balance -= amount;
}
```

```
[TestMethod()]
public void DebitTest()
{
    BankAccount target = new BankAccount("Mr. Bryan Walton", 11.99);
    double amount = 11.22;
    target.Debit(amount);
    Assert.AreEqual((System.Convert.ToDouble(0.77)), target.Balance, 0.05);
}
```

```
[TestMethod()]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
public void DebitTest()
{
    BankAccount target = new BankAccount("Mr. Bryan Walton", 10);
    double amount = 20;
    target.Debit(amount);
    Assert.AreEqual(10, target.Balance);
}
```

|                          | Result   | Test Name | Project      | Error Message |
|--------------------------|--|-----------|--------------|---------------|
| <input type="checkbox"/> |  Passed | DebitTest | TestProject1 |               |

# Create project and Unit test example

## 1. Create project *MyCalcLib*

```
namespace MyCalcLib
{
    public class MyCalc
    {
        public int Sum(int x, int y)
        {
            return x + y;
        }
        static void Main(string[] args)
        {
            //...
        }
    }
}
```

2. Right click on *Solution*  
**Add** →  
**Test “MyCalcTests”**

3. Right click on *References*  
**Add References** →  
**project “MyCalcLib”**

# Create project and Unit test example

```
namespace MyCalcLib.Tests
{
    [TestClass]
    public class MyCalcTests
    {
        [TestMethod]
        public void Sum_10and20_30returned()
        {
            //arrange налаштувати
            int x = 10;
            int y = 20;
            int expected = 30;

            //actual
            MyCalc c = new MyCalc();
            int actual = c.Sum(x, y);

            //assert
            Assert.AreEqual(expected, actual);
        }
    }
}
```

❖ After this click ***Build Solution***



# Task & Homework 10

- ❖ Add Test project to your solution and write unit tests for Point and Triangle classes.
- ❖ Add Unit tests to one of your previous class – Person, Car ...