



หน่วยที่ 12

การสร้างคลาส



หัวข้อเรื่อง



12.1 คลาสและออบเจกต์

12.2 ออบเจกต์และไทป์

12.3 การใช้งานออบเจกต์

12.4 เอ็นแคปซูลชัน

12.5 การสืบทอด

12.6 โพลีมอร์ฟิซึม

12.7 เมธอด

12.1

คลาสและออบเจกต์

คลาสและออบเจกต์ เป็นแนวคิดพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ ซึ่งคลาสเป็นพิมพ์เขียว หรือต้นแบบที่ผู้ใช้กำหนดซึ่งวัตถุถูกสร้างขึ้น โดยพื้นฐานแล้วคลาสจะรวมฟิลด์และเมธอด (ฟังก์ชันสมาชิกซึ่งกำหนดการกระทำ) ไว้ในหน่วยเดียว คลาสและออบเจกต์เป็นสิ่งที่เกี่ยวข้องกันโดยตรง คลาสคือ โค้ด โปรแกรมที่เราเขียนขึ้นเพื่อทำหน้าที่เป็นพิมพ์เขียวของออบเจกต์



รูปที่ 12.2 คลาสและออบเจกต์

โดยทั่วไปการประกาศคลาสสามารถรวมคอมโพเนนต์เหล่านี้ตามลำดับ

1. **Modifiers** : คลาสสามารถเป็นพับบลิคหรือไพรเวท ดีฟอลต์ของคลาส คือ พับบลิค
2. **Keyword class** : คำหลักที่ใช้ในการประกาศ
3. **Class Identifier** : ตัวระบุ หรือ ชื่อของคลาสตัวอักษรเริ่มต้นซึ่งควรเป็นตัวพิมพ์ใหญ่
4. **Base class or Super class** : ชื่อของพารেন্টของคลาส (ซูเปอร์คลาส) ถ้ามีนำหน้าด้วย : (โคลอน)
5. **Interfaces** : รายการของอินเทอร์เฟซที่คั่นด้วยเครื่องหมายจุลภาคที่นำไปใช้โดยคลาสหากมีนำหน้าด้วย: (โคลอน) คลาสสามารถใช้อินเทอร์เฟซได้มากกว่าหนึ่งอินเทอร์เฟซ
6. **Body** : ร่างกายของคลาสนั้นล้อมรอบด้วย { } (เครื่องหมายปีกกา)

12.2

ออบเจกต์และไทป์

ในภาษาโปรแกรมที่ไม่ใช่ภาษาแบบ OOP เช่น ภาษา C จะมีไทป์ ต่าง ๆ ให้ใช้งาน เช่น int, float, double, char เป็นต้น ไทป์ก็คือชนิดข้อมูล เมื่อเรานิยามตัวแปรซึ่งมีชนิดข้อมูลใด ๆ ตามที่ต้องการตัวแปรภาษา จะทราบทันทีว่าต้องทำอะไรกับชนิดข้อมูลนั้น ๆ เนื่องจากว่าไทป์เหล่านั้นเป็นไทป์ที่ได้นิยาม หรือ กำหนดไว้ล่วงหน้าแล้ว เราเรียกไทป์แบบนี้ว่าไทป์ภายใน

ในภาษา C หากพบว่าไทป์ที่มีให้ไม่ตรงกับความต้องการของเรา เราสามารถนิยามไทป์ขึ้นมาใช้งานเองได้ ซึ่งเราจะเรียกไทป์แบบนี้ว่า User Defined Data Type โดยใช้สตรักเจอร์ ซึ่งการสร้างUDT ด้วยสตรักเจอร์ก็คือการนำไทป์ชนิดต่าง ๆ มาผสมกัน

คลาสก็เหมือนสตรักเจอร์ เรานิยามคลาสขึ้นมาใช้งานเพราะต้องการสร้าง UDT ใหม่ นั่นเอง เมื่อเรานิยามไทป์ขึ้นมาใหม่ ไทป์แบบ UDT ก็จะถูกคอมไพเลอร์ปรับแก้ต่อไทป์ที่เราสร้างขึ้นแบบเดียวกับ Build inType .NET Framework Library นับว่าเป็นแหล่งรวบรวมไทป์มากมายหลายพัน ไทป์ ให้เราสามารถนำมาสร้างเป็นไทป์ชนิดใหม่ที่มีความ สามารถในการทำงานที่ซับซ้อนขึ้นได้ อย่างสะดวกและรวดเร็ว

12.3

การใช้งานออบเจกต์

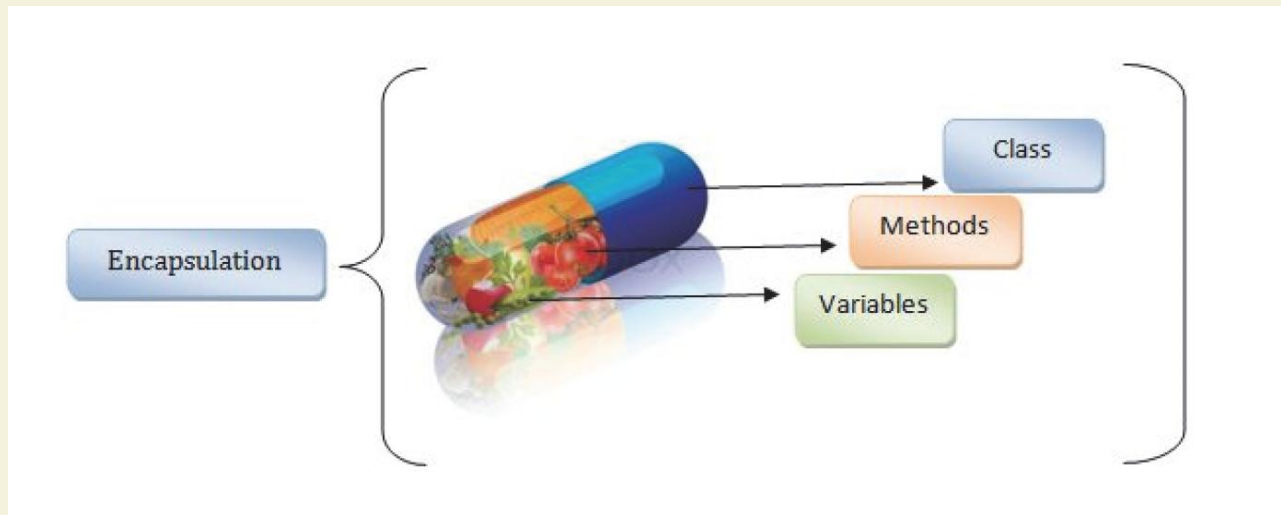
ออบเจกต์เปรียบเสมือนกล่องดำ (Black Box) ซึ่งเวลาเรานำมาใช้งานเราไม่จำเป็นต้องรู้ว่ามันมีกลไกในการทำงานอย่างไร หรือ ไม่จำเป็นต้องรู้ว่าภายในออบเจกต์นั้นมีโค้ดอะไรภายใน เมื่อเราต้องการใช้งาน เราจะส่งงานผ่านเมธอด (Method) ออบเจกต์จะผนวกขบวนการ (Method) และข้อมูล (Information) ไว้ภายในตัวของมันเอง ออบเจกต์จะเก็บข้อมูลภายในไว้ในฟิลด์ (field) หรือ ตัวแปรท้องถิ่นของคลาสนั่นเอง

โปรแกรมที่เรียกใช้ออบเจกต์ไม่จำเป็นและสามารถเข้าถึง หรือเปลี่ยนแปลงข้อมูลในฟิลด์ได้โดยตรงแต่สามารถเข้าถึงได้โดยทางอ้อมผ่านส่วนเชื่อมต่อที่เรียกว่าพร็อพเพอร์ตี้ (Property)

12.4

เอ็นแคปซูลชัน

เอ็นแคปซูลชัน หรือ คุณสมบัติการห่อหุ้ม การห่อหุ้ม คือ แนวคิดของออบเจกต์ที่แยกโค้ดส่วนเชื่อมต่อ กับโค้ดส่วนประมวลผลออกจากกัน ข้อมูลและส่วนประมวลผลทั้งหมดจะถูกซ่อนไว้เบื้องหลังอินเทอร์เฟซ หากเราจัดให้มีการอินเทอร์เฟซที่ดีและรักษารูปแบบการเชื่อมต่อไว้อย่างคงที่



รูปที่ 12.3 Encapsulation

12.4.1 รวมส่วนข้อมูลและส่วนประมวลผลไว้ด้วยกัน

ส่วนของข้อมูลคลาสเราจะเรียกส่วนนี้ว่า Data Field หรือ Variable Filed ทำหน้าที่เก็บข้อมูล ในส่วนของการประมวลผลซึ่งเป็นสมการคณิตศาสตร์อย่างง่าย ๆ ที่ทำหน้าที่แปลงหน่วยวัดอุณหภูมิจากองศาเซลเซียสไปเป็นองศาฟาเรนไฮต์การรวมข้อมูลและส่วนประมวลผลไว้ด้วยกันแบบนี้คือลักษณะอย่างหนึ่งของหลักการเอ็นแคปซูลेशन

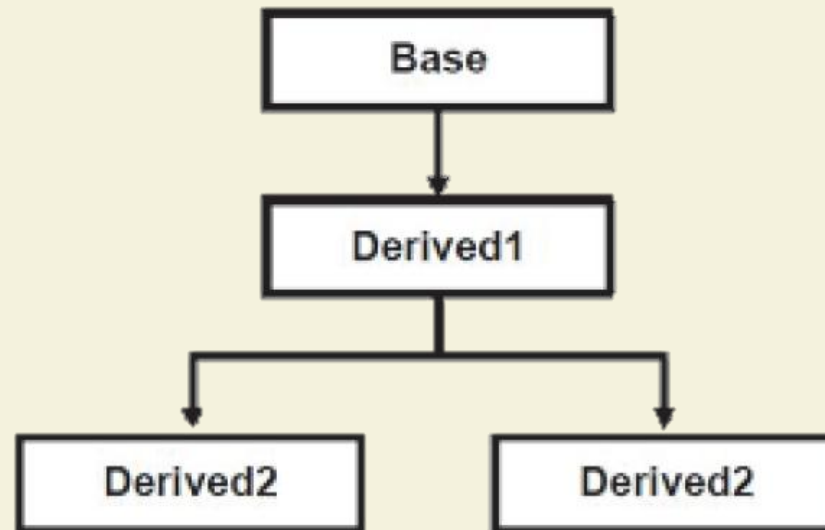
12.4.2 ใช้กลไกในการเชื่อมกับโค้ดภายนอกด้วยพรีอปเพอร์ตี้

หากเราสามารถกันไม่ให้โค้ดภายนอก สามารถเข้าถึงส่วนของข้อมูลภายในออบเจกต์ได้ คลาสนั้น ๆ ก็มีคุณสมบัติของการเอ็นแคปซูลेशन ทำให้เราสามารถส่งค่าองศาเซลเซียสไปให้ออบเจกต์ได้โดยไม่มีการแตะต้องส่วนของ Data Filed ของคลาสเลย

12.5

การสืบทอด

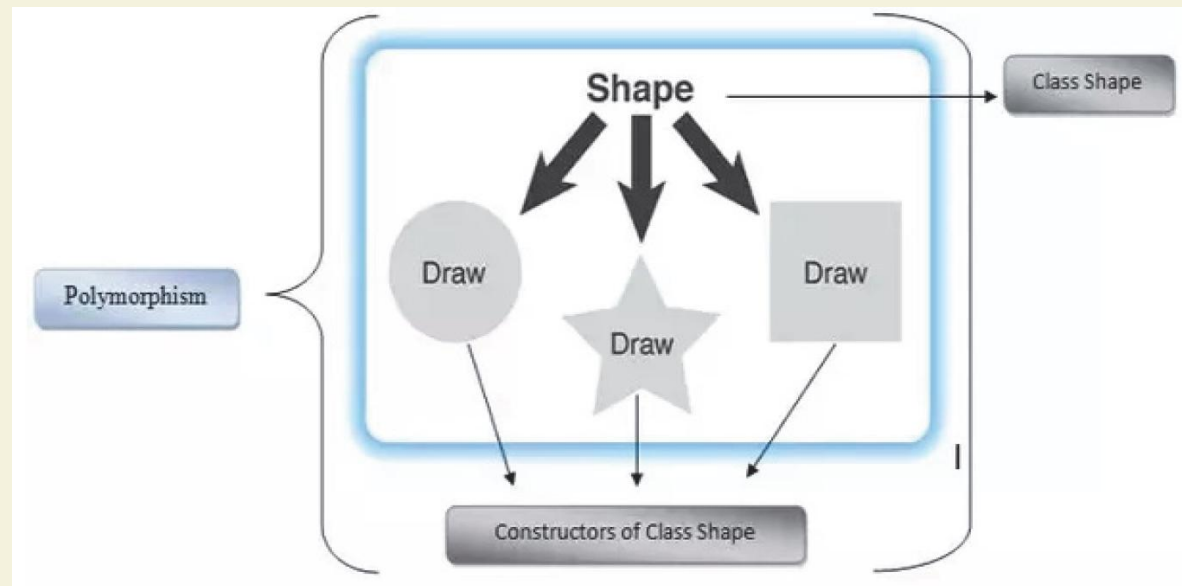
วิธีการอย่างง่าย ๆ ในการตรวจสอบดูว่าภาษาคอมพิวเตอร์ใดสนับสนุนแนวความคิดในแบบ OOPหรือไม่นั้น ให้ตรวจสอบดูว่าภาษาคอมพิวเตอร์ภาษานั้นมีคุณสมบัติของการสืบทอด หรือ การอินเฮอร์ิตเอนซ์หรือไม่ เรียกว่าอินเฮอร์ิต หลักการสำคัญนี้มีไว้เพื่อให้เราสามารถต่อยอดงานใหม่จากงานเดิมที่มีอยู่แล้วนั่นเอง



รูปที่ 12.4 การสืบทอด

12.6 โพลีมอร์ฟิสม์

การแปรคุณสมบัติ คำว่าโพลี (Poly) แปลว่าหลาย และคำว่ามอร์ฟิสม์ แปลว่ารูปร่าง ดังนั้นคำว่าโพลีมอร์ฟิสม์นี้จึงแปลว่าหลายรูปร่าง หรือ เปลี่ยนสภาพได้ อะไรที่เปลี่ยนรูปร่างได้คำตอบ คือ เมธอดที่อินเฮียริตมาจากเบสคลาส



รูปที่ 12.5 Polmorphism

ตัวอย่างแสดงการทำเมธอดโอเวอร์ไรท์อย่างง่าย ๆ มีดังนี้

```
Class CustomerData
{
    protected virtual void CustomerPhone()
    {
        Console.WriteLine("02 123 456");
    }
    protected virtual void ClearAll()
    {
        Console.WriteLine("deleat all data");
    }
}
```

สร้างคลาสใหม่โดยการอินเียริตจากคลาส CustomerData

```
Class CustomerData2 : CustomerData
{
    protected override void CustomerPhone()
    {
        Base. CustomerPhone()
        Cosole.WriteLine("02 123 456 7");
    }
    protected new virtual void ClearAll()
    {
        Cosole.WriteLine("deleat all data");
    }
}
```


12.7

เมธอด

เมธอด (Method) เป็นองค์ประกอบอย่างหนึ่งของคลาส สำหรับกำหนดรูปแบบ วิธีการกระทำอย่างใดอย่างหนึ่ง

12.7.1 การสร้างเมธอด

จากตัวอย่างที่ผ่านมาเรามีเมธอดที่เราได้รู้จัก คือ Main()

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //...
        }
    }
}
```

โดยทั่วไปแล้วคลาสมักประกอบไปด้วยเมธอดมากกว่า 1 เมธอด สำหรับกำหนดการทำงานเฉพาะอย่างนั้น คือ เราสามารถเพิ่มเมธอดให้กับคลาสได้ จะที่เมธอดก็ได้ตามความจำเป็น หากเราเพิ่มเมธอดลงไปในคลาสจะมีลักษณะดังนี้

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //...
        }
        static (returnValueType, void) MethodName(parameter (s))
        {
            //...
        }
    }
}
```

จากเมธอดที่ได้เพิ่มเติมลงไป สามารถอธิบายได้ดังนี้

1. returnType ใช้ในกรณีที่เมธอดนั้นจำเป็นต้องส่งผลลัพธ์ที่ประมวลผลภายในเมธอดนั้นกลับไปยังส่วนที่เรียกใช้ เราจำเป็นต้องระบุชนิดข้อมูลเสมอ

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //...
        }
        static double CalCircleArea(int r)
        {
            //...
        }
    }
}
```

2. ส่งผลลัพธ์ด้วย return

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //...
        }
        static double CalCircleArea(int r)
        {
            Double area = 3.14 * r * r;
            Return area;
        }
    }
}
```


3. การรับพารามิเตอร์ กรณีที่เมธอดมีหลายพารามิเตอร์จะเป็นดังนี้

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //...
        }
        static double CalXYZ(int x, double y, string z)
        {
            //...
        }
    }
}
```

12.7.2 การเรียกใช้เมธอด

การเรียกใช้เมธอดมี 2 แบบ คือ

1. การเรียกใช้ void method หรือ เมธอดไม่คืนค่า กรณีนี้เพียงระบุชื่อเมธอดและพารามิเตอร์(ถ้ามี) เช่น

```
String msg = "Hello World;  
Say(msg);
```

2. การเรียกใช้เมธอดแบบคืนค่า จำเป็นต้องมีตัวแปรมารองรับค่าที่ส่งกลับ เช่น

```
Double myArea = CalCircleArea(10);
```

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            double radius; string s;
            Console.Write("Please enter radius : ");
            s = Console.ReadLine();
            radius = double.Parse(s);
            Console.WriteLine("Area = {0}, + Circumference = {1}"
                , CalArea(radius), CalCircumference(radius));
            Console.Read();
        }
        static double CalArea(double r)
        {
            return 3.14 * r * r;
        }
        static double CalCircumference(double r)
        {
            return 2 * 3.141 * r;
        }
    }
}
```

Please enter radius : 100

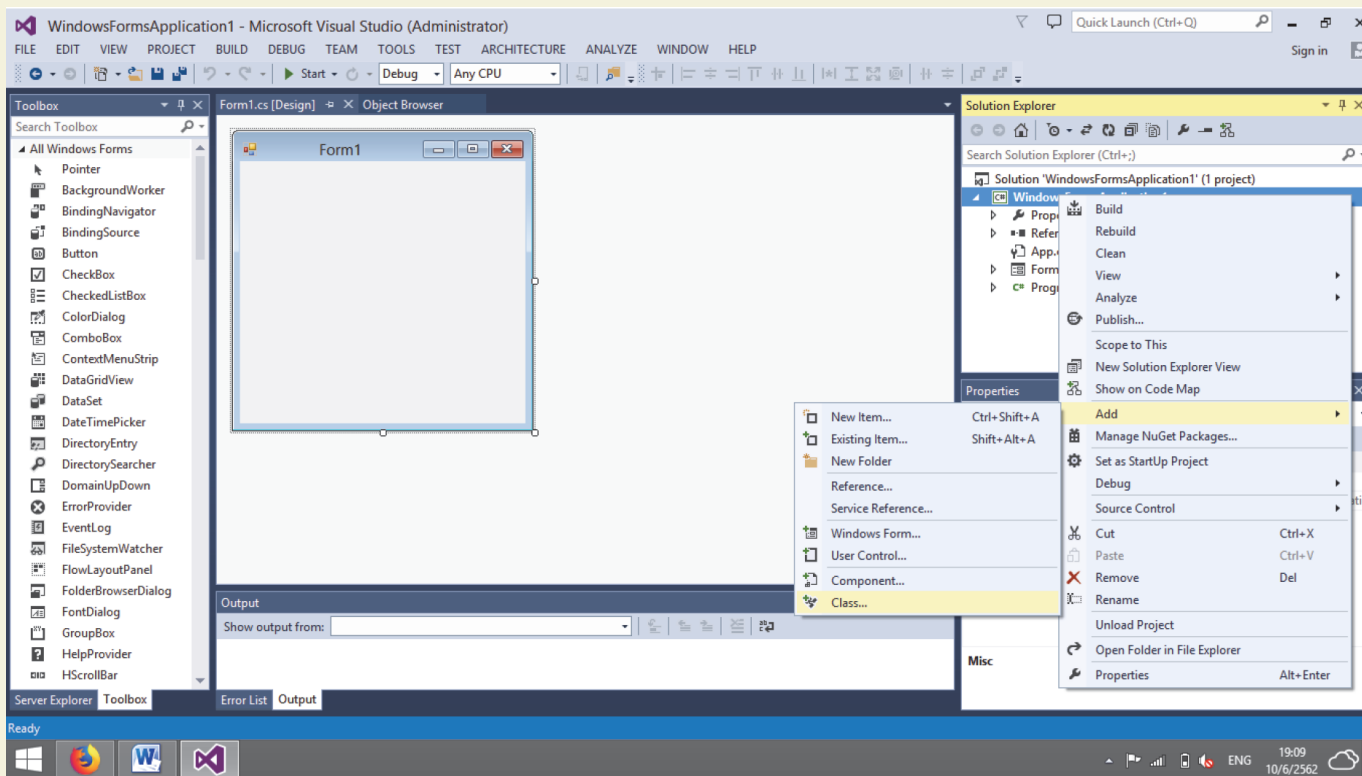
Area = 13140, + Circumference = 628.2

—

ฝึกปฏิบัติการสร้าง Class และเรียกใช้งาน Class C#

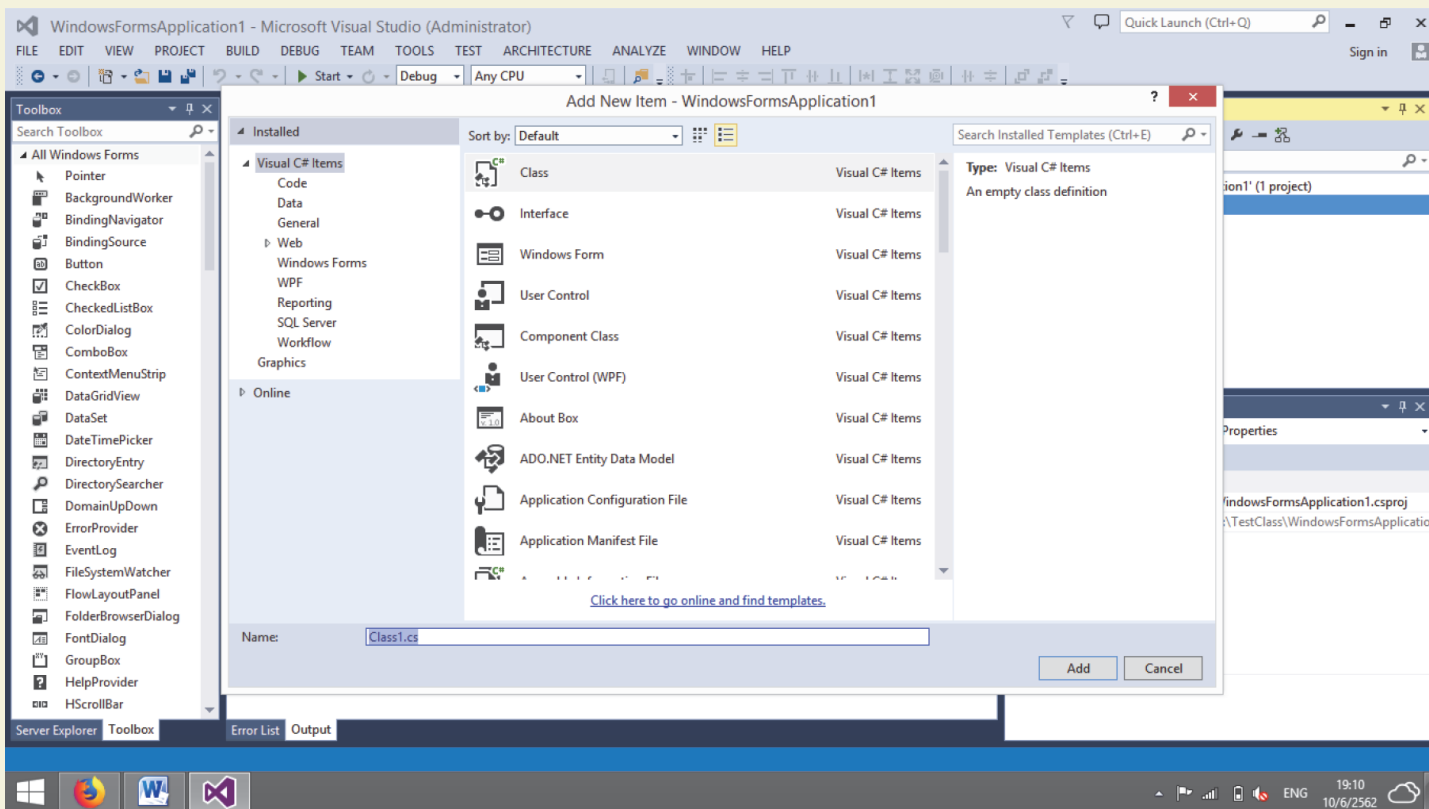
1. สร้าง Class ในแบบ Windows App ดังนี้

1.1 สร้างโปรเจกต์ใหม่ในแบบ Windows App



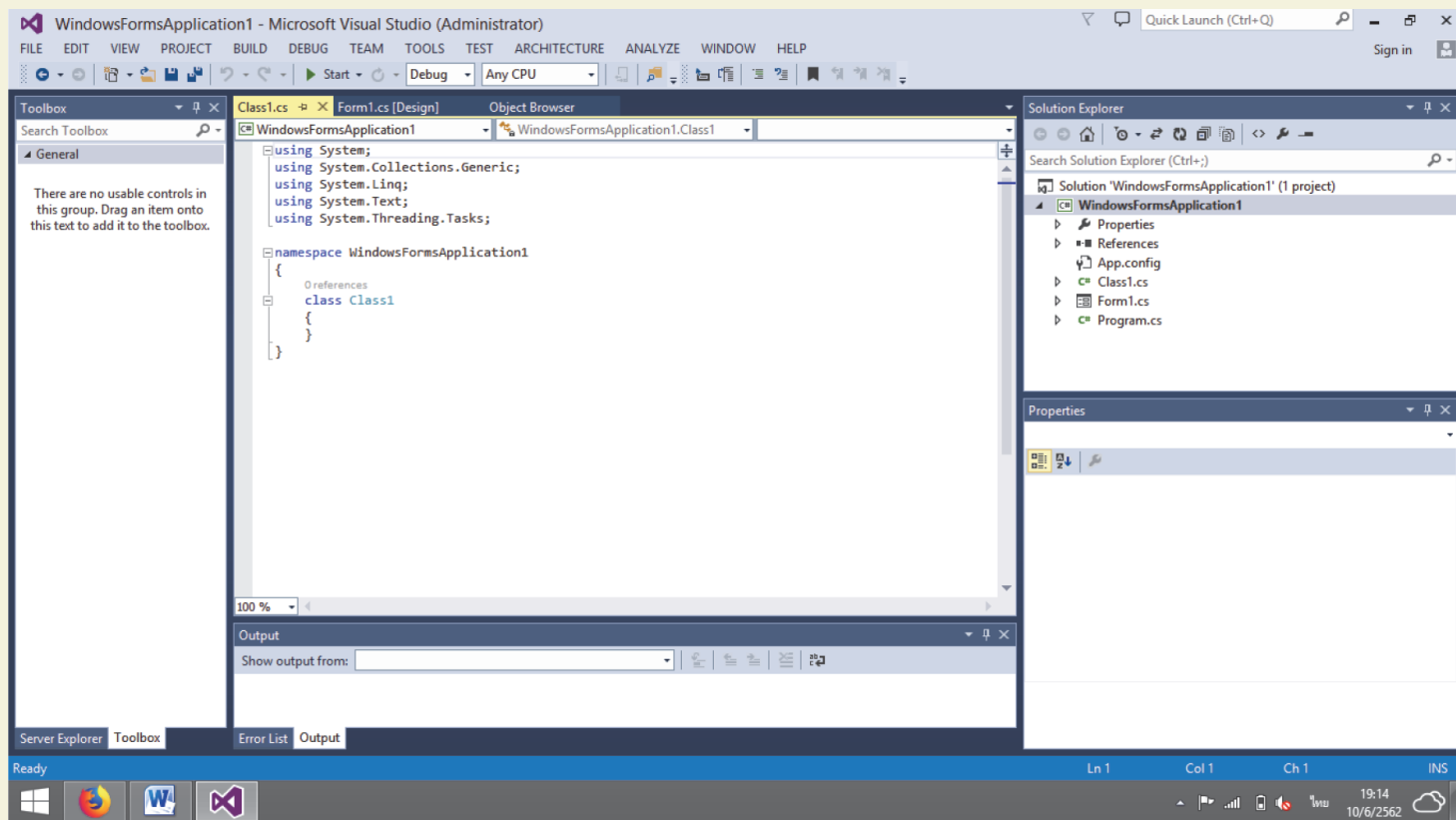
รูปที่ 12.6 การสร้างโปรเจกต์ใหม่ในแบบ Windows App

1.2 คลิกร Add



รูปที่ 12.7 คลิกร Add

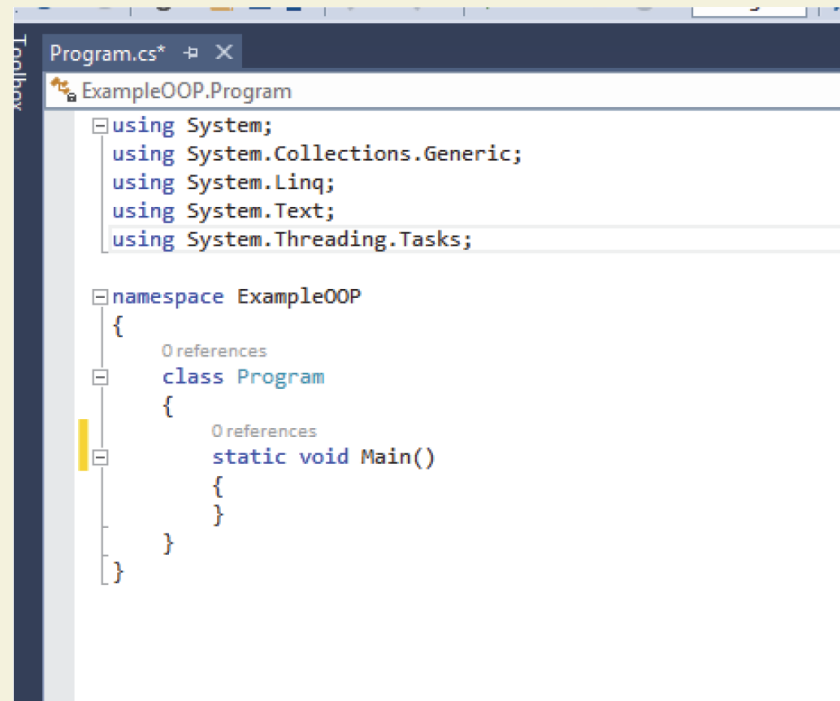
1.3 จะได้ Class1 ตั้งชื่อใหม่หรือไม่ก็ได้



รูปที่ 12.8 Class1

2. สร้าง Class ในแบบ Console App ดังนี้

2.1 สร้างโปรเจกต์ใหม่ในแบบ Windows App

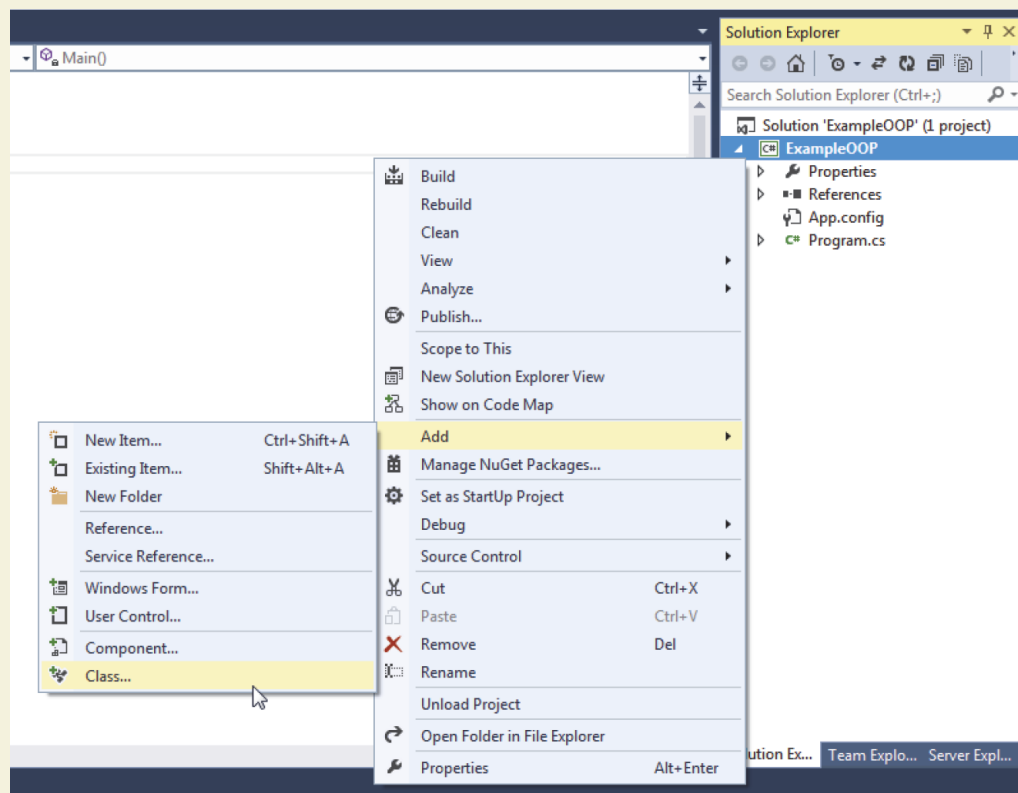


```
Program.cs* X
ExampleOOP.Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ExampleOOP
{
    0 references
    class Program
    {
        0 references
        static void Main()
        {
        }
    }
}
```

รูปที่ 12.9 สร้างโปรเจกต์ใหม่ในแบบ Windows App

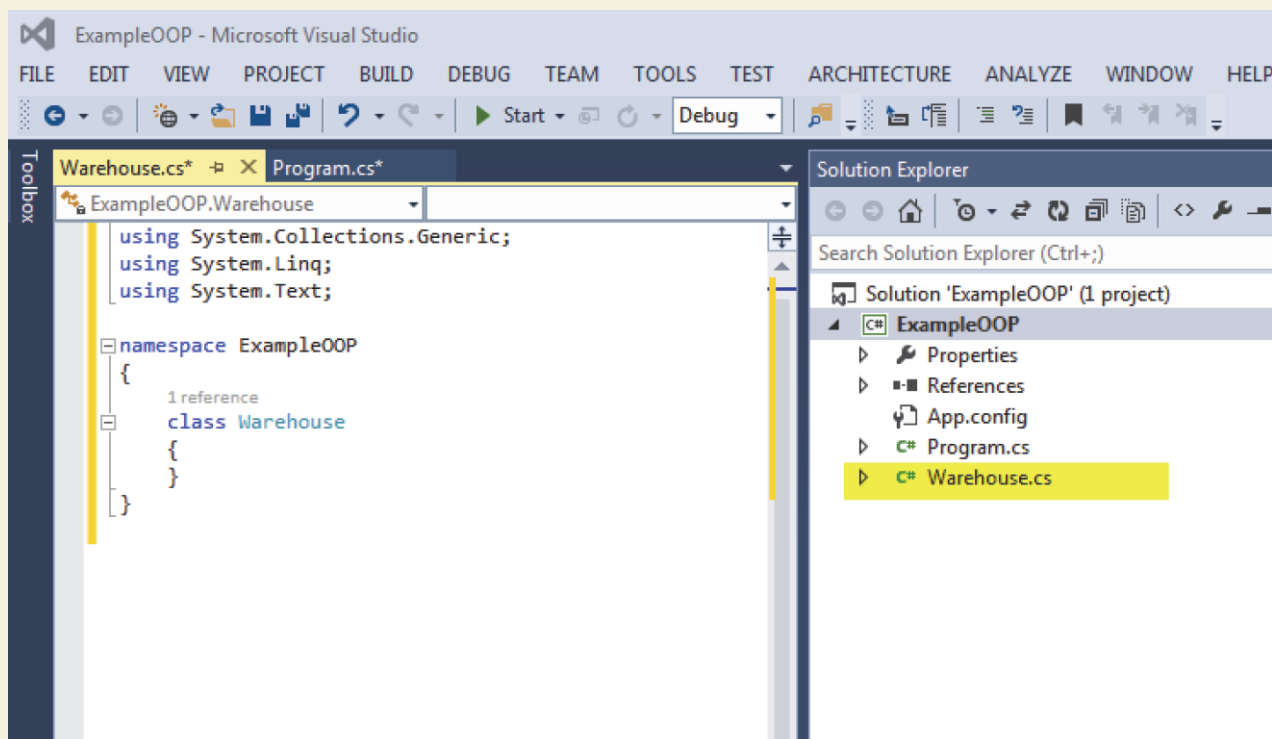
2.2 คลิก Add



รูปที่ 12.10 คลิก Add

2.3 คลิก Add

2.4 ตั้งชื่อ Class



รูปที่ 12.11 Class