Objects and Classes in C#

What is a class?

- A struct with built-in functions
- A blueprint from which individual objects are created.

```
class Circle
{
      double radius = 0.0;

      double Area()
      {
          return 3.141592 * radius * radius;
      }
  }
```

*Encapsulation

A class definition, by default, *encapsulates*, or hides, the data inside it.

This is a key concept of object oriented programming.

The outside world can see and use the data only by calling the build-in functions.

- Called “methods”

Class Members

Methods and variables declared inside a class are called *members* of that class.

- Member variables are called *fields*.
- Member functions are called *methods*.

In order to be visible outside the class definition, a member must be declared *public*.

As written in the previous example, neither the variable radius nor the method Area could be seen outside the class definition.

Making a Method Visible

To make the Area() method visible outside we would write it as:

```
public double Area()
{
        return 3.141592 * radius * radius;
}
```

Unlike C++, we have to designate individual members as public.

Not a block of members.

We will keep the radius field private.

A Naming Convention

- By convention, public methods and fields are named with the first letter capitalized.
  - Also class names.
- Private methods and fields are named in all lower case.
- This is *just a convention*.
  - It is not required, and it means nothing to the compiler.

Interface vs. Implementation

- The public definitions comprise the *interface* for the class
  - A *contract* between the creator of the class and the users of the class.
  - Should never change.

- *Implementation* is private
  - Users cannot see.
  - Users cannot have dependencies.
  - Can be changed without affecting users.

Creating Objects

- The class definition does not allocate memory for its fields.

  (Except for *static* fields, which we will discuss later.)

- To do so, we have to create an *instance* of the class.

```
static void Main(string[ ] args)
{
        Circle c;
        c = new Circle();
}
```

Objects

An object is an instance of a class.

You can create any number of instances of a given class.

- Each has its own identity and lifetime.
- Each has its own copy of the fields associated with the class.

When you call a class method, you call it through a particular object.

The method sees the data associated with *that object*.

Using Classes and Objects

- Classes and objects are used much like traditional types and variables:
  - Declare variables
    - Like pointers to structs
      - `Circle c1;`
    - Can be member variables in other classes
  - Assignment

    `c2 = c1;`
  - Function arguments

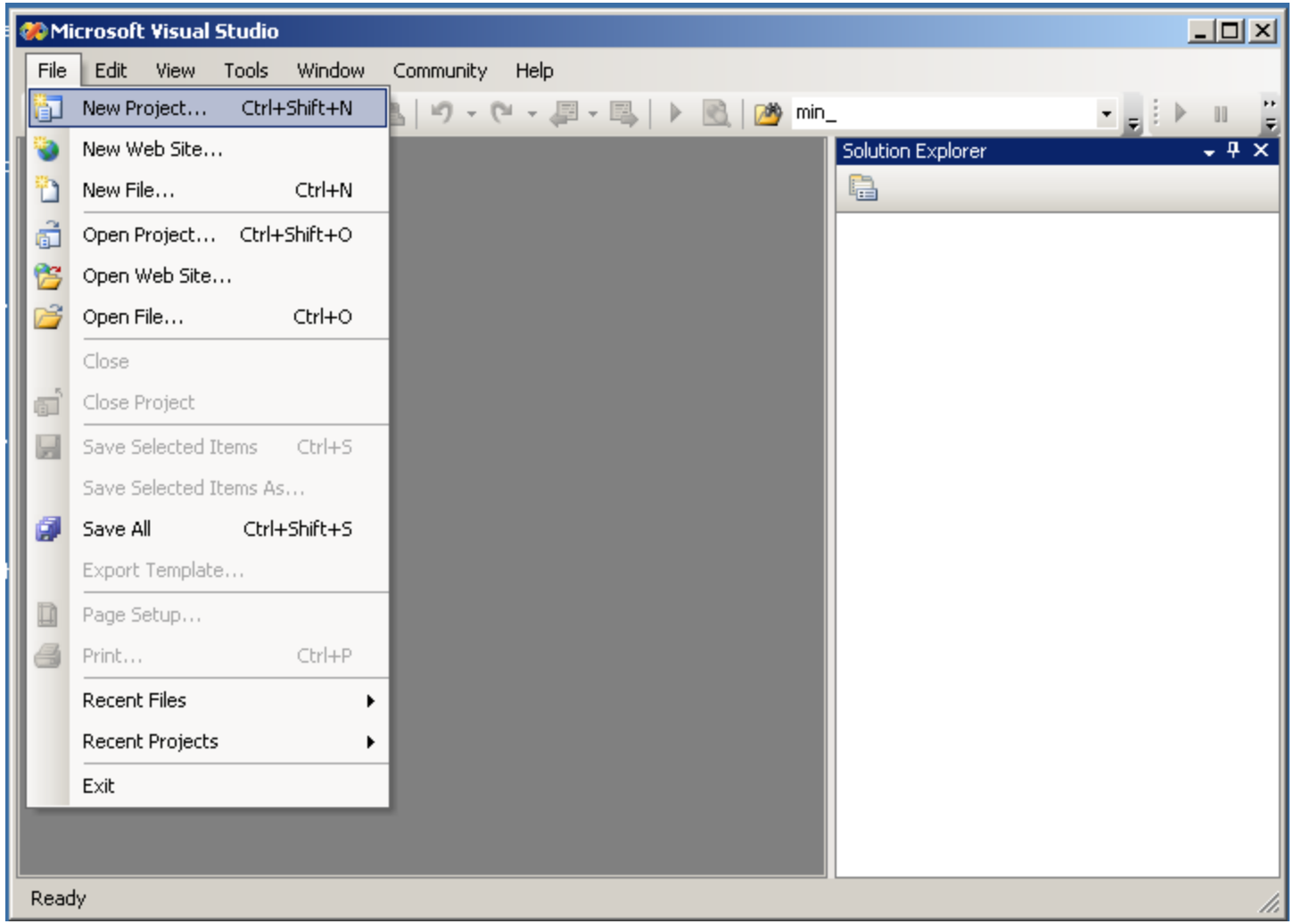
    `picture1.crop(c1);`

Program Circle Demo

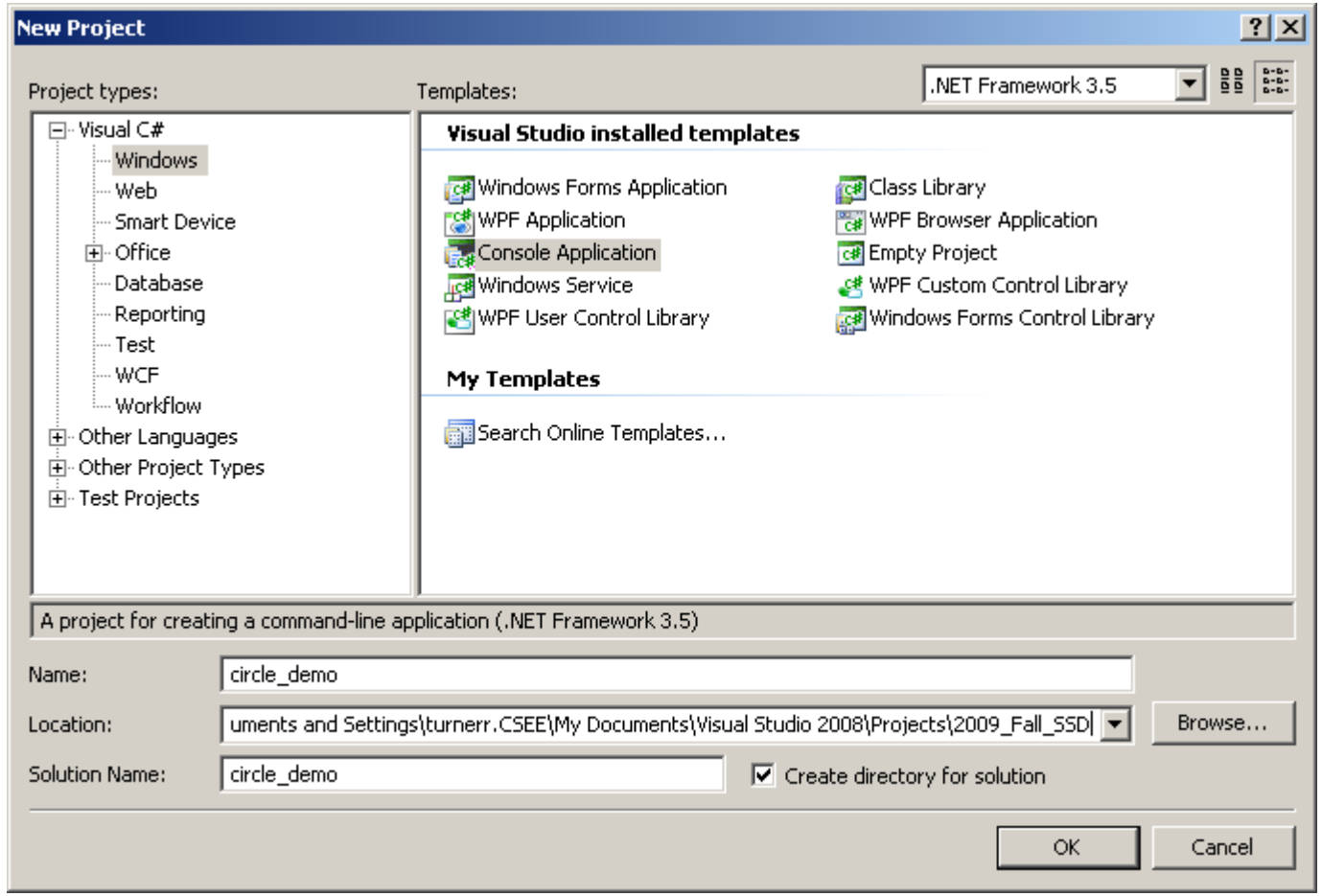
- Demonstrate creating Program Circle in Visual Studio.

- Demonstrate adding a class to a project

Create a New Project



Create a New Project



Program Template

circle_demo - Microsoft Visual Studio

File Edit View Project Build Debug Data Tools Test Window Help

Debug | Any CPU

Program.cs | Start Page

circle_demo.Program | Main(string[] args)

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace circle_demo
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12         }
13     }
14 }
15
```

Solution Explorer - circle_demo

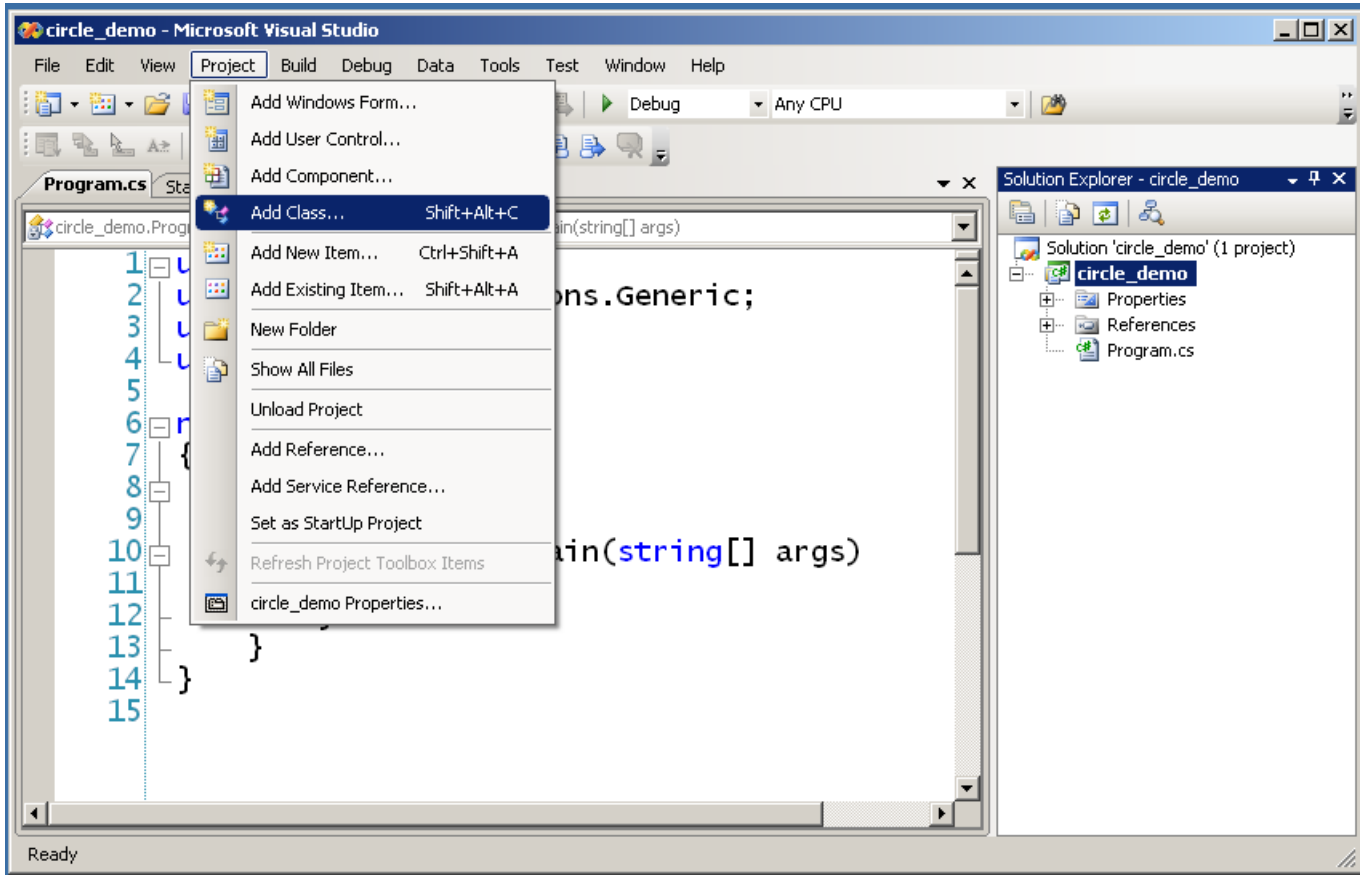
- Solution 'circle_demo' (1 project)
  - circle_demo
    - Properties
    - References
    - Program.cs

Ready

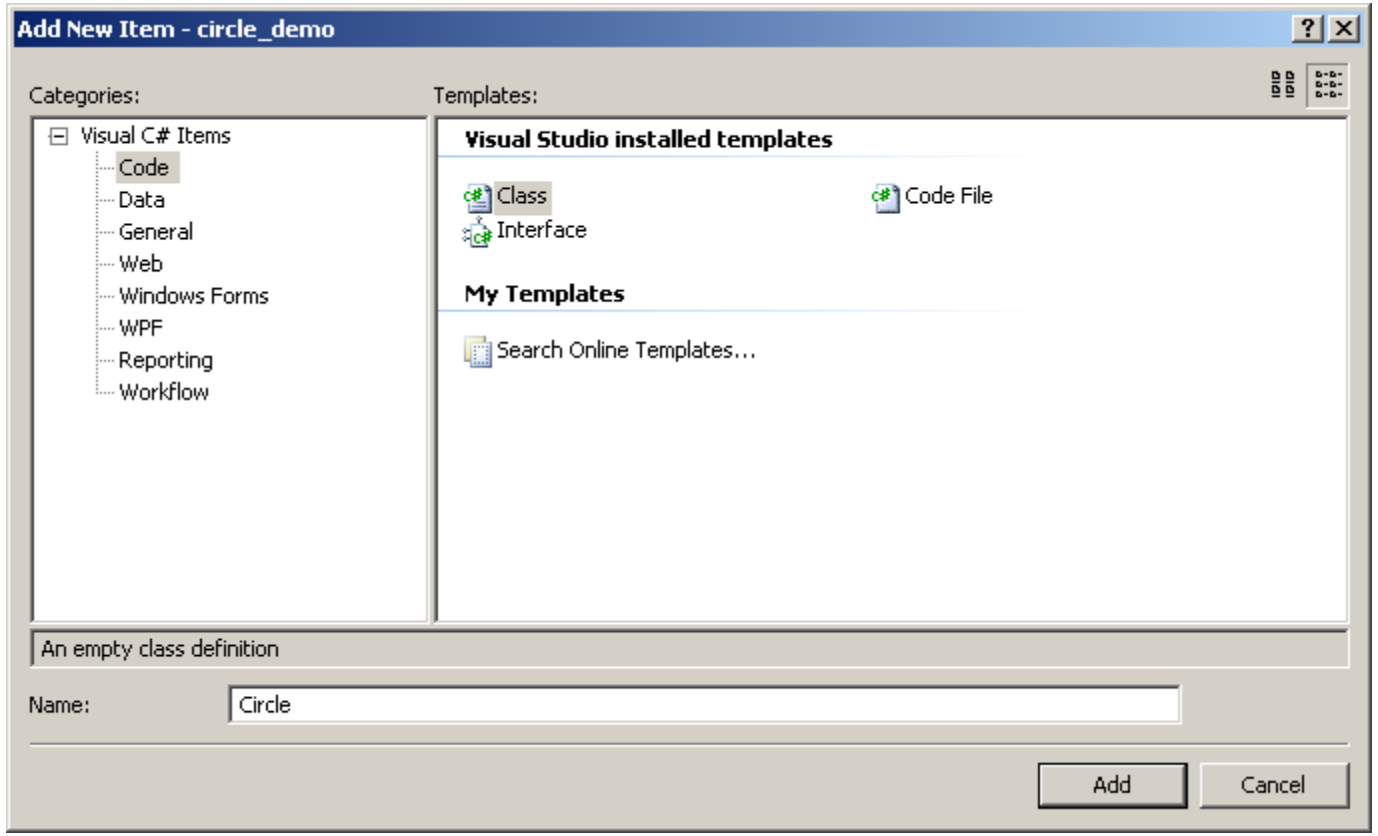
Adding a Class to a Program

- Each class definition *should* be a separate file.
- In Visual Studio.
  - Project menu > Add Class
  - Use class name as file name.

Add a Class to the Project



Adding Class Circle



Initial Source File

circle_demo - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Window Help

Debug | Any CPU

Circle.cs | Program.cs | Start Page

circle_demo.Circle

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace circle_demo
7  {
8      class Circle
9      {
10     }
11 }
12
```

Solution Explorer - ...

Solution 'circle_demo' (1 p

circle_demo

Properties

References

Circle.cs

Program.cs

Ready | Ln 1 | Col 1 | Ch 1 | INS

Fill in Class Definition

circle_demo - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Window Help

Debug | Any CPU

Circle.cs* | Program.cs | Start Page

circle_demo.Circle | radius

```
1  using System;
2
3  namespace circle_demo
4  {
5      class Circle
6      {
7          private double radius = 0.0;
8
9          public double Area()
10         {
11             return Math.PI * radius * radius;
12         }
13     }
14 }
15
```

Solution Explorer - ...

Solution 'circle_demo' (1 p

circle_demo

Properties

References

Circle.cs

Program.cs

Ready | Ln 2 | Col 1 | Ch 1

Fill in Main()

circle_demo - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Window Help

Circle.cs | Program.cs | Start Page

circle_demo.Program | Main(string[] args)

```
1  using System;
2
3  namespace circle_demo
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Circle c1;
10
11             c1 = new Circle();
12
13             double c1_Area = c1.Area();
14
15             Console.WriteLine(c1_Area);
16
17             Console.ReadLine();  // keep window open.
18         }
19     }
20 }
21
```

Call function Area() of Circle object c1

Solution 'circle_demo'
circle_demo
Properties
References
Circle.cs
Program.cs

Item(s) Saved Ln 18 Col 1 Ch 1

Build and Run

circle_demo - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Window Help

Build circle_demo
Rebuild circle_demo
Clean circle_demo
Publish circle_demo
Configuration Manager...

Circle.cs Program.cs* Output

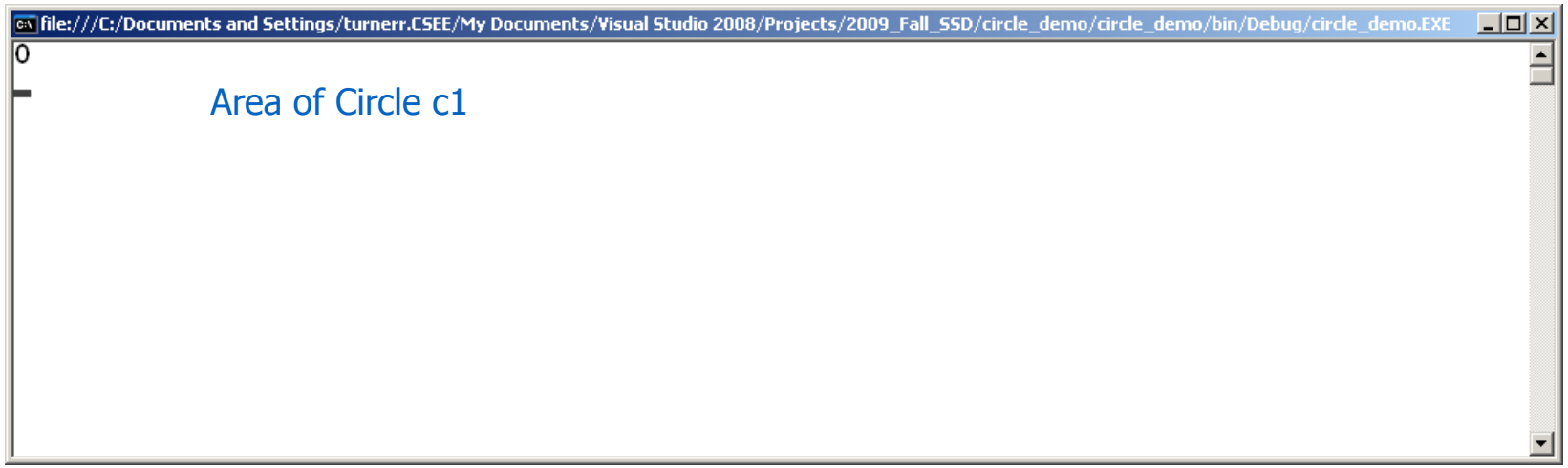
```
1  using System;
2
3  namespace circle_demo
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Circle c1;
10
11             c1 = new Circle();
12
13             double c1_area = c1.Area();
14
15             Console.WriteLine(c1_area);
16
17             Console.ReadLine();  // Keep window open.
18         }
19     }
20 }
21
```

Solution Explorer...
circle_demo
Properties
References
Circle.cs
Program.cs

Solutio... Properties

Build succeeded Ln 1 Col 14 Ch 14 INS

Program circle_demo in Action



Constructors

So far we have no way to set or change the value of radius of a Circle.

We can use a *constructor* to set an initial value.

A constructor is a method with the same name as the class. It is invoked when we call *new* to create an instance of a class.

In C#, unlike C++, you *must* call new to create an object.

Just declaring a variable of a class type does not create an object.

A Constructor for Class Circle

We can define a constructor for Circle so that it sets the value of radius.

```
class Circle
{
        private double radius;
        ...
        public Circle (double r)
        {
            radius = r;
        }
        ...
}
```

Note: Constructors have no return type.

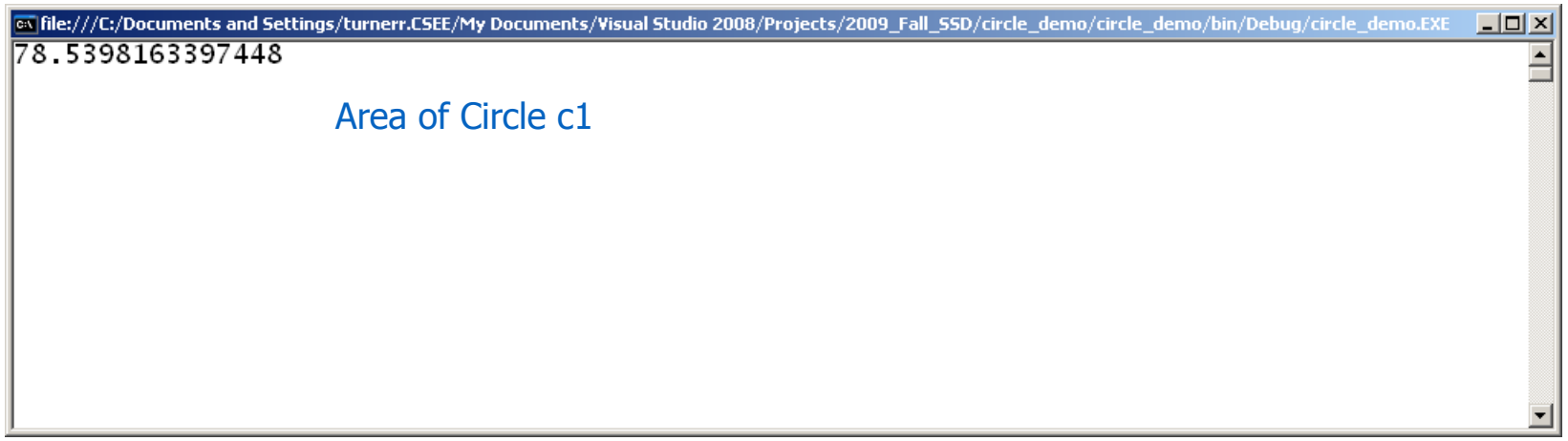
Using a Constructor

circle_demo - Microsoft Visual Studio

```
using System;

namespace circle_demo
{
    class Program
    {
        static void Main(string[] args)
        {
            Circle c1;

            c1 = new Circle(5.0);

            double c1_Area = c1.Area();

            Console.WriteLine(c1_Area);

            Console.ReadLine();  // keep window open.
        }
    }
}
```

Calls constructor

Program Running



Multiple Constructors

A class can have any number of constructors.

All must have different *signatures*.
(The pattern of types used as arguments.)

This is called *overloading* a method.

Applies to *all* methods in C#. Not just constructors.

Different *names* for arguments don't matter,
Only the types.

Default Constructor

If you don’t write a constructor for a class, the compiler creates a default constructor.

The default constructor is public and has no arguments.

```
c = new Circle();
```

The default constructor sets numeric variables to zero and Boolean fields to *false*.

In constructors that you write, the same is true for any variables that you don’t initialize.

Creating multiple objects of the same type

circle_demo - Microsoft Visual Studio

File Edit View Project Build Debug Data Tools Test Window Help

Debug | Any CPU | table

Start Page | Circle.cs | Program.cs*

circle_demo.Program | Main(string[] args)

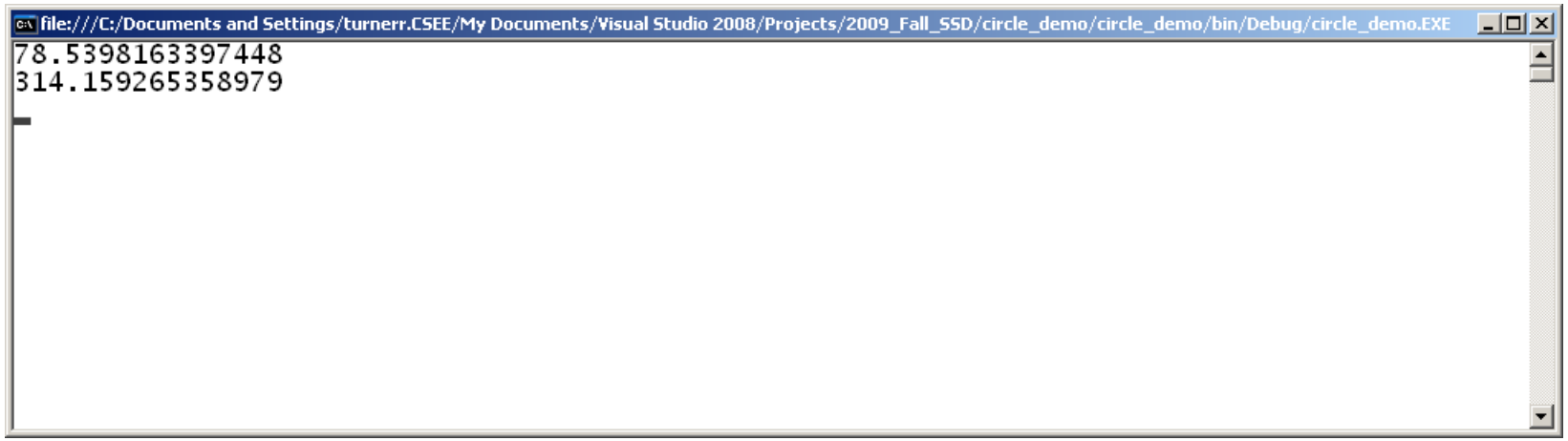
```
1  using System;
2
3  namespace circle_demo
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Circle c1;
10             Circle c2;
11
12             c1 = new Circle(5.0);
13             c2 = new Circle(10.0);
14
15             double c1_Area = c1.Area();
16
17             Console.WriteLine(c1_Area);
18             Console.WriteLine(c2.Area());
19
20             Console.ReadLine();  // keep window open.
21         }
22     }
23 }
```

Solution Explorer

Solution 'circle_demo' (1 project)
- circle_demo
  - Properties
  - References
  - Circle.cs
  - Program.cs

Ready

Program Running



Good Programming Practice

- All member variables should be private.
  - except const variables
- Users of the class can use them and manipulate them *only* by invoking the public methods of the class.
- Only way for users to do *anything* with an object.

Class Circle

- Let's extend class Circle by providing *names* for circle objects.

- Also provide *accessor* functions
  - Public functions that let the outside world access attributes of an object.

Class Circle

circle_demo - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Data Tools Test Window Help

Start Page | Circle.cs | Program.cs

circle_demo.Circle | Area()

```
1  using System;
2
3  namespace circle_demo
4  {
5      class Circle
6      {
7          private String name;
8          private double radius = 0.0;
9
10         public Circle(String n, double r)
11         {
12             name = n;
13             radius = r;
14         }
15
16         public String Name() { return name; }
17         public double Radius() { return radius; }
18
19         public double Area()
20         {
21             return Math.PI * radius * radius;
22         }
23
```

New member

New constructor

Accessor Methods

Solution Explorer

Solution 'circle_demo' (1 project)

circle_demo

Properties

References

Circle.cs

Program.cs

Build succeeded | Ln 18 | Col 9 | Ch 9

Getting User Input

- What if we want the user to specify the radius of a Circle at run time?
  - Could overload the constructor and provide a version that asks for input.
  - Better to provide a separate function outside the class definition.
    - *Separate User Interface from class logic.*

- Let's write a function that asks the user for a name and a radius and creates a Circle of that radius with that name.

Getting User Input

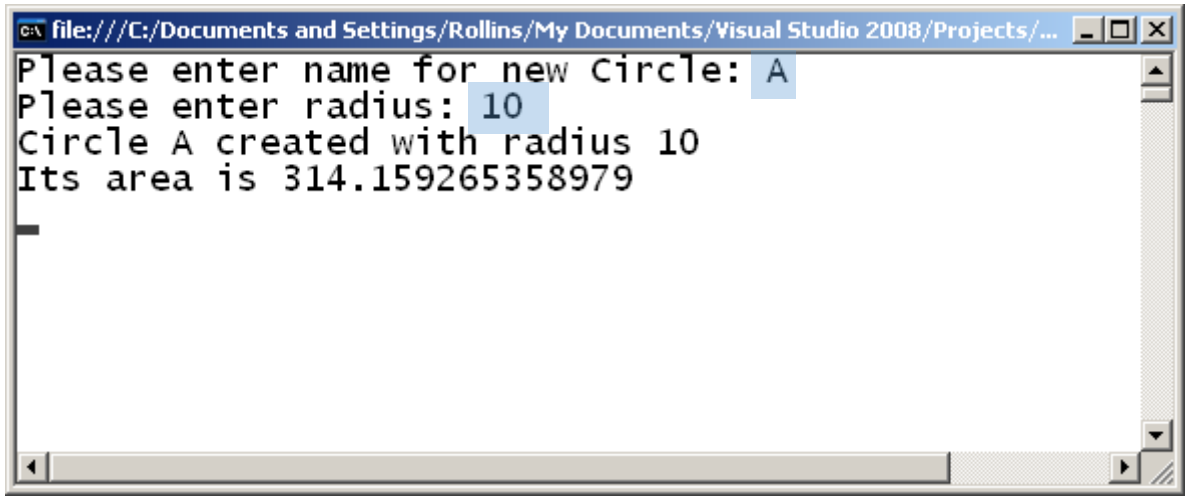
In class Program (along with Main())

```
static Circle Create_Circle()
{
    String name, temp;
    double radius;
    Console.Write("Please enter name for new Circle: ");
    name = Console.ReadLine();
    Console.Write("Please enter radius: ");
    temp = Console.ReadLine();
    radius = double.Parse(temp);
    return new Circle(name, radius);
}
```

Main()

```
static void Main(string[] args)
{
    Circle c1 = Create_Circle();
    Console.Write("Circle " + c1.Name());
    Console.WriteLine(" created with radius " + c1.Radius());
    Console.WriteLine("Its area is " + c1.Area());

    Console.ReadLine();  // Keep window open.
}
```

Running Program Circle



Passing Objects to a Function

- Let's extend class Circle with a method to compare one Circle to another.

- In class Circle ...

```
public bool Is_Greater_Than(Circle other)
{
    if (this.Radius() > other.Radius())
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Note keyword "this"

Call Radius() in Circle object passed as argument.

Using "Is_Greater_Than" Method

```
static void Main(string[] args)
{
    Circle Circle_A = Create_Circle();
    Console.Write ("Circle " + Circle_A.Name() );
    Console.WriteLine (" created with radius " + Circle_A.Radius());
    Console.WriteLine ("Its area is " + Circle_A.Area());

    Circle c2= Create_Circle();
    Console.Write ("Circle " + c2.Name() );
    Console.WriteLine (" created with radius " + c2.Radius());
    Console.WriteLine ("Its area is " + c2.Area());
```

Using "Is_Greater_Than" Method

```
if (c1.Is_Greater_Than(c2))
{
    Console.Write ("Circle " + c1.Name() + " is greater than ");
    Console.WriteLine( "Circle " + c2.Name());
}
else if (c2.Is_Greater_Than(c1))
{
    Console.Write ("Circle " + c2.Name() + " is greater than ");
    Console.WriteLine( "Circle " + c1.Name());
}
else
{
    Console.Write("Circle " + c1.Name() + " and Circle " + c2.Name());
    Console.WriteLine (" are the same size.");
}
```

Program Running

```
file:///C:/Documents and Settings/Rollins/My Documents/Visual Studio 2008/Projects/...
Please enter name for new Circle: A
Please enter radius: 5
Circle A created with radius 5
Its area is 78.5398163397448
Please enter name for new Circle: B
Please enter radius: 10
Circle B created with radius 10
Its area is 314.159265358979
Circle B is greater than Circle A
```

End of Section

Static Fields

Sometimes we need a single variable that is shared by all members of a class.

Declare the field *static*.

You can also declare a field *const* in order to ensure that it cannot be changed.

Not declared *static* – but *is* a static variable

- There is only one instance

Static Fields

```
class Math
{
    ...
    public const double PI = 3.14159265358979;
}
```

In class Circle --

```
public double Area()
{
  return Math.PI * radius * radius;
}
```

Class name rather than object name.

Static Methods

- Sometimes you want a *method* to be independent of a particular object.

- Consider class Math, which provides functions such as Sin, Cos, Sqrt, etc.

- These functions don't need any data from class Math. They just operate on values passed as arguments. So there is no reason to instantiate an object of class Math.

Static Methods

- Static methods are similar to functions in a procedural language.
  - The class just provides a home for the function.
  - Recall Main()
    - Starting point for every C# program
    - No object

Static Methods

Example:

```
class Math
{
 public static double Sqrt(double d)
 {
   ...
 }
 ...
}
```

Static Methods

To call a static method, you use the *class name* rather than an object name.

Example:

```
double d = Math.Sqrt(42.24);
```

Note: If the class has any nonstatic fields, a static method cannot refer to them.

Static Class

- A class that is intended to have only static members can be declared static.
- The compiler will ensure that no nonstatic members are ever added to the class.
  - Class cannot be instantiated.
- Math is a static class.
  - Book says otherwise on page 138. According to the VS2008 documentation this is incorrect.

End of Section

Partial Classes

- In C#, a class definition can be divided over multiple files:
  - Helpful for large classes with many methods.
  - Used by Microsoft in some cases to separate automatically generated code from user written code.

- If class defintion is divided over multiple files, each part is declared as a *partial* class.

Partial Classes

In file circ1.cs

```
partial class Circle
{
   // Part of class defintion
   ...
}
```

In file circ2.cs

```
partial class Circle
{
   // Another part of class definition
   ...
}
```