Introduction to Object Oriented Programming

Chapter 10

Objectives

You will be able to:

1. Write a simple class definition.
2. Control access to the methods and data in a class.
3. Create instances of a class.
4. Write and use class constructors.

What is a class?

Essentially a struct with built-in functions

```
class Circle
{
    double radius;

    // Constructor
    Circle(double r);

    // Return the area of this circle.
    double Area();
};
```

A class declaration

Encapsulation

- By default the class definition *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 class functions.
  - Called *methods*.

Class Members

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

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

As written, the previous class would be unusable, because all members are private by default.

Making Methods Visible

To make the methods visible outside we would write the class declaration as:

```
class Circle
{
private:
    double radius;

public:
    // Constructor
    Circle(double r);

    // Return the area of this circle.
    double Area();
};
```

File Circle.h

Constructor

The function

```
Circle(double r)
```

is called a *constructor*.

Invoked to initialize object when an object of this class is created.

Note: Name same as class name.

No return type

Implementing Member Functions

```
#include "Circle.h"

Circle::Circle(double r)
{
    radius = r;
}

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

File Circle.cpp

Creating Objects

- The class definition does not *do anything!*
  - Doesn't allocate memory.
  - Doesn't store data.
  - Doesn't perform actions.
- It just defines a type.
- To *do* anything, we have to create an *instance* of the class.
  - Call its methods to perform actions.

Objects

An instance of a class is called an *object*.

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

Each object has its own identity and its own lifetime.

Each object has its own copy of the data associated with the class.

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

The method sees the data in *that object*.

Using Classes and Objects

- Classes and objects are used much like traditional types and variables:
  - Local variables
    - `Circle c1(5);`
  - Can be member variables in other classes
  - Assignment

    ```
    c2 = c1;
    ```
  - Function arguments

    ```
    picture1.crop (c1);
    ```

Using Classes and Objects

- In C++ we can create objects by simply *declaring* them.
- Same as ints, doubles, etc.
  - Except that we may need to provide parameter values for the constructor.
- Lifetime of an object declared as a local variable is the same as the lifetime of the function invocation.

Example: Using Class Circle

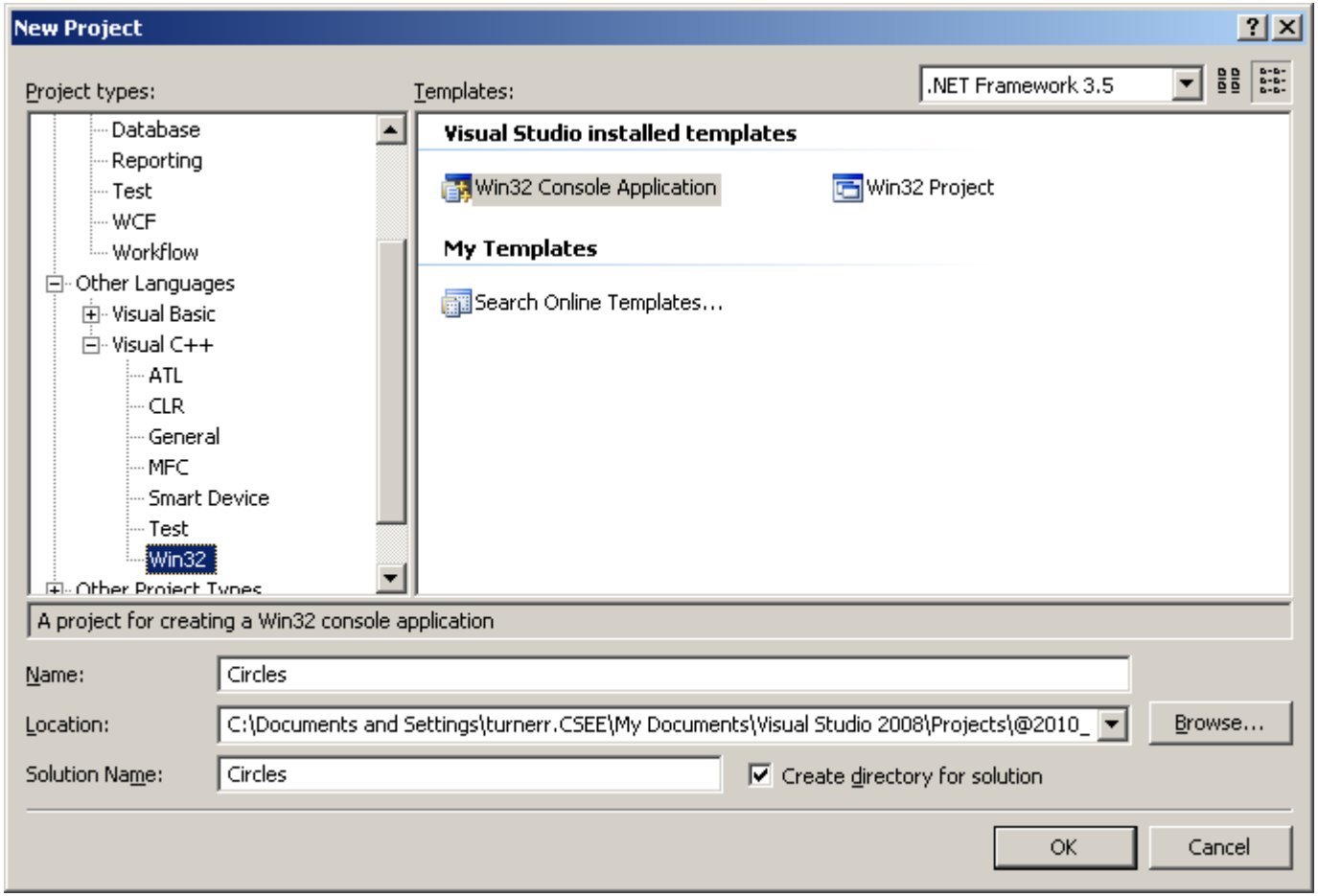
```
#include <iostream>
#include "Circle.h"

using namespace std;

int main()
{
    Circle c1(5);

    double c1_area = c1.Area();

    cout << "Area of circle c1 is " << c1_area << endl;

    cin.get();
    return 0;
}
```

File Circle_Program.cpp

Try it!

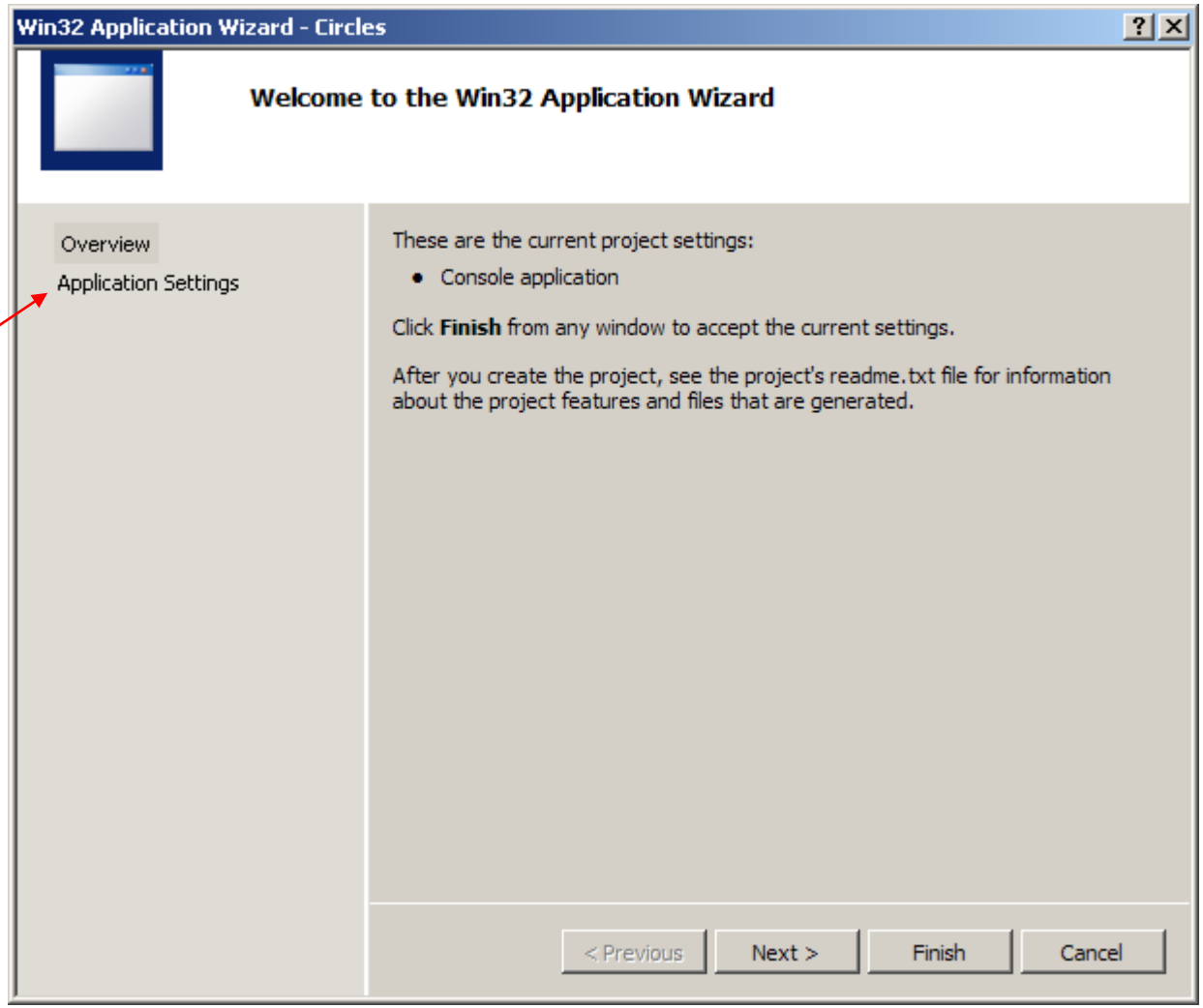
- Start up Visual Studio.
- Create a new project called "Circles"
  - Visual C++
  - Win32
  - Console Project

Creating the Project

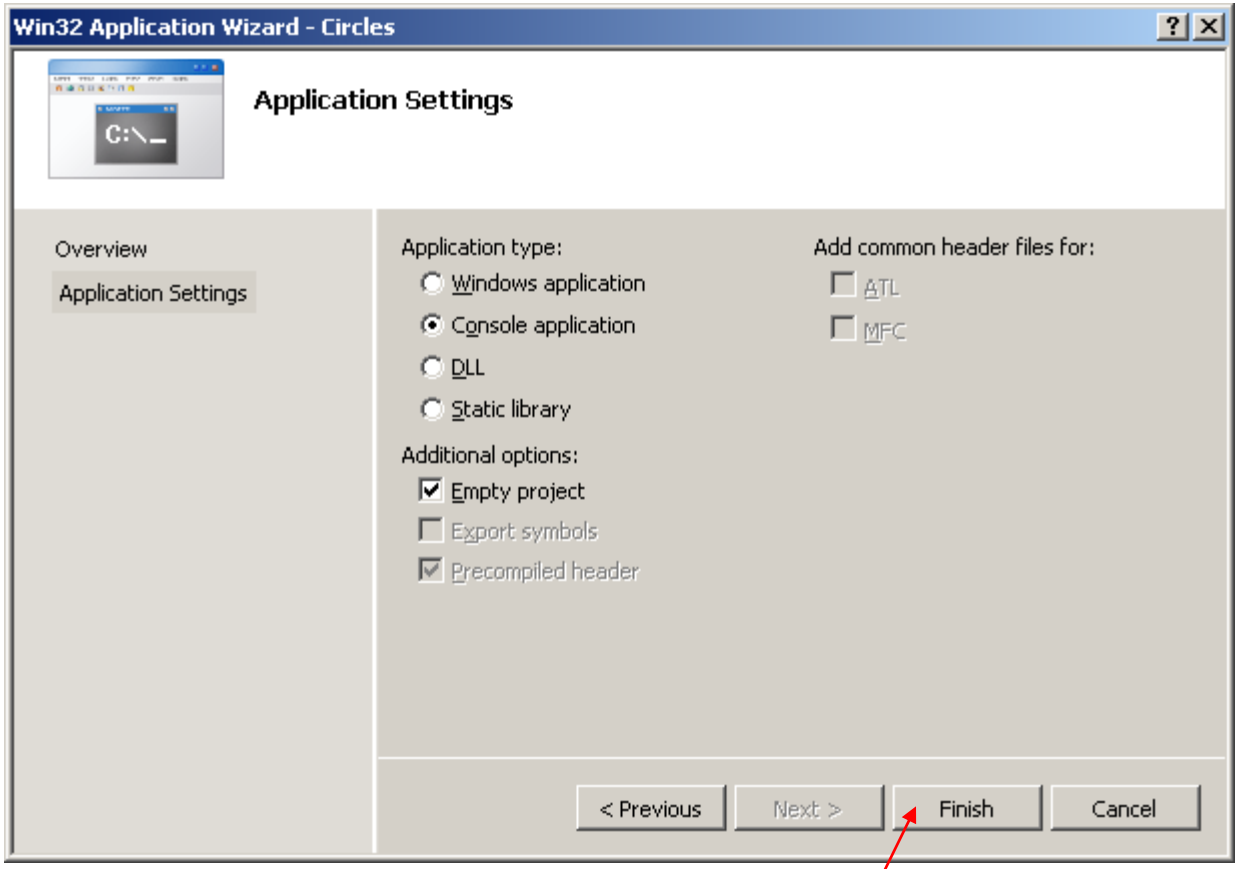


Creating the Project

Click here

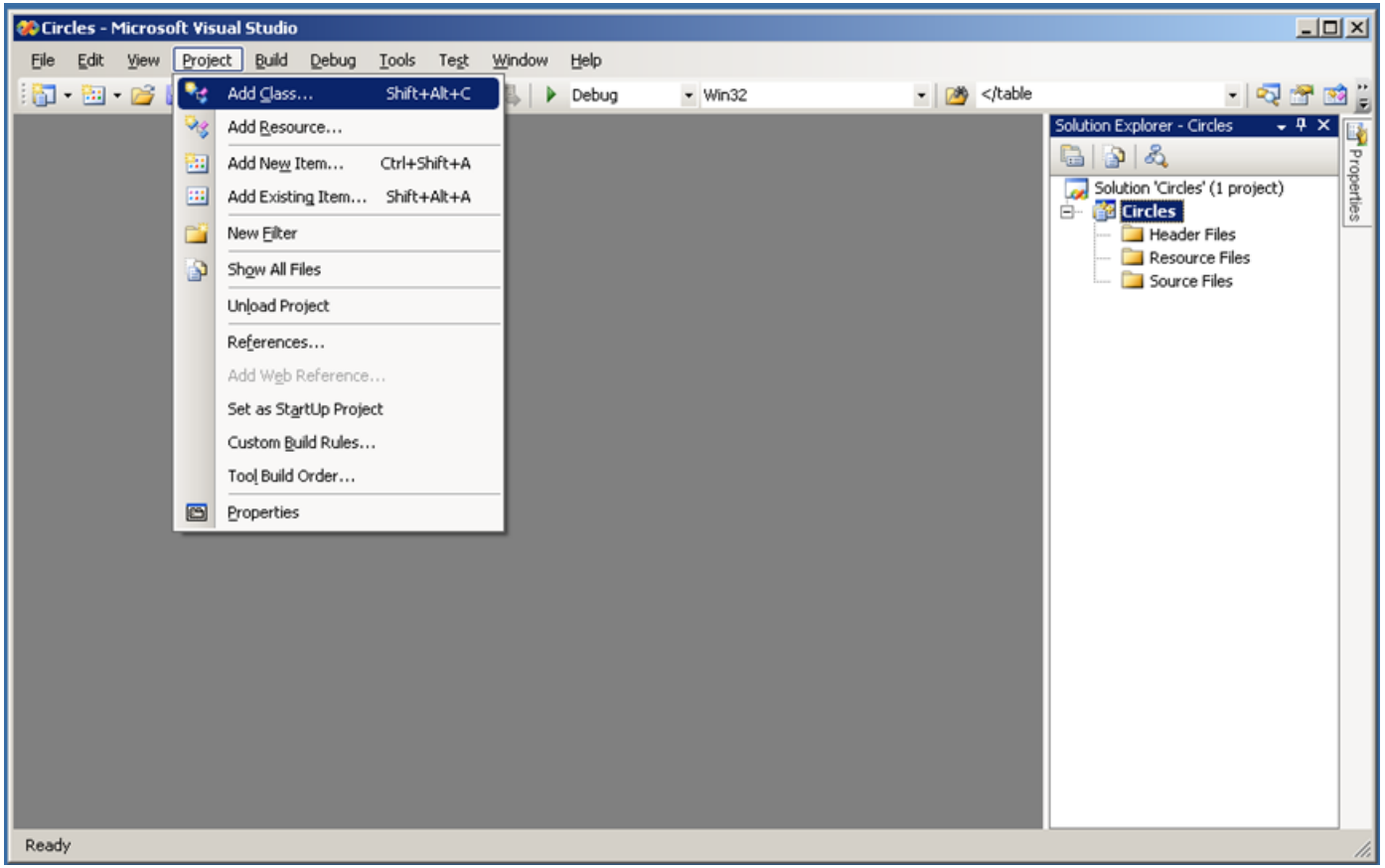


Start with an Empty Project

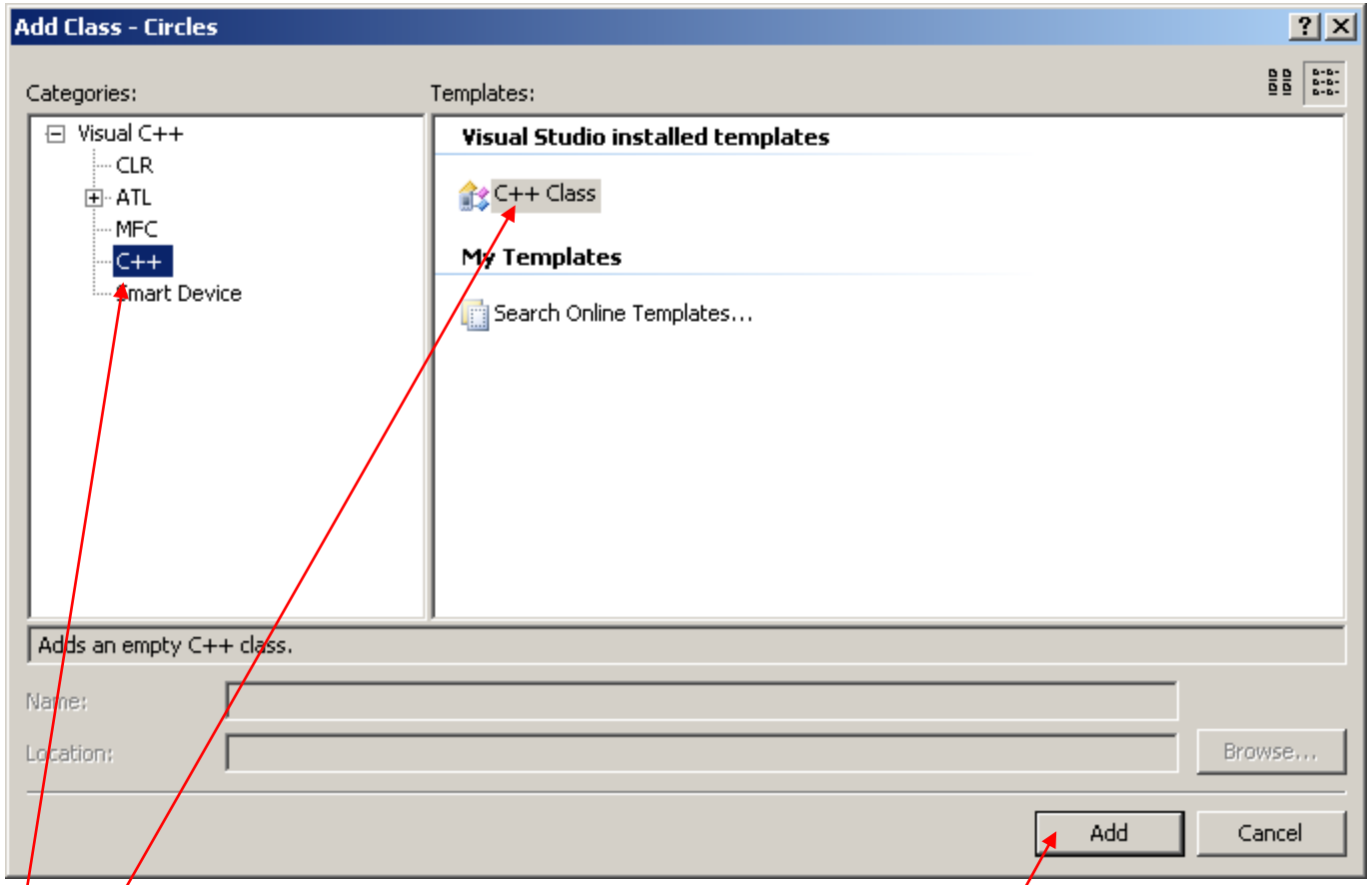


Click here

Add a Class to the Project



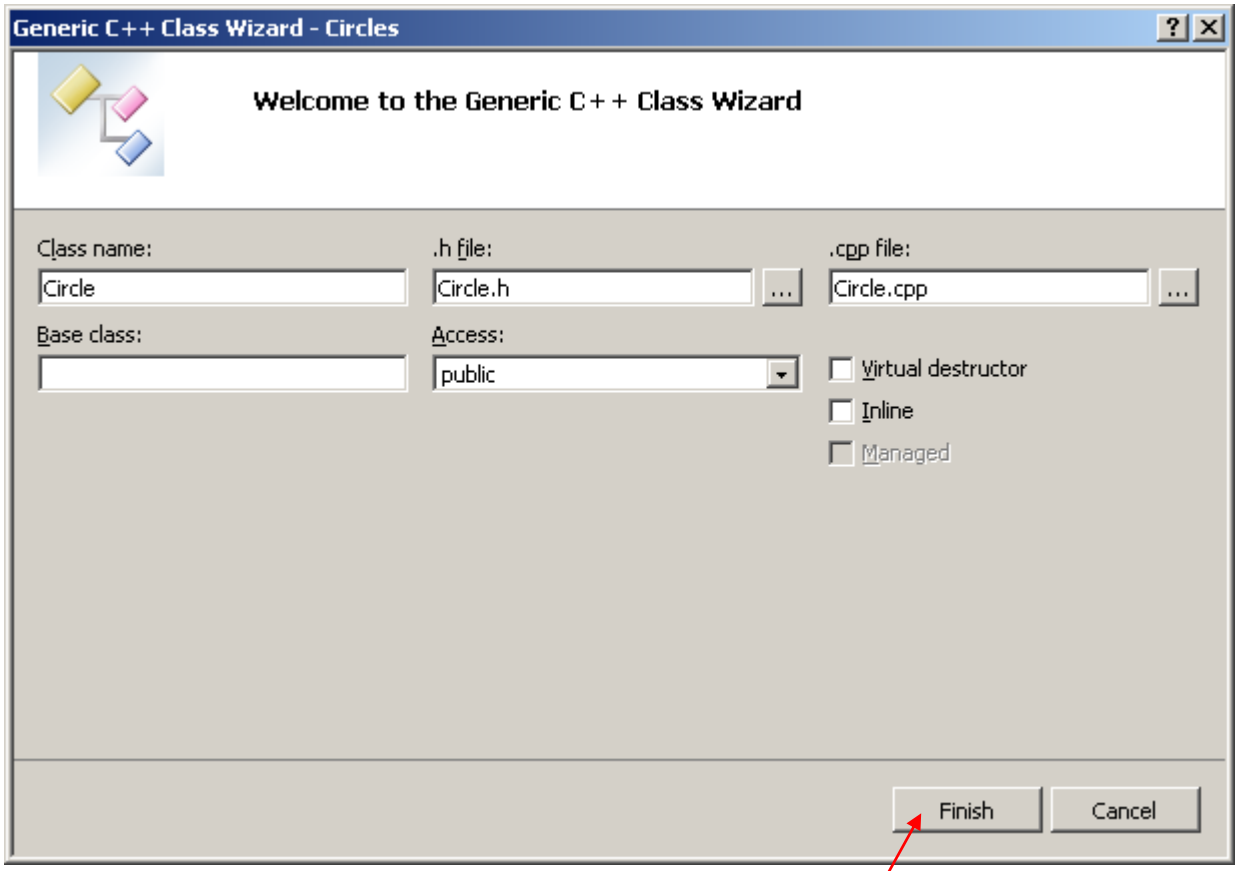
Add a Class to the Project



Click here

Be sure to select C++

Fill in the Class Name



Click here

Initial Class Files are Provided

Circles - Microsoft Visual Studio

File Edit View Project Build Debug Tools Window Community Help

Debug Win32

Circle.h* Output

Circle

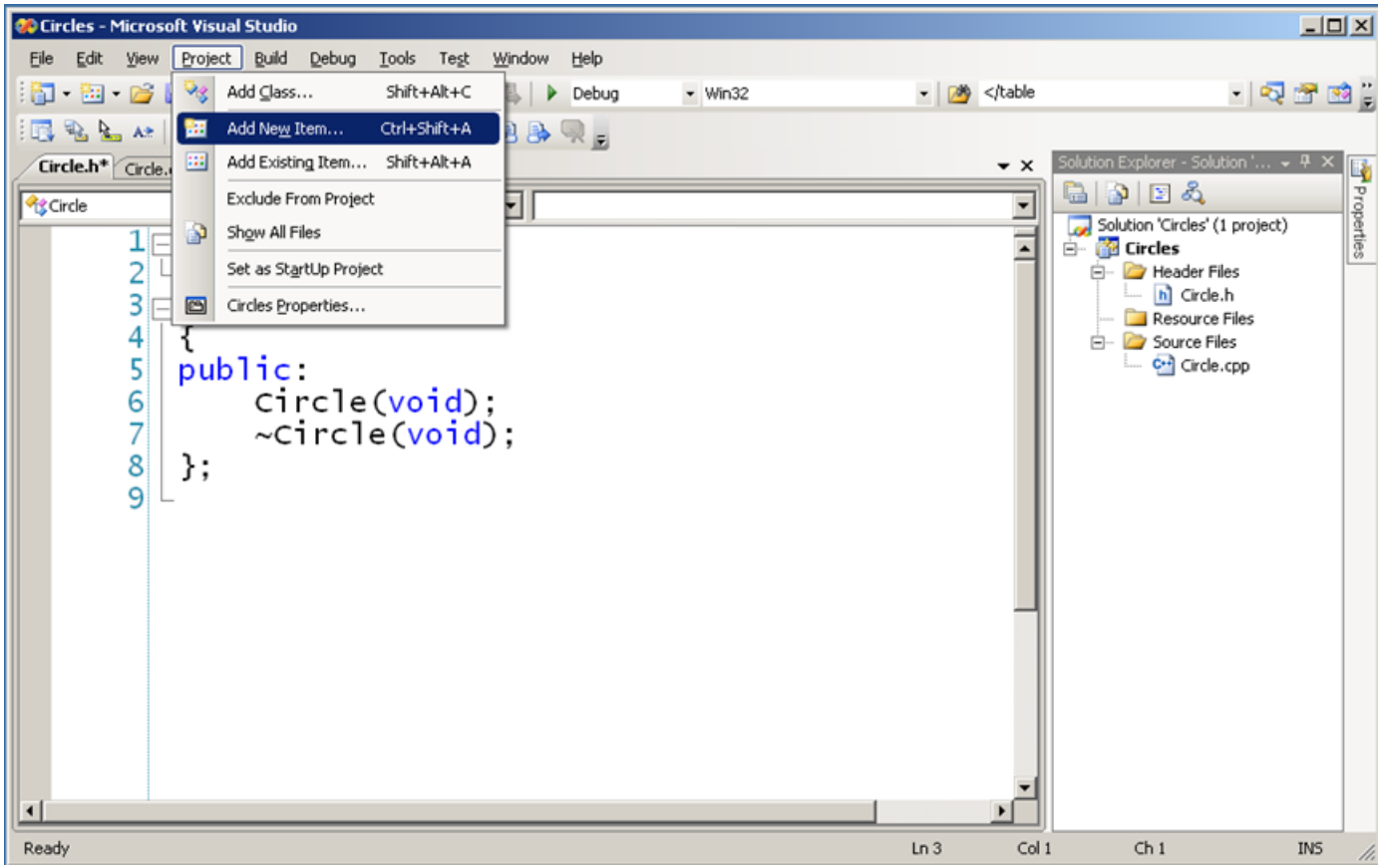
```
1  #pragma once
2
3  class Circle
4  {
5  public:
6      Circle(void);
7  public:
8      ~Circle(void);
9  };
10
```

Solution Explorer - Solution 'Circles' ...

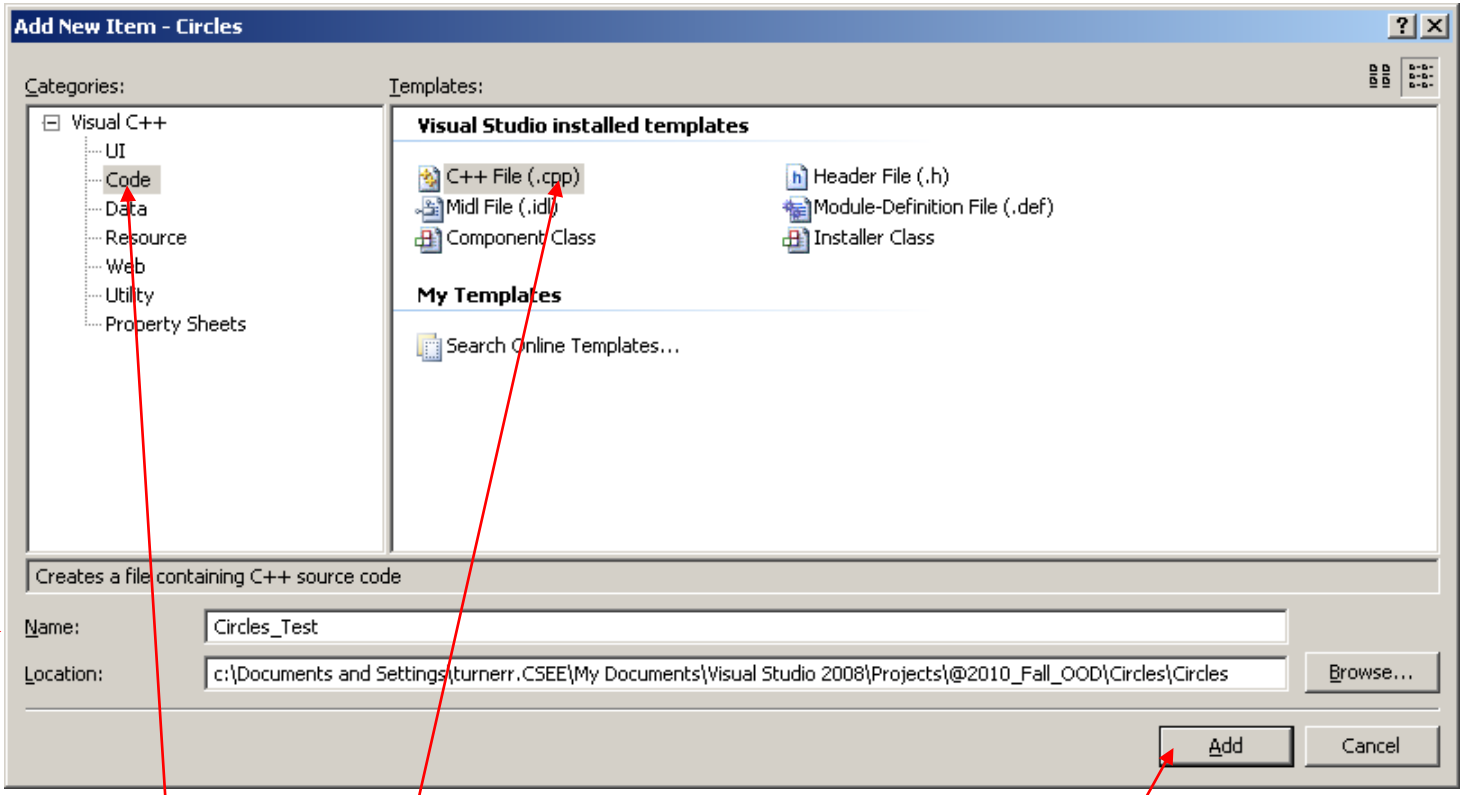
- Solution 'Circles' (1 project)
  - Circles
    - Header Files
      - Circle.h
    - Resource Files
    - Source Files
      - Circle.cpp

Ready Ln 3 Col 1 Ch 1 INS

Add a file for main()



Add a file for main()



Fill in the name

Click here

Be sure to select Code and C++

Start with a Stub

Circles - Microsoft Visual Studio

File Edit View Project Build Debug Tools Test Window Help

Debug Win32

Circles_Test.cpp* Circle.h Circle.cpp

(Global Scope) main()

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello\n";
6
7      std::cin.get();
8      return 0;
9  }
10
```

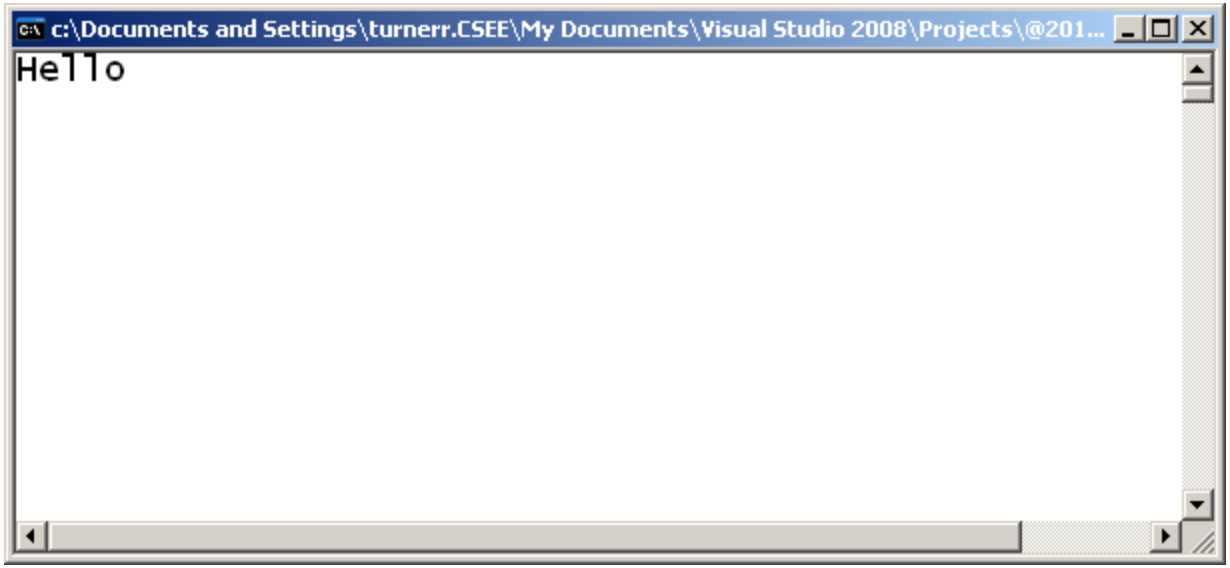
Solution Explorer - Circles

Solution 'Circles' (1 project)
- Circles
  - Header Files
    - Circle.h
  - Resource Files
  - Source Files
    - Circle.cpp
    - Circles_Test.cpp

Properties

Ready Ln 9 Col 2 Ch 2 INS

Build and Run



Class Circle

Circles - Microsoft Visual Studio

File Edit View Project Build Debug Tools Test Window Help

Circles_Test.cpp | Circle.h | Circle.cpp

Circle

```
1  #ifndef CIRCLE_H
2  #define CIRCLE_H
3
4  class Circle
5  {
6  private:
7      double radius;
8
9  public:
10     // Constructor
11     Circle(double r);
12
13     // Return the area of this circle.
14     double Area();
15 };
16
17 #endif
18
```

Solution Explorer - Circles

- Solution 'Circles' (1 project)
  - Circles
    - Header Files
      - Circle.h
    - Resource Files
    - Source Files
      - Circle.cpp
      - Circles_Test.cpp

Ready | Ln 4 | Col 13 | Ch 13 | INS

Implementation of Circle

Circles - Microsoft Visual Studio

File Edit View Project Build Debug Tools Test Window Help

Circles_Test.cpp | Circle.h | Circle.cpp

(Global Scope)

```
1  #include "Circle.h"
2
3  Circle::Circle(double r)
4  {
5      radius = r;
6  }
7
8  double Circle::Area()
9  {
10     return 3.141592 * radius * radius;
11 }
12
```

Solution Explorer - Solution '...

- Solution 'Circles' (1 project)
  - Circles
    - Header Files
      - Circle.h
    - Resource Files
    - Source Files
      - Circle.cpp
      - Circles_Test.cpp

Item(s) Saved | Ln 12 | Col 1 | Ch 1 | INS

The Main Program

Circles - Microsoft Visual Studio

File Edit View Project Build Debug Tools Test Window Help

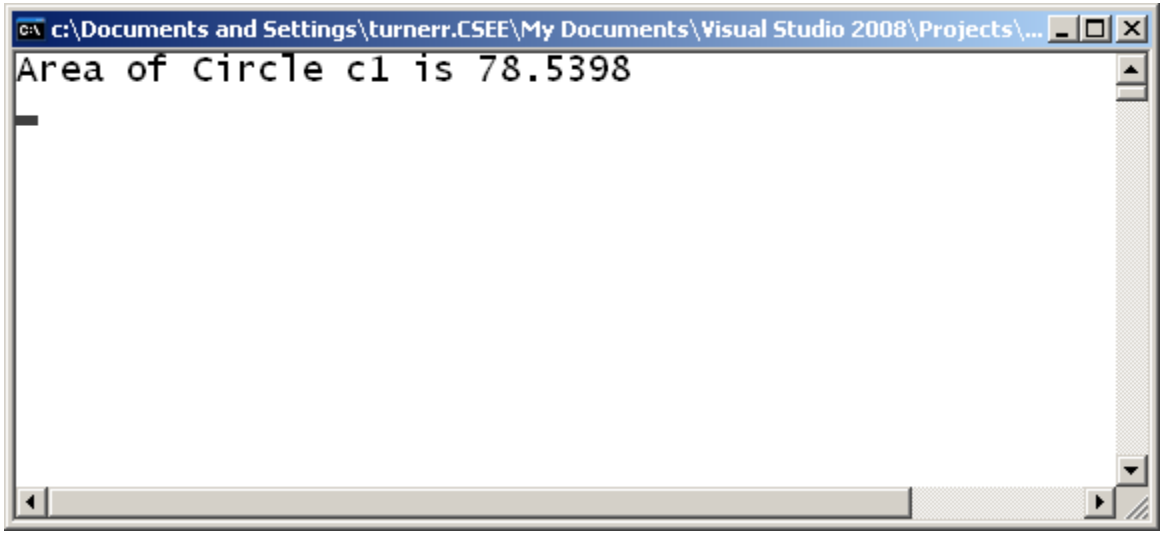
Circles_Test.cpp | Circle.h | Circle.cpp

(Global Scope) | main()

```
1  #include <iostream>
2  #include "Circle.h"
3
4  using namespace std;
5
6  int main()
7  {
8      Circle c1(5);
9
10     double c1_area = c1.Area();
11
12     cout << "Area of circle c1 is " << c1_area << endl;
13
14      std::cin.get();
15      return 0;
16 }
17
```

Ready | Ln 12 | Col 36 | Ch 36 | INS

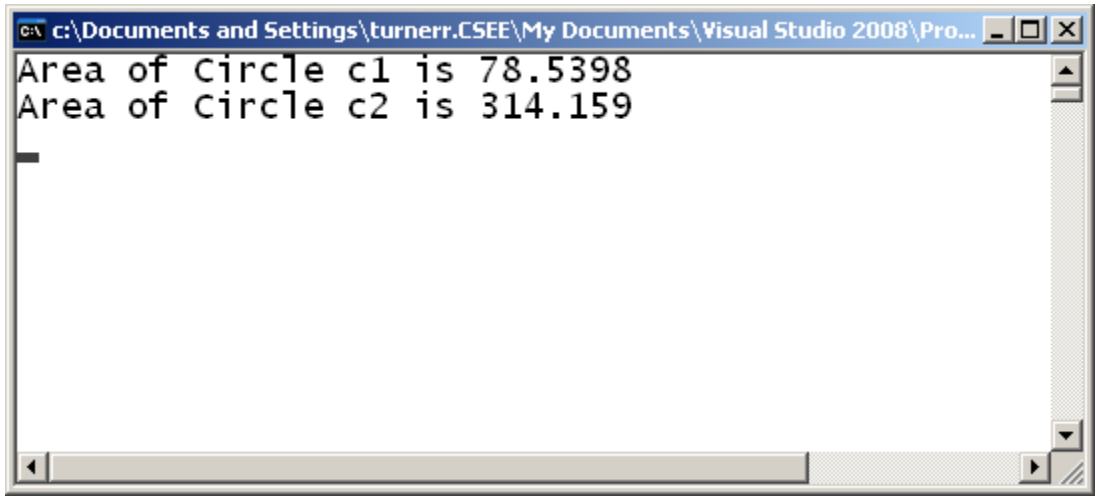
Build and Run



Example: Creating multiple objects of the same type

```
#include <iostream>
#include "Circle.h"

using namespace std;

int main()
{
    Circle c1(5);
    Circle c2(10);

    double c1_area = c1.Area();
    cout << "Area of circle c1 is " << c1_area << endl;

    cout << "Area of circle c2 is " << c2.Area() << endl;

    cin.get();          // Keep the window open.
    return 0;
}
```

Circles Program Running



Constructor

When function main() is entered

- The Circle objects, c1 and c2, are created.
  - Space allocated on the stack.
- The constructor is invoked for c1 with the argument 5.
  - Initializes the object's data
- The constructor is invoked for c2 with the argument 10.
  - Initializes the object's data

Multiple Constructors

A class can have any number of constructors.

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

This is called *overloading* a method.

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

Different *names* for parameters 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();
```

Essentially does nothing.

Class member variables will be uninitialized.

Destructor

- Each class can also have a special method called a *destructor*.
- Invoked when the object is deallocated.
- Free any resources used by the object.
- Name is class name preceded by ~
  - ~Circle
- No return value.
- No parameters.

Assignment

- Do today's examples for yourself
  if you have not done them in class.

- Read Chapter 10 through page 529.