Abstract Classes

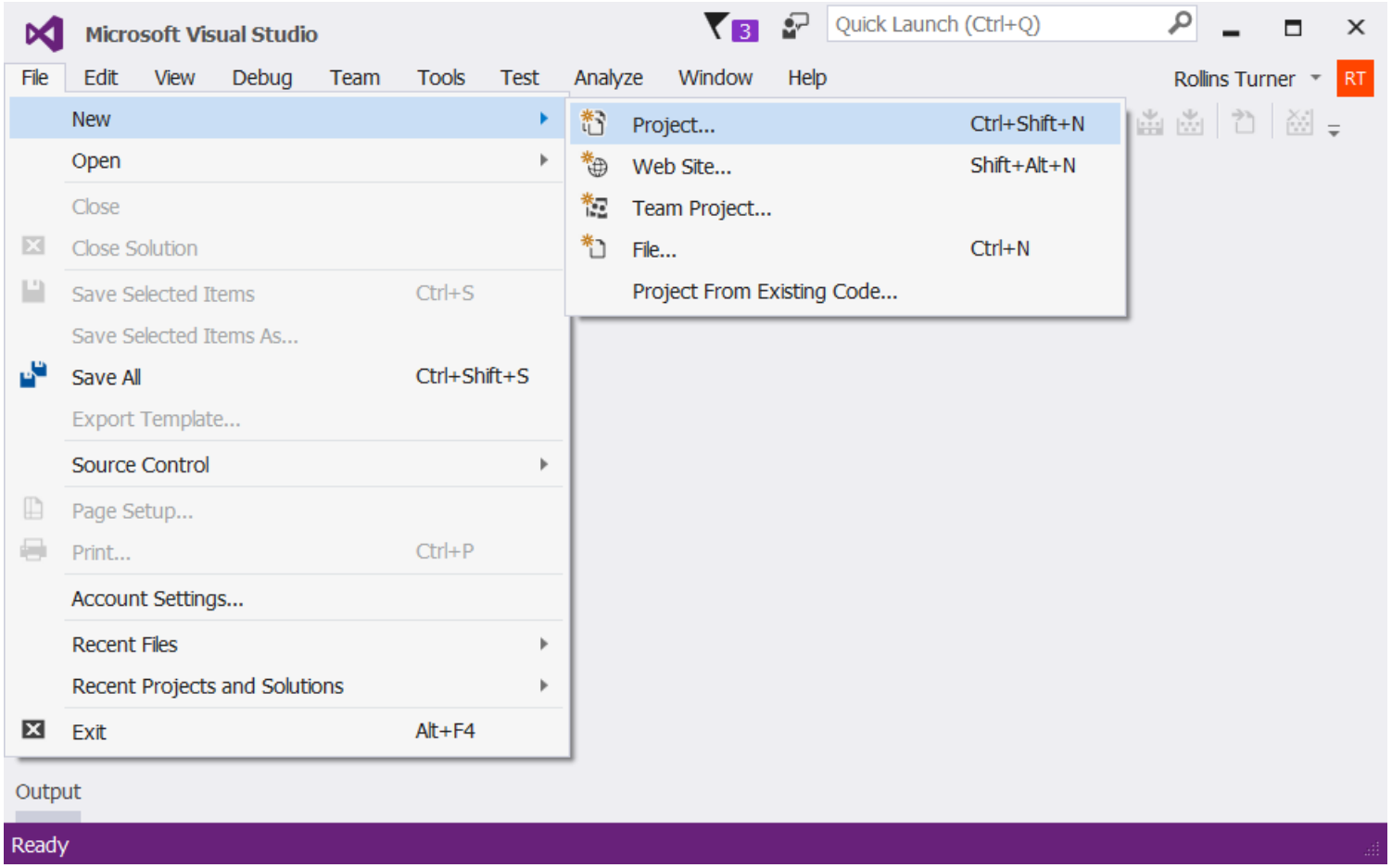
Objectives

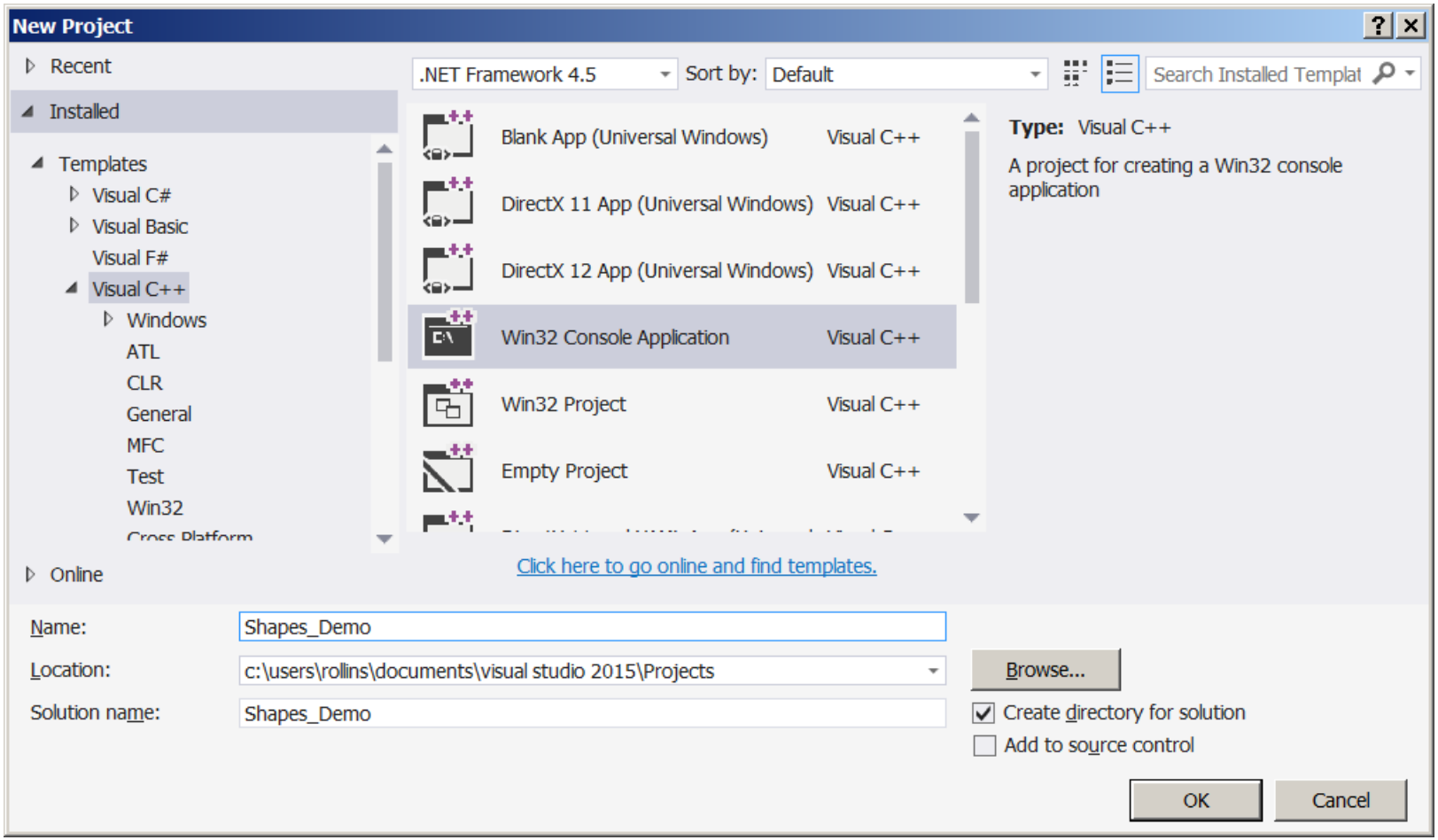
You will be able to:

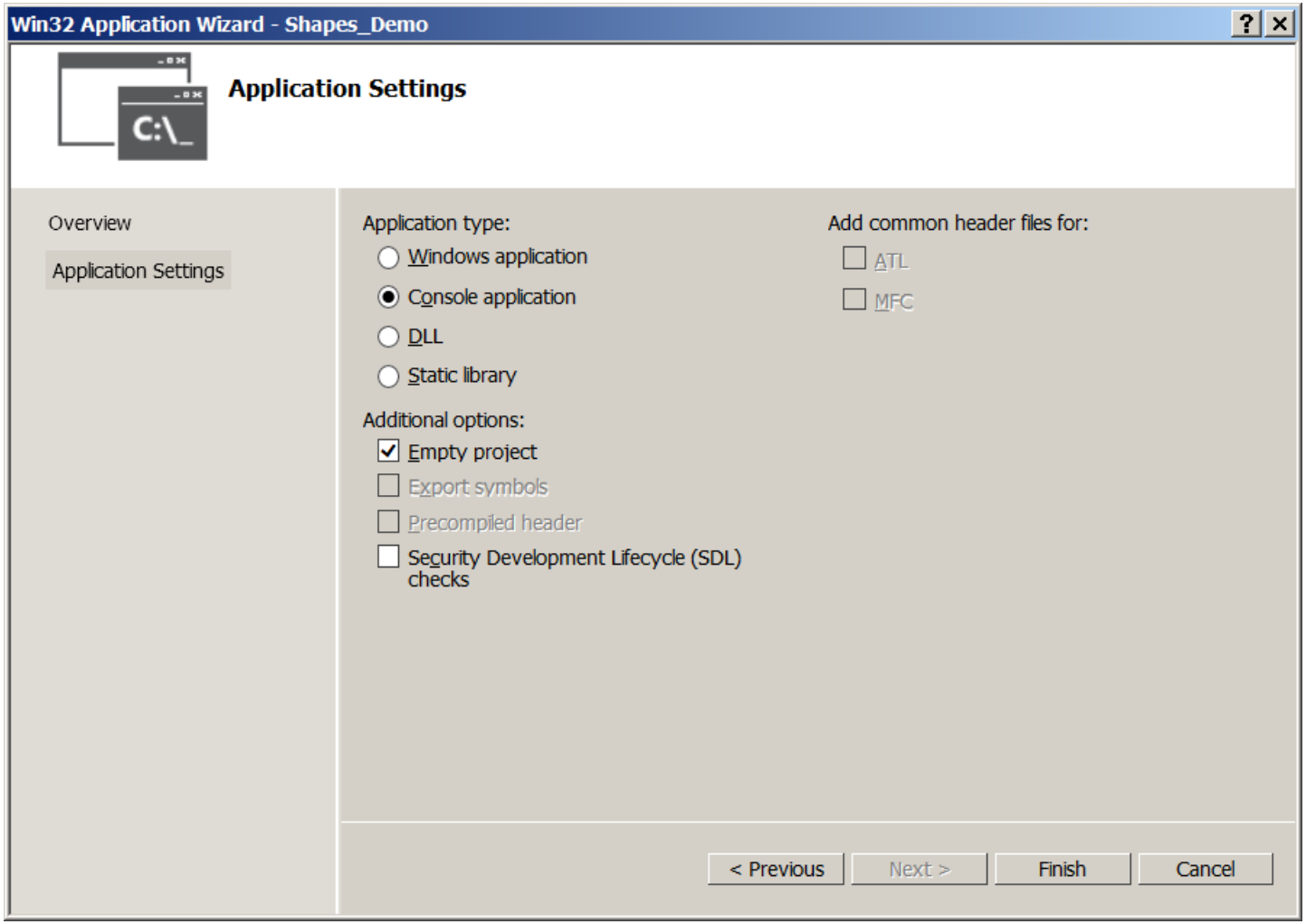
- Say what an abstract class is.
- Define and use abstract classes.

Getting Started

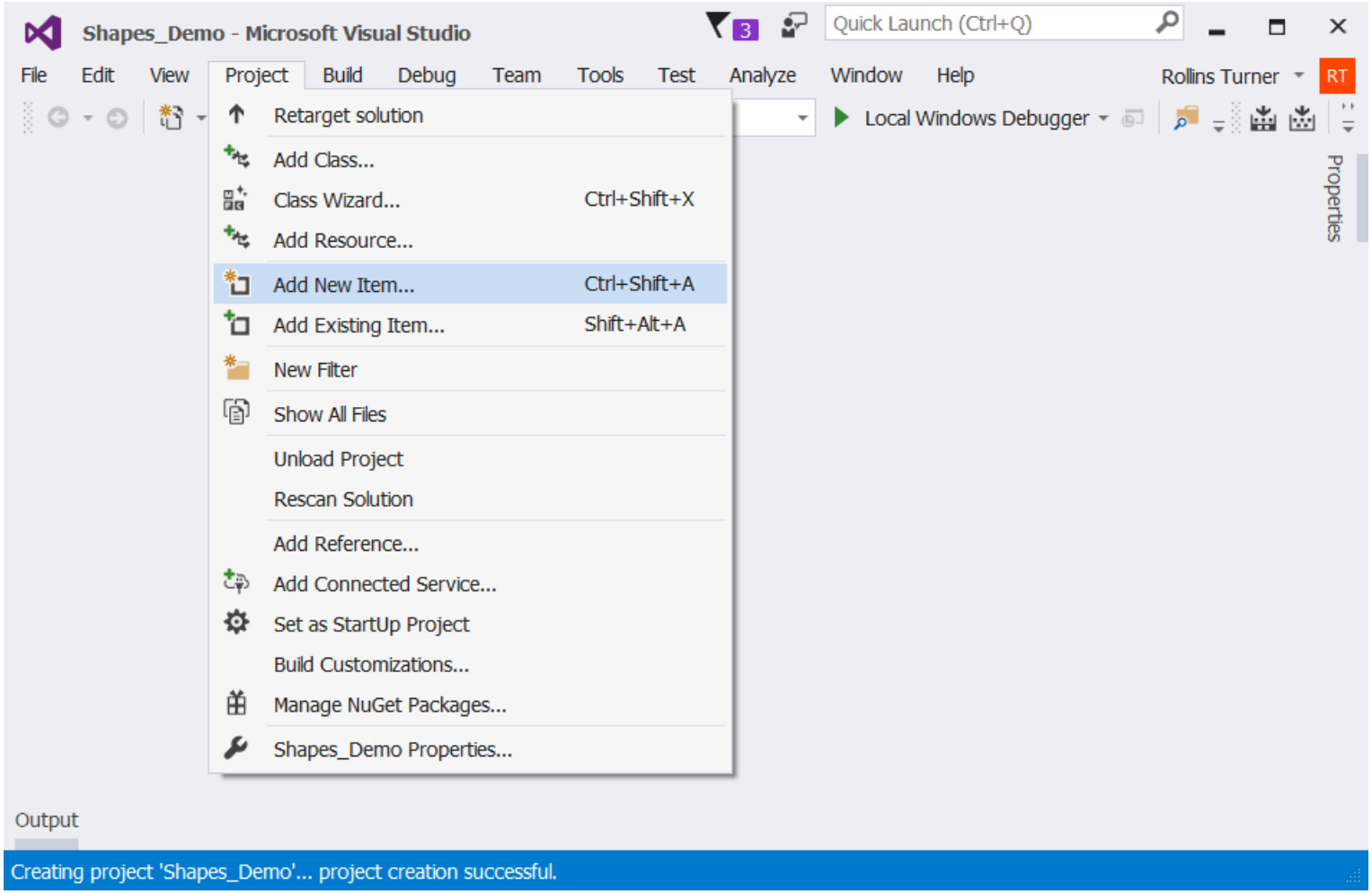
- Create a new C++ console project.



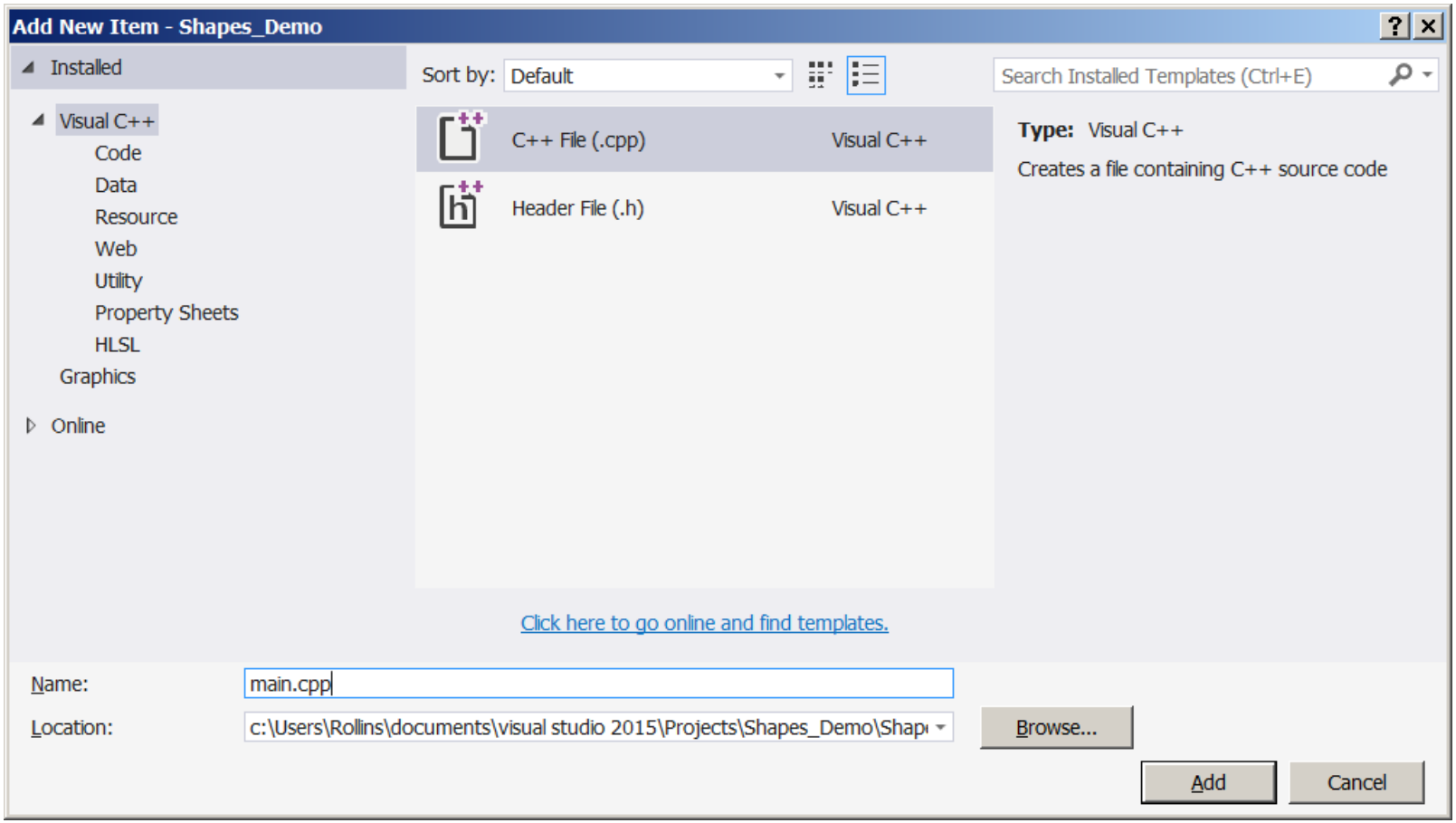

New Project
Recent
Installed
Templates
Visual C#
Visual Basic
Visual F#
Visual C++
Windows
ATL
CLR
General
MFC
Test
Win32
Online
.NET Framework 4.5
Sort by:
Default
Search Installed Templat
Blank App (Universal Windows) Visual C++
DirectX 11 App (Universal Windows) Visual C++
DirectX 12 App (Universal Windows) Visual C++
Win32 Console Application Visual C++
Win32 Project Visual C++
Empty Project Visual C++
Type: Visual C++
A project for creating a Win32 console application
Click here to go online and find templates.
Name:
Shapes_Demo
Location:
c:\users\rollins\documents\visual studio 2015\Projects
Browse...
Solution name:
Shapes_Demo
Create directory for solution
Add to source control
OK
Cancel


Win32 Application Wizard - Shapes_Demo
Application Settings
Overview
Application Settings
Application type:
Windows application
Console application
DLL
Static library
Additional options:
Empty project
Export symbols
Precompiled header
Security Development Lifecycle (SDL) checks
Add common header files for:
ATL
MFC
< Previous
Next >
Finish
Cancel

Add New Item


Shapes_Demo - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File
Edit
View
Project
Build
Debug
Team
Tools
Test
Analyze
Window
Help
Rollins Turner
RT
Local Windows Debugger
Properties
Retarget solution
Add Class...
Class Wizard... Ctrl+Shift+X
Add Resource...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
New Filter
Show All Files
Unload Project
Rescan Solution
Add Reference...
Add Connected Service...
Set as StartUp Project
Build Customizations...
Manage NuGet Packages...
Shapes_Demo Properties...
Output
Creating project 'Shapes_Demo'... project creation successful.

main.cpp


Add New Item - Shapes_Demo
Installed
Visual C++
Code
Data
Resource
Web
Utility
Property Sheets
HLSL
Graphics
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
C++ File (.cpp)
Visual C++
Header File (.h)
Visual C++
Type: Visual C++
Creates a file containing C++ source code
Click here to go online and find templates.
Name:
main.cpp
Location:
c:\Users\Rollins\documents\visual studio 2015\Projects\Shapes_Demo\Shap
Browse...
Add
Cancel

main.cpp

Shapes_Demo - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test Analyze Window Help

Rollins Turner RT

Debug x86

main.cpp*

Shapes_Demo (Global Scope) main()

```
#include <iostream>

using namespace std;

int main()
{
    cout << "This is the Shapes program\n";

    cout << "Normal termination\n";
    cin.get();
    cin.get();
    return 0;
}
```

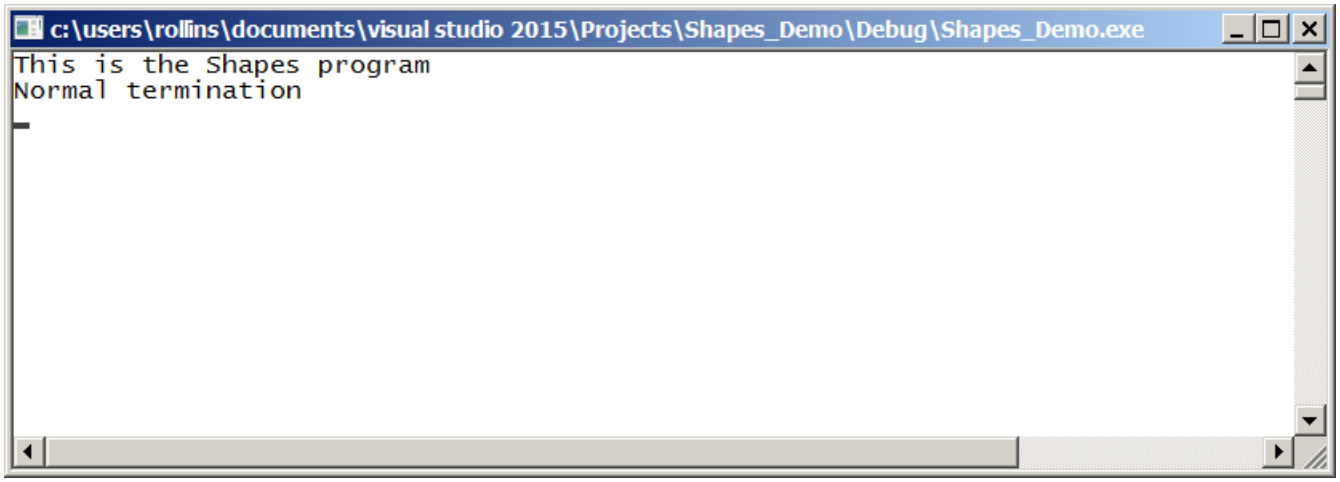
100 %

Output

Ready Ln 13 Col 2 Ch 2 INS

Build and run

Program Running



Add Class Point and Class Circle

Point.h

```
#pragma once
#include <iostream>

using std::ostream;

class Point
{
private:
    int x;
    int y;

public:
    Point(int X, int Y);
    friend ostream& operator<<(ostream& os, const Point& p);
};
```

Point.cpp

```
#include "Point.h"

using namespace std;

Point::Point(int X, int Y) : x(X), y(Y)
{}

ostream& operator<<(ostream& os, const Point& p)
{
    os << "(" << p.x << "," << p.y << ")";
    return os;
}
```

Circle.h

```
#pragma once
#include <iostream>
#include <string>
#include "Point.h"
using std::string;
using std::ostream;

class Circle
{
private:
    int radius;
    string name;
    Point center;
public:
    Circle(const string& Name, const Point& Center, int Radius);
    string Name() const {return name;};
    Point Center() const {return center;};
    int Radius() const {return radius;};
    double Area() const {return 3.14159 * radius;};
    bool operator<(const Circle& rhs) const {return Area() < rhs.Area();};
    void Display(ostream& os) const;
};
```

Circle.cpp

```
#include "Circle.h"

using namespace std;

Circle::Circle(const string& Name, const Point& Center, int Radius) :
    name(Name), center(Center), radius(Radius)
{}

void Circle::Display(ostream& os) const
{
    os << "Circle: " << name << ", "
       << "Center: " << center << ", "
       << "Radius: " << radius << endl;
}
```

main.cpp

```
#include <iostream>
#include "Point.h"
#include "Circle.h"
using namespace std;
int main()
{
    cout << "This is the Shapes program\n";
    Point pt(10, 12);

    Circle* c1 = new Circle("C1", pt, 6);
    c1->Display(cout);
    cout << "Area of " << c1->Name() << " is "
         << c1->Area() << endl;
    Circle* c2 = new Circle("C2", pt, 10);
    c2->Display(cout);
    cout << "Area of " << c1->Name() << " is "
         << c1->Area() << endl;
    if (*c1 < *c2)
    {
        cout << c1->Name() << " is smaller than " << c2->Name() << endl;
    }
    cout << "Normal termination\n";
    cin.get();
    cin.get();
    return 0;
}
```

Add to main.cpp

```
cout << "This is the Shapes program\n";
Point pt(10, 12);

Circle* c1 = new Circle("C1", pt, 6);
c1->Display(cout);
cout << "Area of " << c1->Name() << " is "
<< c1->Area() << endl;
Circle* c2 = new Circle("C2", pt, 10);
c2->Display(cout);
cout << "Area of " << c2->Name() << " is "
     << c2->Area() << endl;

if (*c1 < *c2)
{
    cout << c1->Name() << " is smaller than " << c2->Name();
}
else
{
    cout << c1->Name() << " is not smaller than " << c2->Name();
}
cout << endl << endl;
cout << "Normal termination\n";
```

Program Running

C:\Users\Rollins\Documents\Visual Studio 2015\Projects\Shapes_Demo\Debug\Shapes_Demo.exe

```
This is the Shapes program
Circle: C1, Center: (10,12), Radius: 6
Area of C1 is 18.8495
Circle: C2, Center: (10,12), Radius: 10
Area of C2 is 31.4159
C1 is smaller than C2
Normal termination
```

New Class

- Suppose we find that we need a Rectangle class.
- Add new class Rectangle to the project.

Rectangle.h

```
#pragma once
#include <string>
#include <iostream>
#include "Point.h"
using namespace std;

class Rectangle
{
private:
    string name;
    Point center;
    int width;
    int height;
public:
    Rectangle(string Name, const Point& Center, int W, int H);
    string Name() const {return name;};
    Point Center() const {return center;};
    int Width() const {return width;};
    int Height() const {return height;};
    double Area() const {return width*height;};
    bool operator<(const Rectangle& rhs) const
    {
        return Area() < rhs.Area();
    };
    void Display(ostream& os) const;
};
```

Rectangle.cpp

```
#include "Rectangle.h"

Rectangle::Rectangle(string Name, const Point& Center, int W, int H) :
    name(Name), center(Center), width(W), height(H)
{}

void Rectangle::Display(ostream& os) const
{
    os << "Rectangle: " << name << ", "
       << "Center: " << center << ", "
       << "Width: " << width << ", "
       << "Height: " << height << endl;
}
```

Add to main.cpp

```
#include "Rectangle.h"
...

Rectangle* r1 = new Rectangle("R1", pt, 3, 5);
r1->Display(cout);
cout << "Area of " << r1->Name() << " is "
     << r1->Area() << endl;

Rectangle* r2 = new Rectangle("R2", pt, 4, 4);
r2->Display(cout);
cout << "Area of " << r2->Name() << " is "
     << r2->Area() << endl;

if (*r1 < *r2)
{
    cout << r1->Name() << " is smaller than " << r2->Name() << endl;
}
else
{
    cout << r1->Name() << " is not smaller than " << r2->Name() << endl;
}
cout << endl;
```

Program Running

C:\Users\Rollins\Documents\Visual Studio 2015\Projects\Shapes_Demo\Debug\Shapes_Demo.exe

```
This is the Shapes program
Circle: C1, Center: (10,12), Radius: 6
Area of C1 is 18.8495
Circle: C2, Center: (10,12), Radius: 10
Area of C2 is 31.4159
C1 is smaller than C2


Rectangle: R1, Center: (10,12), Width: 3, Height: 5
Area of R1 is 15
Rectangle: R2, Center: (10,12), Width: 4, Height: 4
Area of R2 is 16
R1 is smaller than R2

Normal termination
```

Duplicated Code

- There is a lot of overlap between class Circle and class Rectangle.
  - `name`
  - `center`
  - `Name()`
  - `Center()`
  - `operator<()`

Duplicated Code

- Duplicated code is usually a bad idea!
- Let's factor out the common code as a base class and let Circle and Rectangle inherit it.
- Add class Shape.
- Modify class Circle and class Rectangle to inherit from class Shape

Class Shape

```
#pragma once
#include <iostream>
#include <string>
#include "Point.h"

using std::string;
using std:: ostream;

class Shape
{
protected:
    string name;
    Point center;

public:
    Shape(const string& Name, const Point& Center);
    string Name() const {return name;};
    Point Center() const {return center;};
    virtual double Area() const {return 0;};
    virtual bool operator<(const Shape& rhs) const
        {return Area() < rhs.Area();};
    virtual void Display(ostream& os) const;
};
```

Shape.cpp

```
#include "Shape.h"

using namespace std;

Shape::Shape(const string& Name, const Point& Center) :
    name(Name), center(Center)
{}

void Shape::Display(ostream& os) const
{
    os << "Shape: " << name << ", "
       << "Center: " << center << endl;
}
```

Circle.h

```
#pragma once
#include <iostream>
#include <string>
#include "Shape.h"
#include "Point.h"

using std::string;
using std:: ostream;

class Circle : public Shape
{
private:
    int radius;
    //string name;
    //Point center;

public:
    Circle(const string& Name, const Point& Center, int Radius);
    //string Name() const {return name;};
    //Point Center() const {return center;};
    int Radius() const {return radius;};
    double Area() const {return 3.14159 * radius;};
    //bool operator<(const Circle& rhs) const {return Area() < rhs.Area();};
    void Display(ostream& os) const;};
```

Circle.cpp

```
#include "Circle.h"
#include "Shape.h"

using namespace std;

Circle::Circle(const string& Name, const Point& Center, int Radius) :
    Shape(Name, Center), radius(Radius)
{}

void Circle::Display(ostream& os) const
{
    os << "Circle: " << name << ", "
       << "Center: " << center << ", "
       << "Radius: " << radius << endl;
}
```

Rectangle.h

```
#pragma once
#include <string>
#include <iostream>
#include "Point.h"
#include "Shape.h"
using namespace std;

class Rectangle : public Shape
{
private:
    //string name;
    //Point center;
    int width;
    int height;
public:
    Rectangle(string Name, Point Center, int W, int H);
    //string Name() const {return name;};
    //Point Center() const {return center;};
    int Width() const {return width;};
    int Height() const {return height;};

    double Area() const {return width*height;};
    //bool operator<(const Rectangle& rhs) const {return Area() < rhs.Area();};
    void Display(ostream& os) const;
};
```

Rectangle.cpp

```
#include "Shape.h"
#include "Rectangle.h"

using namespace std;

Rectangle::Rectangle(string Name, Point Center, int W, int H) :
    Shape(Name, Center), width(W), height(H)
{}


void Rectangle::Display(ostream& os) const
{
    os << "Rectangle: " << name << ", "
       << "Center: " << center << ", "
       << "Width: " << width << ", "
       << "Height: " << height << endl;
}
```

Build and Run

C:\Users\Rollins\Documents\Visual Studio 2015\Projects\Shapes_Demo\Debug\Shapes_Demo.exe

```
This is the Shapes program
Circle: C1, Center: (10,12), Radius: 6
Area of C1 is 18.8495
Circle: C2, Center: (10,12), Radius: 10
Area of C2 is 31.4159
C1 is smaller than C2


Rectangle: R1, Center: (10,12), Width: 3, Height: 5
Area of R1 is 15
Rectangle: R2, Center: (10,12), Width: 4, Height: 4
Area of R2 is 16
R1 is smaller than R2

Normal termination
```

Works the same.

Redundant code has been eliminated.
DRY!

Class Shape

- We can create objects of class Shape

  even though that really doesn't make sense!

```
Shape* s1 = new Shape("S1", pt);
s1->Display(cout);
cout << "Area of " << s1->Name() << " is "
     << s1->Area() << endl;
cout << endl;

Shape* s2 = new Shape("S2", pt);
s2->Display(cout);
cout << "Area of " << s2->Name() << " is "
     << s2->Area() << endl;
```

Program Running

C:\Users\Rollins\Documents\Visual Studio 2015\Projects\Shapes_Demo\Debug\Shapes_Demo.exe

```
This is the Shapes program
Circle: C1, Center: (10,12), Radius: 6
Area of C1 is 18.8495
Circle: C2, Center: (10,12), Radius: 10
Area of C2 is 31.4159
C1 is smaller than C2


Rectangle: R1, Center: (10,12), Width: 3, Height: 5
Area of R1 is 15
Rectangle: R2, Center: (10,12), Width: 4, Height: 4
Area of R2 is 16
R1 is smaller than R2

Shape: S1, Center: (10,12)
Area of S1 is 0
Shape: S2, Center: (10,12)
Area of S2 is 0
Normal termination
```

Abstract Classes

Key Concept 

- Class Shape isn't meant to be instantiated.
  - Exists only to serve as the base class for derived classes.
- Shape::Area() isn't meant to be invoked.
  - Exists only to be overriden in derived classes.
- We can express these ideas by making class Shape an *abstract class*.

Abstract Classes

- Redefine Shape::Area as:

```
virtual double Area() const = 0;
```

- The method is defined, but has no body.
- Called a *pure* virtual method.
  - Cannot be invoked.
  - Must be overridden in derived classes.
- Any class with one or more pure virtual methods is called an *abstract class*.
  - Cannot be instantiated.
  - Exists only to be a base class.

Compile Again

Error List

Entire Solution | 2 Errors | 0 Warnings | 0 Messages | Build + IntelliSense

Search Error List

	Code	Description	Project	File	Line	Supp...
	C2259	'Shape': cannot instantiate abstract class	Shapes_Demo	main.cpp	51	
	C2259	'Shape': cannot instantiate abstract class	Shapes_Demo	main.cpp	56	

Fix the Problems

- In main.cpp:
  - Comment out c1, c2, r1, r2

```
Shape* s1 = new Circle("S1", pt, 10);
s1->Display(cout);
cout << "Area of " << s1->Name() << " is "
     << s1->Area() << endl;
cout << endl;

Shape* s2 = new Rectangle("S2", pt, 3, 5);
s2->Display(cout);
cout << "Area of " << s2->Name() << " is "
     << s2->Area() << endl;
```

Updated Shapes Program Running

C:\Users\Rollins\Documents\Visual Studio 2015\Projects\Shapes_Demo\Debug\Shapes_Demo.exe

```
This is the Shapes program
Circle: S1, Center: (10,12), Radius: 10
Area of S1 is 31.4159

Rectangle: S2, Center: (10,12), Width: 3, Height: 5
Area of S2 is 15
Normal termination
```

Using the < operator

- We can even compare a Circle and a Rectangle as Shapes.

```
if (*s2 < *s1)
{
    cout << s2->Name() << " is smaller than " << s1->Name() << endl;
}
else
{
     cout << s2->Name() << " is not smaller than " << s1->Name() << endl;
}
```

Using the < operator

C:\Users\Rollins\Documents\Visual Studio 2015\Projects\Shapes_Demo\Debug\Shapes_D...

```
This is the Shapes program
Circle: S1, Center: (10,12), Radius: 10
Area of S1 is 31.4159

Rectangle: S2, Center: (10,12), Width: 3, Height: 5
Area of S2 is 15

S2 is smaller than S1
Normal termination
```

Summary

- To define a pure virtual method, write

  = 0;

  following the declaration in the class definition.

- Derived classes must provide a defintion in order to be instantiated.

- To make a class abstract, give the class at least one pure virtual function.
  - Then it can only be used as a base for derived classes.