Chapter 19 - C++ Inheritance

Outline

Objectives

- In this chapter, you will learn:
  - To be able to create new classes by inheriting from existing classes.
  - To understand how inheritance promotes software reusability.
  - To understand the notions of base classes and derived classes.

19.1 Introduction

- Inheritance
  - New classes created from existing classes
  - Absorb attributes and behaviors.
- Polymorphism
  - Write programs in a general fashion
  - Handle a wide variety of existing (and unspecified) related classes
- Derived class
  - Class that inherits data members and member functions from a previously defined base class

19.1 Introduction

- Inheritance
  - Single Inheritance
    - Class inherits from one base class
  - Multiple Inheritance
    - Class inherits from multiple base classes
  - Three types of inheritance:
    - `public`: Derived objects are accessible by the base class objects (focus of this chapter)
    - `private`: Derived objects are inaccessible by the base class
    - `protected`: Derived classes and `friend`s can access protected members of the base class

19.2 Base and Derived Classes

- Often an object from a derived class (subclass) “is an” object of a base class (superclass)

Base class	Derived classes
`Student`	`GraduateStudent` `UndergraduateStudent`

`Shape`	`Circle`
	`Triangle`
	`Rectangle`

`Loan`	`CarLoan`
	`HomeImprovementLoan`
	`MortgageLoan`

`Employee`	`FacultyMember` `StaffMember`
`Account`	`CheckingAccount` `SavingsAccount`

Fig. 19.1 Some simple inheritance examples.

19.2 Base and Derived Classes

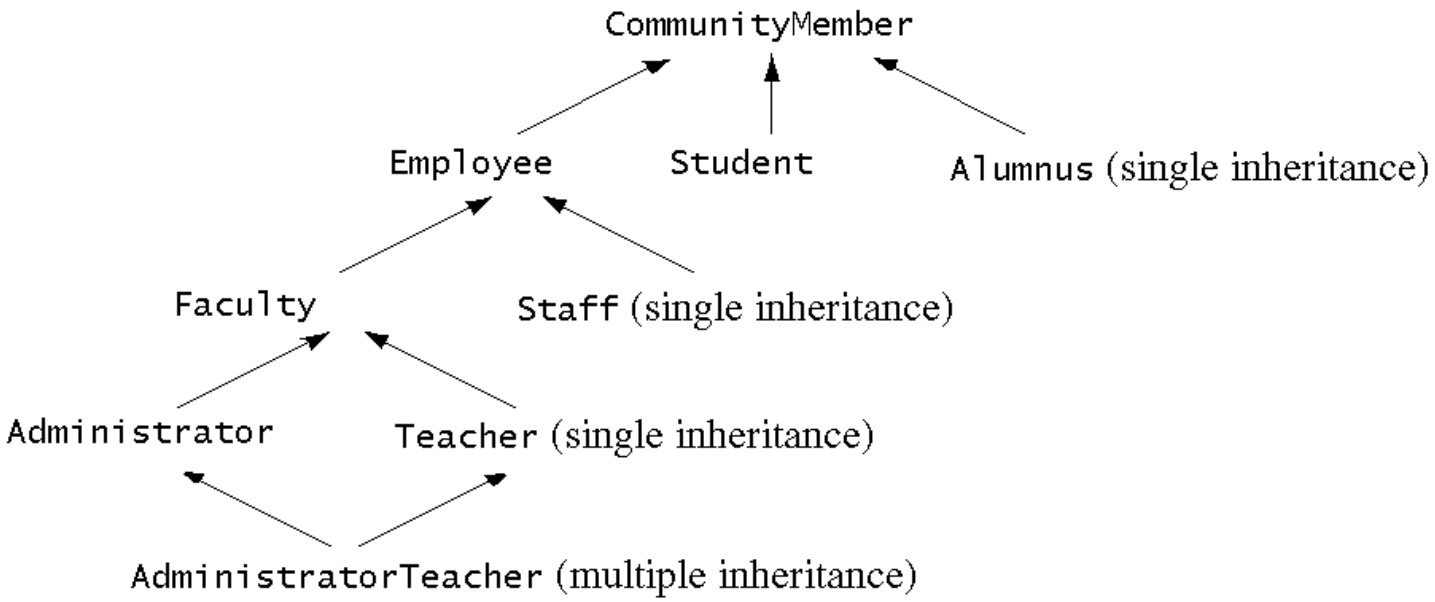


Fig. 19.2 An inheritance hierarchy for university community members.

19.2 Base and Derived Classes

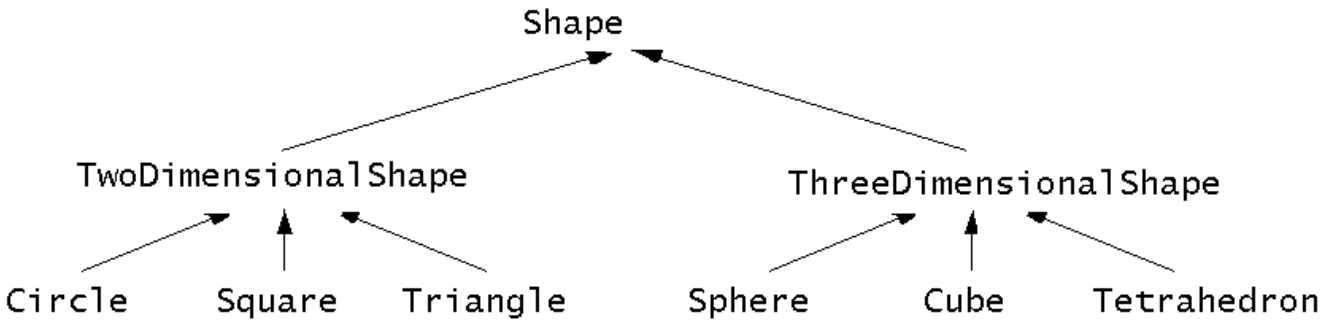


Fig. 19.3 A portion of a Shape class hierarchy.

19.2 Base and Derived Classes

- Implementation of `public` inheritance

```
class CommissionWorker : public Employee {
   ...
};
```

Class `CommissionWorker` inherits from class `Employee`

- `friend` functions not inherited
- `private` members of base class not accessible from derived class

19.3 Protected Members

- `protected` inheritance
  - Intermediate level of protection between `public` and `private` inheritance
  - Derived-class members can refer to `public` and `protected` members of the base class simply by using the member names
  - Note that `protected` data "breaks" encapsulation

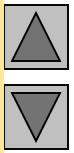
19.4 Casting Base Class Pointers to Derived Class Pointers

- Object of a derived class
  - Can be treated as an object of the base class
  - Reverse not true - base class objects not a derived-class object
- Downcasting a pointer
  - Use an explicit cast to convert a base-class pointer to a derived-class pointer
  - Be sure that the type of the pointer matches the type of object to which the pointer points

```
derivedPtr = static_cast< DerivedClass * > basePtr;
```


19.4 Casting Base-Class Pointers to Derived-Class Pointers

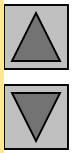
- Example
  - `Circle class` derived from the `Point` base `class`
  - We use pointer of type `Point` to reference a `Circle` object, and vice-versa



Outline

point.h

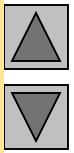
```
1    // Fig. 19.4: point.h
2    // Definition of class Point
3    #ifndef POINT_H
4    #define POINT_H
5
6    #include <iostream>
7
8    using std::ostream;
9
10   class Point {
11      friend ostream &operator<<( ostream &, const Point & );
12   public:
13      Point( int = 0, int = 0 );      // default constructor
14      void setPoint( int, int );      // set coordinates
15      int getX() const { return x; }  // get x coordinate
16      int getY() const { return y; }  // get y coordinate
17   protected:         // accessible by derived classes
18      int x, y;       // x and y coordinates of the Point
19   }; // end class Point
20
21   #endif
```



Outline

point.cpp

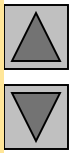
```
22  // Fig. 19.4: point.cpp
23  // Member functions for class Point
24  #include <iostream>
25  #include "point.h"
26
27  // Constructor for class Point
28  Point::Point( int a, int b ) { setPoint( a, b ); }
29
30  // Set x and y coordinates of Point
31  void Point::setPoint( int a, int b )
32  {
33     x = a;
34     y = b;
35  } // end function setPoint
36
37  // Output Point (with overloaded stream insertion operator)
38  ostream &operator<<( ostream &output, const Point &p )
39  {
40     output << '[' << p.x << ", " << p.y << ']';
41
42     return output;   // enables cascaded calls
43  } // end operator<< function
```



Outline

circle.h

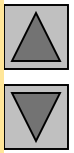
```
// Fig. 19.4: circle.h
// Definition of class Circle
#ifndef CIRCLE_H
#define CIRCLE_H

#include <iostream>

using std::ostream;

#include <iomanip>

using std::ios;
using std::setiosflags;
using std::setprecision;

#include "point.h"

class Circle : public Point {  // Circle inherits from Point
   friend ostream &operator<<( ostream &, const Circle & );
public:
   // default constructor
   Circle( double r = 0.0, int x = 0, int y = 0 );

   void setRadius( double );   // set radius
   double getRadius() const;   // return radius
   double area() const;        // calculate area
protected:
   double radius;
}; // end class Circle

#endif
```



Outline

circle.cpp

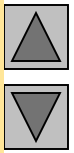
```
// Fig. 19.4: circle.cpp
// Member function definitions for class Circle
#include "circle.h"

// Constructor for Circle calls constructor for Point
// with a member initializer then initializes radius.
Circle::Circle( double r, int a, int b )
   : Point( a, b )       // call base-class constructor
{ setRadius( r ); }

// Set radius of Circle
void Circle::setRadius( double r )
   { radius = ( r >= 0 ? r : 0 ); }

// Get radius of Circle
double Circle::getRadius() const { return radius; }

// Calculate area of Circle
double Circle::area() const
   { return 3.14159 * radius * radius; }

// Output a Circle in the form:
// Center = [x, y]; Radius = #.##
ostream &operator<<( ostream &output, const Circle &c )
{
   output << "Center = " << static_cast< Point >( c )
          << "; Radius = "
          << setiosflags( ios::fixed | ios::showpoint )
          << setprecision( 2 ) << c.radius;

   return output;   // enables cascaded calls
} // end operator<< function
```



Outline

fig19_04.cpp (1 of 2)

```
// Fig. 19.4: fig19_04.cpp
// Casting base-class pointers to derived-class pointers
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

#include "point.h"
#include "circle.h"

int main()
{
   Point *pointPtr = 0, p( 30, 50 );
   Circle *circlePtr = 0, c( 2.7, 120, 89 );

   cout << "Point p: " << p << "\nCircle c: " << c << '\n';

   // Treat a Circle as a Point (see only the base class part)
   pointPtr = &c;   // assign address of Circle to pointPtr
   cout << "\nCircle c (via *pointPtr): "
        << *pointPtr << '\n';

```



Outline

fig19_04.cpp (2 of 2)

```
131     // Treat a Circle as a Circle (with some casting)
132     // cast base-class pointer to derived-class pointer
133     circlePtr = static_cast< Circle * >( pointPtr );
134     cout << "\nCircle c (via *circlePtr):\n" << *circlePtr
135          << "\nArea of c (via circlePtr): "
136          << circlePtr->area() << '\n';
137
138     // DANGEROUS: Treat a Point as a Circle
139     pointPtr = &p;   // assign address of Point to pointPtr
140
141     // cast base-class pointer to derived-class pointer
142     circlePtr = static_cast< Circle * >( pointPtr );
143     cout << "\nPoint p (via *circlePtr):\n" << *circlePtr
144          << "\nArea of object circlePtr points to: "
145          << circlePtr->area() << endl;
146     return 0;
147  } // end function main
```

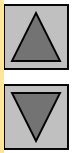
```
Point p: [30, 50]
Circle c: Center = [120, 89]; Radius = 2.70

Circle c (via *pointPtr): [120, 89]

Circle c (via *circlePtr):
Center = [120, 89]; Radius = 2.70
Area of c (via circlePtr): 22.90

Point p (via *circlePtr):
Center = [30, 50]; Radius = 0.00
Area of object circlePtr points to: 0.00
```

19.5 Using Member Functions

- Derived class
  - Cannot directly access `private` members of its base class
  - Hiding `private` members is a huge help in testing, debugging and correctly modifying systems

19.6 Overriding Base-Class Members in a Derived Class

- To override a base-class member function
  - In derived class, supply new version of that function
    - Same function name, different definition
  - The scope-resolution operator may be used to access the base class version from the derived class



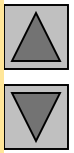
Outline

employ.h

```
1   // Fig. 19.5: employ.h
2   // Definition of class Employee
3   #ifndef EMPLOY_H
4   #define EMPLOY_H
5
6   class Employee {
7   public:
8      Employee( const char *, const char * );  // constructor
9      void print() const;  // output first and last name
10     ~Employee();         // destructor
11  private:
12     char *firstName;     // dynamically allocated string
13     char *lastName;      // dynamically allocated string
14  }; // end class Employee
15
16  #endif
```

employ.cpp (1 of 2)

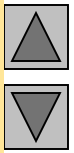
```
17  // Fig. 19.5: employ.cpp
18  // Member function definitions for class Employee
19  #include <iostream>
20
21  using std::cout;
22
23  #include <cstring>
24  #include <cassert>
25  #include "employ.h"
26
```



Outline

employ.cpp (2 of 2)

```
27  // Constructor dynamically allocates space for the
28  // first and last name and uses strcpy to copy
29  // the first and last names into the object.
30  Employee::Employee( const char *first, const char *last )
31  {
32     firstName = new char[ strlen( first ) + 1 ];
33     assert( firstName != 0 ); // terminate if not allocated
34     strcpy( firstName, first );
35
36     lastName = new char[ strlen( last ) + 1 ];
37     assert( lastName != 0 );  // terminate if not allocated
38     strcpy( lastName, last );
39  } // end Employee constructor
40
41  // Output employee name
42  void Employee::print() const
43     { cout << firstName << ' ' << lastName; }
44
45  // Destructor deallocates dynamically allocated memory
46  Employee::~Employee()
47  {
48     delete [] firstName;   // reclaim dynamic memory
49     delete [] lastName;    // reclaim dynamic memory
50  } // end Employee destructor
```



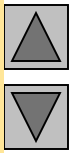
Outline

hourly.h

```
51  // Fig. 19.5: hourly.h
52  // Definition of class HourlyWorker
53  #ifndef HOURLY_H
54  #define HOURLY_H
55
56  #include "employ.h"
57
58  class HourlyWorker : public Employee {
59  public:
60     HourlyWorker( const char*, const char*, double, double );
61     double getPay() const;  // calculate and return salary
62     void print() const;     // overridden base-class print
63  private:
64     double wage;            // wage per hour
65     double hours;           // hours worked for week
66  }; // end class HourlyWorker
67
68  #endif
```

hourly.cpp (1 of 2)

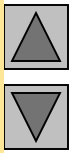
```
69  // Fig. 19.5: hourly.cpp
70  // Member function definitions for class HourlyWorker
71  #include <iostream>
72
73  using std::cout;
74  using std::endl;
75
76  #include <iomanip>
77
```



Outline

hourly.cpp (2 of 2)

```
78   using std::ios;
79   using std::setiosflags;
80   using std::setprecision;
81
82   #include "hourly.h"
83
84   // Constructor for class HourlyWorker
85   HourlyWorker::HourlyWorker( const char *first,
86                               const char *last,
87                               double initHours, double initWage )
88      : Employee( first, last )   // call base-class constructor
89   {
90      hours = initHours;  // should validate
91      wage = initWage;    // should validate
92   } // end HourlyWorker constructor
93
94   // Get the HourlyWorker's pay
95   double HourlyWorker::getPay() const { return wage * hours; }
96
97   // Print the HourlyWorker's name and pay
98   void HourlyWorker::print() const
99   {
100     cout << "HourlyWorker::print() is executing\n\n";
101     Employee::print();   // call base-class print function
102
103     cout << " is an hourly worker with pay of $"
104          << setiosflags( ios::fixed | ios::showpoint )
105          << setprecision( 2 ) << getPay() << endl;
106  } // end function print
```



Outline

fig19_05.cpp

```
// Fig. 19.5: fig19_05.cpp
// Overriding a base-class member function in a
// derived class.
#include "hourly.h"

int main()
{
   HourlyWorker h( "Bob", "Smith", 40.0, 10.00 );
   h.print();
   return 0;
} // end function main
```

```
HourlyWorker::print() is executing

Bob Smith is an hourly worker with pay of $400.00
```

19.7 Public, Private, and Protected Inheritance

Base class member access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by any non-`static` member functions, `friend` functions and non-member functions.	`protected` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.	`private` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.	`private` in derived class. Can be accessed directly by all non-`static` member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by non-`static` member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by non-`static` member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by non-`static` member functions and `friend` functions through `public` or `protected` member functions of the base class.

Fig. 19.6 Summary of base-class member accessibility in a derived class.

19.8 Direct and Indirect Base Classes

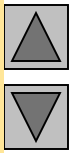
- Direct base class
  - Explicitly listed derived class' header with the colon (`:`) notation when that derived class is declared.
  - **`class HourlyWorker : public Employee`**
    - **`Employee`** is a direct base class of **`HourlyWorker`**
- Indirect base class
  - Inherited from two or more levels up the class hierarchy
  - **`class MinuteWorker : public HourlyWorker`**
    - **`Employee`** is an indirect base class of **`MinuteWorker`**

19.9 Using Constructors and Destructors in Derived Classes

- Base class initializer
  - Uses member-initializer syntax
  - Can be provided in the derived class constructor to call the base-class constructor explicitly
    - Otherwise base class' default constructor called implicitly
  - Base-class constructors and base-class assignment operators are not inherited by derived classes
    - However, derived-class constructors and assignment operators can call still them

19.9 Using Constructors and Destructors in Derived Classes

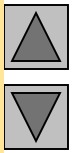
- Derived-class constructor
  - Calls the constructor for its base class first to initialize its base-class members
  - If the derived-class constructor is omitted, its default constructor calls the base-class' default constructor
- Destructors are called in the reverse order of constructor calls.
  - Derived-class destructor is called before its base-class destructor



Outline

point2.h

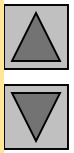
```
1    // Fig. 19.7: point2.h
2    // Definition of class Point
3    #ifndef POINT2_H
4    #define POINT2_H
5
6    class Point {
7    public:
8       Point( int = 0, int = 0 );  // default constructor
9       ~Point();    // destructor
10   protected:      // accessible by derived classes
11      int x, y;    // x and y coordinates of Point
12   }; // end class Point
13
14   #endif
```



Outline

point2.cpp

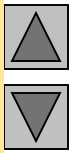
```
15  // Fig. 19.7: point2.cpp
16  // Member function definitions for class Point
17  #include <iostream>
18
19  using std::cout;
20  using std::endl;
21
22  #include "point2.h"
23
24  // Constructor for class Point
25  Point::Point( int a, int b )
26  {
27     x = a;
28     y = b;
29
30     cout << "Point  constructor: "
31          << '[' << x << ", " << y << ']' << endl;
32  } // end Point constructor
33
34  // Destructor for class Point
35  Point::~Point()
36  {
37     cout << "Point  destructor:  "
38          << '[' << x << ", " << y << ']' << endl;
39  } // end Point destructor
```



Outline

circle2.h

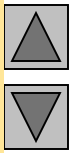
```
40  // Fig. 19.7: circle2.h
41  // Definition of class Circle
42  #ifndef CIRCLE2_H
43  #define CIRCLE2_H
44
45  #include "point2.h"
46
47  class Circle : public Point {
48  public:
49     // default constructor
50     Circle( double r = 0.0, int x = 0, int y = 0 );
51
52     ~Circle();
53  private:
54     double radius;
55  }; // end class Circle
56
57  #endif
```



Outline

circle2.cpp

```
58  // Fig. 19.7: circle2.cpp
59  // Member function definitions for class Circle
60  #include <iostream>
61
62  using std::cout;
63  using std::endl;
64
65  #include "circle2.h"
66
67  // Constructor for Circle calls constructor for Point
68  Circle::Circle( double r, int a, int b )
69     : Point( a, b )   // call base-class constructor
70  {
71     radius = r;  // should validate
72     cout << "Circle constructor: radius is "
73          << radius << " [" << x << ", " << y << ']' << endl;
74  } // end Circle constructor
75
76  // Destructor for class Circle
77  Circle::~Circle()
78  {
79     cout << "Circle destructor:  radius is "
80          << radius << " [" << x << ", " << y << ']' << endl;
81  } // end Circle destructor
```

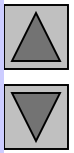


Outline

fig19_07.cpp (1 of 2)

```
// Fig. 19.7: fig19_07.cpp
// Demonstrate when base-class and derived-class
// constructors and destructors are called.
#include <iostream>

using std::cout;
using std::endl;

#include "point2.h"
#include "circle2.h"

int main()
{
   // Show constructor and destructor calls for Point
   {
      Point p( 11, 22 );
   } // end block

   cout << endl;
   Circle circle1( 4.5, 72, 29 );
   cout << endl;
   Circle circle2( 10, 5, 5 );
   cout << endl;
   return 0;
} // end function main
```

```
Point  constructor: [11, 22]
Point  destructor:  [11, 22]

Point  constructor: [72, 29]
Circle constructor: radius is 4.5 [72, 29]

Point  constructor: [5, 5]
Circle constructor: radius is 10 [5, 5]

Circle destructor:  radius is 10 [5, 5]
Point  destructor:  [5, 5]
Circle destructor:  radius is 4.5 [72, 29]
Point  destructor:  [72, 29]
```



Outline

fig19_07.cpp (2 of 2)

19.10 Implicit Derived-Class Object to Base-Class Object Conversion

- `baseClassObject = derivedClassObject;`
  - This will work
    - Remember, the derived class object has more members than the base class object
  - Extra data is not given to the base class
- `derivedClassObject = baseClassObject;`
  - May not work properly
    - Unless an assignment operator is overloaded in the derived class, data members exclusive to the derived class will be unassigned
  - Base class has less data members than the derived class
    - Some data members missing in the derived class object

19.10 Implicit Derived-Class Object to Base-Class Object Conversion

- Four ways to mix base and derived class pointers and objects:
  - Referring to a base-class object with a base-class pointer
    - Allowed
  - Referring to a derived-class object with a derived-class pointer
    - Allowed
  - Referring to a derived-class object with a base-class pointer
    - Possible syntax error
    - Code can only refer to base-class members, or syntax error
  - Referring to a base-class object with a derived-class pointer
    - Syntax error
    - The derived-class pointer must first be cast to a base-class pointer

19.11 Software Engineering With Inheritance

- Classes are often closely related
  - "Factor out" common attributes and behaviors and place these in a base class
  - Use inheritance to form derived classes
- Modifications to a base class
  - Derived classes do not change as long as the `public` and `protected` interfaces are the same
  - Derived classes may need to be recompiled

19.12 Composition vs. Inheritance

- "is a" relationship
  - Inheritance
- "has a" relationship
  - Composition - class has an object from another class as a data member

`Employee` "is a" `BirthDate;  //Wrong!`

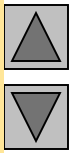
`Employee` "has a" `BirthDate;//Composition`

19.13 *Uses A* And *Knows A* Relationships

- "uses a" relationship
  - One object issues a function call to a member function of another object
- "knows a" relationship
  - One object is aware of another
    - Contains a pointer handle or reference handle to another object
  - Also called an association

19.14 Case Study: Point, Circle, Cylinder

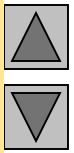
- Define class `Point`
  - Derive `Circle`
    - Derive `Cylinder`



Outline

point2.h

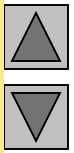
```
1    // Fig. 19.8: point2.h
2    // Definition of class Point
3    #ifndef POINT2_H
4    #define POINT2_H
5
6    #include <iostream>
7
8    using std::ostream;
9
10   class Point {
11      friend ostream &operator<<( ostream &, const Point & );
12   public:
13      Point( int = 0, int = 0 );      // default constructor
14      void setPoint( int, int );      // set coordinates
15      int getX() const { return x; }  // get x coordinate
16      int getY() const { return y; }  // get y coordinate
17   protected:         // accessible to derived classes
18      int x, y;       // coordinates of the point
19   }; // end class Point
20
21   #endif
```



Outline

point2.cpp

```
22  // Fig. 19.8: point2.cpp
23  // Member functions for class Point
24  #include "point2.h"
25
26  // Constructor for class Point
27  Point::Point( int a, int b ) { setPoint( a, b ); }
28
29  // Set the x and y coordinates
30  void Point::setPoint( int a, int b )
31  {
32     x = a;
33     y = b;
34  } // end function setPoint
35
36  // Output the Point
37  ostream &operator<<( ostream &output, const Point &p )
38  {
39     output << '[' << p.x << ", " << p.y << ']';
40
41     return output;           // enables cascading
42  } // end operator<< function
```

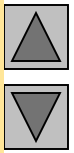


Outline

fig19_08.cpp

```
43  // Fig. 19.8: fig19_08.cpp
44  // Driver for class Point
45  #include <iostream>
46
47  using std::cout;
48  using std::endl;
49
50  #include "point2.h"
51
52  int main()
53  {
54     Point p( 72, 115 );   // instantiate Point object p
55
56     // protected data of Point inaccessible to main
57     cout << "X coordinate is " << p.getX()
58          << "\nY coordinate is " << p.getY();
59
60     p.setPoint( 10, 10 );
61     cout << "\n\nThe new location of p is " << p << endl;
62
63     return 0;
64  } // end function main
```

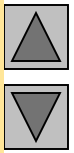
```
X coordinate is 72
Y coordinate is 115

The new location of p is [10, 10]
```



Outline

circle2.h

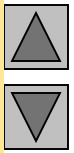
```
1   // Fig. 19.9: circle2.h
2   // Definition of class Circle
3   #ifndef CIRCLE2_H
4   #define CIRCLE2_H
5
6   #include <iostream>
7
8   using std::ostream;
9
10  #include "point2.h"
11
12  class Circle : public Point {
13     friend ostream &operator<<( ostream &, const Circle & );
14  public:
15     // default constructor
16     Circle( double r = 0.0, int x = 0, int y = 0 );
17     void setRadius( double );    // set radius
18     double getRadius() const;    // return radius
19     double area() const;         // calculate area
20  protected:          // accessible to derived classes
21     double radius;   // radius of the Circle
22  }; // end class Circle
23
24  #endif
```



Outline

circle2.cpp (1 of 2)

```
25  // Fig. 19.9: circle2.cpp
26  // Member function definitions for class Circle
27  #include <iomanip>
28
29  using std::ios;
30  using std::setiosflags;
31  using std::setprecision;
32
33  #include "circle2.h"
34
35  // Constructor for Circle calls constructor for Point
36  // with a member initializer and initializes radius
37  Circle::Circle( double r, int a, int b )
38     : Point( a, b )       // call base-class constructor
39  { setRadius( r ); }
40
41  // Set radius
42  void Circle::setRadius( double r )
43     { radius = ( r >= 0 ? r : 0 ); }
44
45  // Get radius
46  double Circle::getRadius() const { return radius; }
47
48  // Calculate area of Circle
49  double Circle::area() const
50     { return 3.14159 * radius * radius; }
51
```



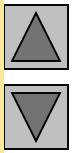
Outline

circle2.cpp (2 of 2)

```
52   // Output a circle in the form:
53   // Center = [x, y]; Radius = #.##
54   ostream &operator<<( ostream &output, const Circle &c )
55   {
56      output << "Center = " << static_cast< Point > ( c )
57             << "; Radius = "
58             << setiosflags( ios::fixed | ios::showpoint )
59             << setprecision( 2 ) << c.radius;
60
61      return output;   //  enables cascaded calls
62   } // end operator<< function
```

fig19_09.cpp (1 of 2)

```
63   // Fig. 19.9: fig19_09.cpp
64   // Driver for class Circle
65   #include <iostream>
66
67   using std::cout;
68   using std::endl;
69
70   #include "point2.h"
71   #include "circle2.h"
72
```

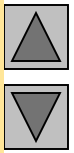


Outline

fig19_09.cpp (2 of 2)

```
73   int main()
74   {
75      Circle c( 2.5, 37, 43 );
76
77      cout << "X coordinate is " << c.getX()
78           << "\nY coordinate is " << c.getY()
79           << "\nRadius is " << c.getRadius();
80
81      c.setRadius( 4.25 );
82      c.setPoint( 2, 2 );
83      cout << "\n\nThe new location and radius of c are\n"
84           << c << "\nArea " << c.area() << '\n';
85
86      Point &pRef = c;
87      cout << "\nCircle printed as a Point is: " << pRef << endl;
88
89      return 0;
90   } // end function main
```

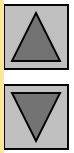
```
X coordinate is 37
Y coordinate is 43
Radius is 2.5

The new location and radius of c are
Center = [2, 2]; Radius = 4.25
Area 56.74

Circle printed as a Point is: [2, 2]
```



Outline

cylindr2.h

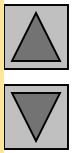
```
1   // Fig. 19.10: cylindr2.h
2   // Definition of class Cylinder
3   #ifndef CYLINDR2_H
4   #define CYLINDR2_H
5
6   #include <iostream>
7
8   using std::ostream;
9
10  #include "circle2.h"
11
12  class Cylinder : public Circle {
13     friend ostream &operator<<( ostream &, const Cylinder & );
14
15  public:
16     // default constructor
17     Cylinder( double h = 0.0, double r = 0.0,
18               int x = 0, int y = 0 );
19
20     void setHeight( double );   // set height
21     double getHeight() const;   // return height
22     double area() const;        // calculate and return area
23     double volume() const;      // calculate and return volume
24
25  protected:
26     double height;              // height of the Cylinder
27  }; // end class Cylinder
28
29  #endif
```



Outline

cylindr2.cpp (1 of 2)

```
30  // Fig. 19.10: cylindr2.cpp
31  // Member and friend function definitions
32  // for class Cylinder.
33  #include "cylindr2.h"
34
35  // Cylinder constructor calls Circle constructor
36  Cylinder::Cylinder( double h, double r, int x, int y )
37     : Circle( r, x, y )   // call base-class constructor
38  { setHeight( h ); }
39
40  // Set height of Cylinder
41  void Cylinder::setHeight( double h )
42     { height = ( h >= 0 ? h : 0 ); }
43
44  // Get height of Cylinder
45  double Cylinder::getHeight() const { return height; }
46
47  // Calculate area of Cylinder (i.e., surface area)
48  double Cylinder::area() const
49  {
50     return 2 * Circle::area() +
51            2 * 3.14159 * radius * height;
52  } // end function area
53
54  // Calculate volume of Cylinder
55  double Cylinder::volume() const
56     { return Circle::area() * height; }
57
```



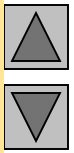
Outline

cylindr2.cpp (2 of 2)

```
58   // Output Cylinder dimensions
59   ostream &operator<<( ostream &output, const Cylinder &c )
60   {
61      output << static_cast< Circle >( c )
62             << "; Height = " << c.height;
63
64      return output;   // enables cascaded calls
65   } // end operator<< function
```

fig19_10.cpp (1 of 3)

```
66   // Fig. 19.10: fig19_10.cpp
67   // Driver for class Cylinder
68   #include <iostream>
69
70   using std::cout;
71   using std::endl;
72
73   #include "point2.h"
74   #include "circle2.h"
75   #include "cylindr2.h"
76
77   int main()
78   {
79      // create Cylinder object
80      Cylinder cyl( 5.7, 2.5, 12, 23 );
81
```

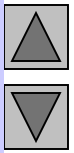


Outline

fig19_10.cpp (2 of 3)

```
82      // use get functions to display the Cylinder
83      cout << "X coordinate is " << cyl.getX()
84           << "\nY coordinate is " << cyl.getY()
85           << "\nRadius is " << cyl.getRadius()
86           << "\nHeight is " << cyl.getHeight() << "\n\n";
87
88      // use set functions to change the Cylinder's attributes
89      cyl.setHeight( 10 );
90      cyl.setRadius( 4.25 );
91      cyl.setPoint( 2, 2 );
92      cout << "The new location, radius, and height of cyl are:\n"
93           << cyl << '\n';
94
95      cout << "The area of cyl is:\n"
96           << cyl.area() << '\n';
97
98      // display the Cylinder as a Point
99      Point &pRef = cyl;   // pRef "thinks" it is a Point
100     cout << "\nCylinder printed as a Point is: "
101          << pRef << "\n\n";
102
103     // display the Cylinder as a Circle
104     Circle &circleRef = cyl;  // circleRef thinks it is a Circle
105     cout << "Cylinder printed as a Circle is:\n" << circleRef
106          << "\nArea: " << circleRef.area() << endl;
107
108     return 0;
109 } // end function main
```

```
X coordinate is 12
Y coordinate is 23
Radius is 2.5
Height is 5.7

The new location, radius, and height of cyl are:
Center = [2, 2]; Radius = 4.25; Height = 10.00
The area of cyl is:
380.53

Cylinder printed as a Point is: [2, 2]

Cylinder printed as a Circle is:
Center = [2, 2]; Radius = 4.25
Area: 56.74
```



Outline

fig19_10.cpp (3 of 3)