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

Base and Derived Classes

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

<table>
<thead>
<tr>
<th>Base class</th>
<th>Derived classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>GraduateStudent, UndergraduateStudent</td>
</tr>
<tr>
<td>Shape</td>
<td>Circle, Triangle, Rectangle</td>
</tr>
<tr>
<td>Loan</td>
<td>CarLoan, HomeImprovementLoan, MortgageLoan</td>
</tr>
<tr>
<td>Employee</td>
<td>FacultyMember, StaffMember</td>
</tr>
<tr>
<td>Account</td>
<td>CheckingAccount, SavingsAccount</td>
</tr>
</tbody>
</table>

Inheritance in C++

- Introduction
- Inheritance: Base Classes and Derived Classes
- Protected Members
- Casting Base-Class Pointers to Derived-Class Pointers
- Using Member Functions
- Overriding Base-Class Members in a Derived Class
- Public, Protected and Private Inheritance
- Direct Base Classes and Indirect Base Classes
- Using Constructors and Destructors in Derived Classes
- Implicit Derived-Class Object to Base-Class Object
- Conversion
- Software Engineering with Inheritance
- Composition vs. Inheritance
- "Uses A" and "Knows A" Relationships
- Case Study: Point, Circle, Cylinder
**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

---

### 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

---

#### 1. Point class definition

```cpp
class Point {  // Point class definition  
public:
  Point( int = 0, int = 0 );   // default constructor
  void setPoint( int, int );     // set coordinates
  int getX() const { return x; }  // get x coordinate
  int getY() const { return y; }  // get y coordinate
protected: // accessible by derived classes
  int x, y;       // x and y coordinates of the Point
};
```

---

#### 1.1 Function definitions

```cpp
1.1 Function definitions
------------------------
```

#### 1. Circle definition

```cpp
class Circle : public Point { // Circle inherits from Point
private:
  double radius;  // radius of the Circle
};
```

---

#### 1.1 Function Definitions

```cpp
1.1 Function Definitions
------------------------
```
13 // Fig. 19.4: fig19_04.cpp
14 // Casting base-class pointers to derived-class pointers
15 #include <iostream>
16
17 using std::cout;
18 using std::endl;
19
20 #include <iomanip>
21
22 #include "point.h"
23 #include "circle.h"
24
25 int main()
26 {
27    Point *pointPtr = 0, p(30, 50);
28    Circle *circlePtr = 0, c(2.7, 120, 89);
29    
30    // Get radius of Circle
31    double Circle::getRadius() const { return radius; }
32    
33    // Calculate area of Circle
34    double Circle::area() const
35    { return 3.14159 * radius * radius; }
36    
37    // Output a Circle in the form:
38    // Center = [x, y]; Radius = #.##
39    ostream &operator<<( ostream &output, const Circle &c )
40    {
41        output << "Center = " << static_cast< Point >( c )
42        << "; Radius = "
43        << setiosflags( ios::fixed | ios::showpoint )
44        << setprecision( 2 ) << c.radius;
45    
46        return output;   // enables cascaded calls
47    }
48    
49    Circle *circlePtr = static_cast< Circle * >( pointPtr );
50    cout << "Point p (via *circlePtr):
51    " << *circlePtr << "\n";
52    cout << "Area of object circlePtr points to: "
53    << circlePtr->area() << endl;
54    return 0;
55}
1.1 Function definitions

```cpp
#include "hourly.h"

// Overriding a base-class member function in a
// Example.cpp

class MinuteWorker : public HourlyWorker
{

 public:

 // Constructor for class MinuteWorker
 MinuteWorker( const char* first, const char* last, double wage, double hours );

};

public:

 // Constructor for class HourlyWorker
 HourlyWorker( const char* first, const char* last, double initWage, double initHours );

};

```

1.1 Initialize object

1.2 Function call

```cpp
void print() const
{
    // Print the HourlyWorker's name and pay
    std::cout << std::setprecision( 2 ) << getPay() << std::endl;
    std::cout << std::setiosflags( std::ios::fixed | std::ios::showpoint )
    << std::endl << " is an hourly worker with pay of " << getPay();

    // Call base-class print function
    Employee::print();
    std::cout << "HourlyWorker::print() is executing\n\n";

    // Overriding a base-class member function in a
    void print() const
    {
        // Output employee name
        std::cout << Employee::print() << std::endl;
    }
}
```
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

1. 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

2. Destructors are called in the reverse order of constructor calls.
   - Derived-class destructor is called before its base-class destructor

```cpp
1 // 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};
13
14 #endif // point2.h
```

```cpp
15 // 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    cout << "Point constructor: [" << x << ", " << y << "]" << endl;
30}
31
32 // Destructor for class Point
33 Point::~Point()
34 {
35    cout << "Point destructor: [" << x << ", " << y << "]" << endl;
36}
```

```cpp
82 // Example.cpp
83 // Demonstrate when base-class and derived-class constructors and destructors are called.
84 #include "point2.h"
85 #include "circle2.h"
86
87 int main()
88 {
89    // Show constructor and destructor calls for Point
90    Point p( 11, 22 );
91
92    cout << endl;
93    Circle circle1( 4.5, 72, 29 );
94    cout << endl;
95    Circle circle2( 10, 5, 5 );
96    cout << endl;
97    return 0;
98}
```

Program Output

```
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 constructor: radius is 4.5 [72, 29]  
Point destructor: [72, 29]  
```
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

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

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
```

“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 or handle to another object
    - Also called an association

Case Study: Point, Circle, Cylinder

- Define class `Point`
- Derive `Circle`
- Derive `Cylinder`
1. Point definition

1.1 Function definitions

1. Circle definition

1.1 Function definitions

1. Cylinder definition

1.1 Function definitions

1. Function definitions

1.1 Cylinder definition

1.1 Function definitions
# include <iostream>

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

# include "point2.h"
# include "circle2.h"
# include "cylindr2.h"

int main()
{
    // create Cylinder object
    Cylinder cyl( 5.7, 2.5, 12, 23 );

    // use get functions to display the Cylinder
    cout << "X coordinate is " << cyl.getX() << "
Y coordinate is " << cyl.getY() << "
Radius is " << cyl.getRadius() << "
Height is " << cyl.getHeight() << "

    // use set functions to change the Cylinder's attributes
    cyl.setHeight( 10 );
    cyl.setRadius( 4.25 );
    cyl.setPoint( 2, 2 );

    // output << static_cast< Circle >( c ) << "; Height = " << c.height;
    output << cyl; // enables cascaded calls

    return output;   // enables cascaded calls
}

---

Driver

1. Load headers
1.1 Initialize object
2. Function calls
2.1 Change attributes
3. Output

---

Program Output

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

---

output << static_cast< Circle >( c ) << "; Height = " << c.height;
output << cyl; // enables cascaded calls
return output;   // enables cascaded calls

---

Program Output

X coordinate is 12
Y coordinate is 23
Radius = 2.5
Height = 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