

Tính đa hình

Giới thiệu

- Tính đa hình (Polymorphism)
 - “Program in the general”
 - coi các đối tượng của lớp kế thừa như đối tượng của lớp cơ sở
 - hàm ảo và kết nối động
 - giúp chương trình có tính dễ mở rộng
 - thêm lớp mới dễ
- Ví dụ
 - Sử dụng lớp trừu tượng **Shape**
 - Định nghĩa interface chung
 - **Point**, **Circle** và **Cylinder** kế thừa từ **Shape**

Quan hệ giữa các đối tượng trong cây kế thừa³

- Các bài trước,
 - **Circle** kế thừa từ **Point**
 - Các thao tác trên đối tượng của **Point** và **Circle** sử dụng hàm thành phần
- Từ bài này,
 - Gọi hàm sử dụng con trỏ lớp cơ sở/lớp dẫn xuất
 - Khái niệm hàm ảo (**virtual**)
- Ý nghĩa
 - coi đối tượng của lớp dẫn xuất như đối tượng của lớp cơ sở
 - quan hệ “is-a”
 - Lớp cơ sở không phải là một đối tượng của lớp dẫn xuất

Gọi hàm lớp cơ sở từ đối tượng lớp dẫn xuất⁴

- Con trỏ (cơ sở, dẫn xuất) trỏ vào đối tượng (con trỏ, dẫn xuất)
 - con trỏ lớp cơ sở trỏ vào đối tượng lớp cơ sở
 - con trỏ lớp dẫn xuất trỏ vào đối tượng lớp dẫn xuất
 - con trỏ lớp cơ sở trỏ vào đối tượng lớp dẫn xuất
 - quan hệ “is a”
 - **Circle** “is a” **Point**
 - sẽ gọi hàm ở lớp cơ sở
 - Lời gọi hàm phụ thuộc vào lớp của pointer
 - không phụ thuộc vào đối tượng nó trỏ tới
 - Với hàm ảo (**virtual** functions), điều này sẽ thay đổi



```
1 // Fig. 10.1: point.h
2 // Point class definition represents an x-y coordinate pair.
3 #ifndef POINT_H
4 #define POINT_H
5
6 class Point {
7
8 public:
9     Point( int = 0, int = 0 ); // default constructor
10
11     void setX( int ); // set x in coordinate pair
12     int getX() const; // return x from coordinate pair
13
14     void setY( int ); // set y in coordinate pair
15     int getY() const; // return y from coordinate pair
16
17     void print() const; // output Point object
18
19 private:
20     int x; // x part of coordinate pair
21     int y; // y part of coordinate pair
22
23 }; // end class Point
24
25 #endif
```

Hàm print của lớp cơ sở



```
1  // Fig. 10.2: point.cpp
2  // Point class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "point.h"    // Point class definition
8
9  // default constructor
10 Point::Point( int xValue, int yValue )
11     : x( xValue ), y( yValue )
12 {
13     // empty body
14
15 } // end Point constructor
16
17 // set x in coordinate pair
18 void Point::setX( int xValue )
19 {
20     x = xValue; // no need for validation
21
22 } // end function setX
23
```



```
24 // return x from coordinate pair
25 int Point::getX() const
26 {
27     return x;
28
29 } // end function getX
30
31 // set y in coordinate pair
32 void Point::setY( int yValue )
33 {
34     y = yValue; // no need for validation
35
36 } // end function setY
37
38 // return y from coordinate pair
39 int Point::getY() const
40 {
41     return y;
42
43 } // end function getY
44
45 // output Point object
46 void Point::print() const
47 {
48     cout << '[' << getX() << ", " << getY() << ']' ;
49
50 } // end function print
```

Hiển thị tọa độ x, y của
Point.



```
1  // Fig. 10.3: circle.h
2  // Circle class contains x-y coordinate pair and radius.
3  #ifndef CIRCLE_H
4  #define CIRCLE_H
5
6  #include "point.h"  // Point class definition
7
8  class Circle : public Point {
9
10 public:
11
12     // default constructor
13     Circle( int = 0, int = 0, double =
14
15     void setRadius( double );    // set radius
16     double getRadius() const;    // return radius
17
18     double getDiameter() const;  // return diameter
19     double getCircumference() const; // return circumference
20     double getArea() const;      // return area
21
22     void print() const;          // output Circle object
23
24 private:
25     double radius;  // Circle's radius
26
27 }; // end class Circle
28
29 #endif
```

Circle kế thừa từ **Point**,
nhưng định nghĩa lại hàm
print



```
1  // Fig. 10.4: circle.cpp
2  // Circle class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "circle.h"    // Circle class definition
8
9  // default constructor
10 Circle::Circle( int xValue, int yValue, double radiusValue )
11     : Point( xValue, yValue )    // call base-class constructor
12 {
13     setRadius( radiusValue );
14
15 } // end Circle constructor
16
17 // set radius
18 void Circle::setRadius( double radiusValue )
19 {
20     radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
21
22 } // end function setRadius
23
```



```
24 // return radius
25 double Circle::getRadius() const
26 {
27     return radius;
28
29 } // end function getRadius
30
31 // calculate and return diameter
32 double Circle::getDiameter() const
33 {
34     return 2 * getRadius();
35
36 } // end function getDiameter
37
38 // calculate and return circumference
39 double Circle::getCircumference() const
40 {
41     return 3.14159 * getDiameter();
42
43 } // end function getCircumference
44
45 // calculate and return area
46 double Circle::getArea() const
47 {
48     return 3.14159 * getRadius() * getRadius();
49
50 } // end function getArea
```

```
51
52 // output Circle object
53 void Circle::print() const
54 {
55     cout << "center = ";
56     Point::print(); // invoke Point's print function
57     cout << "; radius = " << getRadius();
58
59 } // end function print
```

Circle định nghĩa làm hàm print . Nó gọi hàm print của **Point** để hiển thị tọa độ tâm x, y và hiển thị bán kính



```
1  // Fig. 10.5: fig10_05.cpp
2  // Aiming base-class and derived-class pointers at base-
class
3  // and derived-class objects, respectively.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8  using std::fixed;
9
10 #include <iomanip>
11
12 using std::setprecision;
13
14 #include "point.h"    // Point class definition
15 #include "circle.h"  // Circle class definition
16
17 int main()
18 {
19     Point point( 30, 50 );
20     Point *pointPtr = 0;    // base-class pointer
21
22     Circle circle( 120, 89, 2.7 );
23     Circle *circlePtr = 0; // derived-class pointer
24
```

fig10_05.cpp
 (2 of 3)

```

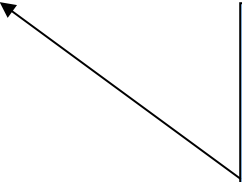
25 // set floating-point numeric form
26 cout << fixed << setprecision( 2 )
27
28 // output objects point and circle
29 cout << "Print point and circle objects:"
30     << "\nPoint: ";
31 point.print(); // invokes Point's print
32 cout << "\nCircle: ";
33 circle.print(); // invokes Circle's print
34
35 // aim base-class pointer at base-class object and print
36 pointPtr = &point;
37 cout << "\n\nCalling print with base-class pointer to "
38     << "\nbase-class object invokes base-class print "
39     << "function:\n";
40 pointPtr->print(); // invokes Point's print
41
42 // aim derived-class pointer at derived-class object
43 // and print
44 circlePtr = &circle;
45 cout << "\n\nCalling print with derived-class pointer to "
46     << "\nderived-class object invokes derived-class "
47     << "print function:\n";
48 circlePtr->print(); // invokes Circle's print
49

```

Dùng đối tượng và con trỏ để gọi tới hàm **print**. Con trỏ và đối tượng cùng lớp gọi tới cùng hàm **print**.

```
50 // aim base-class pointer at derived-class object and
print
51 pointPtr = &circle;

52 cout << "\n\nCalling print with base-class pointer to
"
53 << "derived-class object\ninvokes base-class
print "
54 << "function on that derived-class object:\n";
55 pointPtr->print(); // invokes Point's print
56 cout << endl;
57
58 return 0;
59
60 } // end main
```



Con trỏ lớp cơ sở trỏ tới đối tượng lớp dẫn xuất (**Circle** “is a” **Point**). Tuy nhiên, nó gọi hàm print của lớp **Point**, do kiểu của con trỏ là Point. Dùng hàm ảo sẽ thay đổi điều này.



Print point and circle objects:

Point: [30, 50]

Circle: center = [120, 89]; radius = 2.70

Calling print with base-class pointer to
base-class object invokes base-class print function:
[30, 50]

Calling print with derived-class pointer to
derived-class object invokes derived-class print function:
center = [120, 89]; radius = 2.70

Calling print with base-class pointer to derived-class
object
invokes base-class print function on that derived-class
object:
[120, 89]

fig10_05.cpp
output (1 of 1)

Con trỏ lớp dẫn xuất trỏ vào đối tượng lớp cơ sở

- Ví dụ trước
 - Con trỏ lớp cơ sở trỏ vào đối tượng lớp dẫn xuất
 - **Circle** “is a” **Point**
- Con trỏ lớp dẫn xuất trỏ vào đối tượng lớp cơ sở
 - Lỗi biên dịch
 - Không phải quan hệ “is a”
 - **Point** not is a **Circle**
 - **Circle** có dữ liệu/hàm mà **Point** không có
 - **setRadius** (định nghĩa trong **Circle**) không được định nghĩa cho **Point**
 - có thể gán địa chỉ của đối tượng cơ sở cho con trỏ của lớp dẫn xuất
 - cho phép dùng các hàm của lớp dẫn xuất



fig10_06.cpp
(1 of 1)

fig10_06.cpp
output (1 of 1)

```
1  // Fig. 10.6: fig10_06.cpp
2  // Aiming a derived-class pointer at a base-class object.
3  #include "point.h"    // Point class definition
4  #include "circle.h"   // Circle class definition
5
6  int main()
7  {
8      Point point( 30, 50 );
9      Circle *circlePtr = 0;
10
11     // aim derived-class pointer at base-class object
12     circlePtr = &point; // Error: a Point is not a Circle
13
14     return 0;
15
16 } // end main
```

C:\cpphttp4\examples\ch10\fig10_06\Fig10_06.cpp(12) : error
C2440:

'=' : cannot convert from 'class Point *' to 'class Circle
*'

Types pointed to are unrelated; conversion requires
reinterpret_cast, C-style cast or function-style

cast

Gọi hàm lớp dẫn xuất thông qua con trỏ lớp cơ sở


- Handle (pointer/reference)
 - Con trỏ lớp cơ sở có thể trỏ tới đối tượng lớp dẫn xuất
 - nhưng chỉ có thể gọi tới hàm lớp cơ sở
 - Gọi tới hàm lớp dẫn xuất sẽ dẫn tới lỗi biên dịch
 - vì hàm không được định nghĩa trong lớp cơ sở
- Nhận xét
 - Kiểu dữ liệu của con trỏ/tham chiếu quyết định hàm nó có thể gọi



```
1  // Fig. 10.7: fig10_07.cpp
2  // Attempting to invoke derived-class-only member
   functions
3  // through a base-class pointer.
4  #include "point.h"    // Point class definition
5  #include "circle.h"   // Circle class definition
6
7  int main()
8  {
9      Point *pointPtr = 0;
10     Circle circle( 120, 89, 2.7 );
11
12     // aim base-class pointer at derived-class object
13     pointPtr = &circle;
14
15     // invoke base-class member functions on derived-
   class
16     // object through base-class pointer
17     int x = pointPtr->getX();
18     int y = pointPtr->getY();
19     pointPtr->setX( 10 );
20     pointPtr->setY( 10 );
21     pointPtr->print();
22
```

fig10_07.cpp
(2 of 2)

```
23 // attempt to invoke derived-class-only member
functions
24 // on derived-class object through base-class pointer
25 double radius = pointPtr->getRadius();
26 pointPtr->setRadius( 33.33 );
27 double diameter = pointPtr->getDiameter();
28 double circumference = pointPtr->getCircumference();
29 double area = pointPtr->getArea();
30
31 return 0;
32
33 } // end main
```



Hàm này định nghĩa trong **Circle**. Còn **pointPtr** có kiểu là **Point**.



fig10_07.cpp
output (1 of 1)

```
C:\cpphttp4\examples\ch10\fig10_07\fig10_07.cpp(25) : error C2039:
'getRadius' : is not a member of 'Point'
    C:\cpphttp4\examples\ch10\fig10_07\point.h(6) :
    see declaration of 'Point'

C:\cpphttp4\examples\ch10\fig10_07\fig10_07.cpp(26) : error C2039:
'setRadius' : is not a member of 'Point'
    C:\cpphttp4\examples\ch10\fig10_07\point.h(6) :
    see declaration of 'Point'

C:\cpphttp4\examples\ch10\fig10_07\fig10_07.cpp(27) : error C2039:
'getDiameter' : is not a member of 'Point'
    C:\cpphttp4\examples\ch10\fig10_07\point.h(6) :
    see declaration of 'Point'

C:\cpphttp4\examples\ch10\fig10_07\fig10_07.cpp(28) : error C2039:
'getCircumference' : is not a member of 'Point'
    C:\cpphttp4\examples\ch10\fig10_07\point.h(6) :
    see declaration of 'Point'

C:\cpphttp4\examples\ch10\fig10_07\fig10_07.cpp(29) : error C2039:
'getArea' : is not a member of 'Point'
    C:\cpphttp4\examples\ch10\fig10_07\point.h(6) :
    see declaration of 'Point'
```

Hàm ảo (Virtual Functions)

- Kiểu của con trỏ quyết định hàm nó có thể gọi
- **virtual** functions
 - Đối tượng (KHÔNG phải con trỏ) quyết định hàm có thể gọi
- Why useful?
 - Giả sử **Circle**, **Triangle**, **Rectangle** dẫn xuất từ **Shape**
 - Mỗi kiểu có hàm **draw** riêng
 - Để có thể vẽ bất kì hình nào
 - Khai báo con trỏ **Shape** , gọi tới **draw**
 - Chương trình xác định hàm **draw** trong quá trình chạy (động)
 - Coi các shape tương tự nhau

Virtual Functions

- Khai báo **draw** là **virtual** trong lớp cơ sở
 - Định nghĩa lại **draw** ở mỗi lớp dẫn xuất
 - Nếu hàm được định nghĩa **virtual**, nó chỉ có thể được ghi đè
 - **virtual void draw() const;**
 - Nếu ta định nghĩa một hàm **virtual**, **virtual** ở trong tất cả các lớp dẫn xuất
- Liên kết động (Dynamic binding)
 - Chọn hàm sẽ được gọi trong quá trình chạy chương trình

Virtual Functions

- Ví dụ
 - Định nghĩa **Point**, **Circle** dùng **virtual** functions
 - Con trỏ **Point** trỏ tới **Circle**
 - Gọi tới hàm **print** của lớp **Circle**


```
1 // Fig. 10.8: point.h
2 // Point class definition represents an x-y coordinate pair.
3 #ifndef POINT_H
4 #define POINT_H
5
6 class Point {
7
8 public:
9     Point( int = 0, int = 0 ); // default constructor
10
11     void setX( int ); // set x in coordinate pair
12     int getX() const; // return x from coordinate pair
13
14     void setY( int ); // set y in coordinate pair
15     int getY() const; // return y from coordinate pair
16
17     virtual void print() const; // output Point object
18
19 private:
20     int x; // x part of coordinate pair
21     int y; // y part of coordinate pair
22
23 }; // end class Point
24
25 #endif
```

Hàm print là hàm ảo. Nó sẽ là hàm ảo ở tất cả các lớp dẫn xuất.



```
1  // Fig. 10.9: circle.h
2  // Circle class contains x-y coordinate pair and radius.
3  #ifndef CIRCLE_H
4  #define CIRCLE_H
5
6  #include "point.h"  // Point class definition
7
8  class Circle : public Point {
9
10 public:
11
12     // default constructor
13     Circle( int = 0, int = 0, double = 0.0 );
14
15     void setRadius( double );    // set radius
16     double getRadius() const;    // return radius
17
18     double getDiameter() const;  // return diameter
19     double getCircumference() const; // return circumference
20     double getArea() const;      // return area
21
22     virtual void print() const;   // output Circle object
23
24 private:
25     double radius;  // Circle's radius
26
27 }; // end class Circle
28
29 // ...
```



```
1 // Fig. 10.10: fig10_10.cpp
2 // Introducing polymorphism, virtual functions and dynamic
3 // binding.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8 using std::fixed;
9
10 #include <iomanip>
11
12 using std::setprecision;
13
14 #include "point.h" // Point class definition
15 #include "circle.h" // Circle class definition
16
17 int main()
18 {
19     Point point( 30, 50 );
20     Point *pointPtr = 0;
21
22     Circle circle( 120, 89, 2.7 );
23     Circle *circlePtr = 0;
24
```



```
25 // set floating-point numeric formatting
26 cout << fixed << setprecision( 2 );
27
28 // output objects point and circle using static binding
29 cout << "Invoking print function on point and circle "
30      << "\nobjects with static binding "
31      << "\n\nPoint: ";
32 point.print();           // static binding
33 cout << "\nCircle: ";
34 circle.print();         // static binding
35
36 // output objects point and circle using dynamic binding
37 cout << "\n\nInvoking print function on point and circle
38      << "\nobjects with dynamic binding";
39
40 // aim base-class pointer at base-class object and print
41 pointPtr = &point;
42 cout << "\n\nCalling virtual function print with base-
43      << "\npointer to base-class object"
44      << "\ninvokes base-class print function:\n";
45 pointPtr->print();
46
```



```
47 // aim derived-class pointer at derived-class
48 // object and print
49 circlePtr = &circle;
50 cout << "\n\nCalling virtual function print with "
51      << "\nderived-class pointer to derived-class object "
52      << "\ninvokes derived-class print function:\n";
53 circlePtr->print();
54
55 // aim base-class pointer at derived-class object and
56 print
57 pointPtr = &circle;
58
59 cout << "\n\nCalling virtual function print with base-
60 class"
61      << "\npointer to derived-class object "
62      << "\ninvokes derived-class print function:\n";
63 pointPtr->print(); // polymorphism: invokes circle's
64 print
65 cout << endl;
66
67 return 0;
68
69 } // end main
```

Trong quá trình chạy, chương trình sẽ xác định **pointPtr** trỏ tới đối tượng **Circle**, và gọi hàm **print** của **Circle**. Đây là một ví dụ về tính đa hình.



Invoking print function on point and circle
objects with static binding

Point: [30, 50]

Circle: Center = [120, 89]; Radius = 2.70

Invoking print function on point and circle
objects with dynamic binding

Calling virtual function print with base-class
pointer to base-class object
invokes base-class print function:
[30, 50]

Calling virtual function print with
derived-class pointer to derived-class object
invokes derived-class print function:
Center = [120, 89]; Radius = 2.70

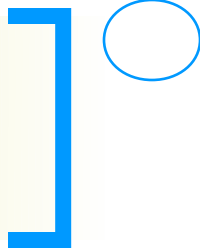
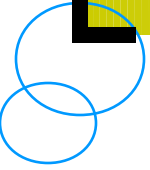
Calling virtual function print with base-class
pointer to derived-class object
invokes derived-class print function:
Center = [120, 89]; Radius = 2.70

Virtual Functions

- Đa hình
 - Cùng thông điệp, “print”, dùng cho nhiều đối tượng
- Tóm tắt
 - Con trỏ lớp cơ sở trỏ tới đối tượng lớp cơ sở, con trỏ lớp dẫn xuất trỏ tới đối tượng lớp dẫn xuất
 - Con trỏ lớp cơ sở trỏ tới đối tượng lớp dẫn xuất
 - chỉ gọi được các hàm lớp cơ sở
 - Con trỏ lớp dẫn xuất trỏ tới đối tượng lớp cơ sở
 - lỗi biên dịch
 - có thể dùng nếu có ép kiểu

Ví dụ

- Ví dụ
 - Giả sử **Rectangle** dẫn xuất từ **Quadrilateral**
 - **Rectangle** là một trường hợp của **Quadrilateral**
 - Các thao tác trên **Quadrilateral** có thể làm được trên **Rectangle** (ví dụ, tính diện tích...)
- Ví dụ thiết kế video game
 - Lớp cơ sở **SpaceObject**
 - Lớp dẫn xuất **Martian**, **SpaceShip**, **LaserBeam**
 - Hàm cơ sở **draw**
 - Để refresh screen
 - Screen manager có **vector** của con trỏ cơ sở tới đối tượng
 - Gọi hàm **draw** tới đối tượng
 - Cùng tên hàm cho “many forms”



Lớp trừu tượng

- Lớp trừu tượng
 - là lớp cơ sở (lớp cơ sở trừu tượng)
 - Chưa hoàn thiện
 - Lớp dẫn xuất bổ xung những chỗ còn thiếu ("missing pieces")
 - Không thể tạo đối tượng từ lớp trừu tượng
 - Tuy nhiên, có thể có con trỏ, tham chiếu
- Lớp cụ thể (Concrete classes)
 - định nghĩa chi tiết
 - cài đặt các hàm
 - có thể định nghĩa đối tượng

Lớp trừu tượng

- Không nhất thiết phải định nghĩa lớp trừu tượng, nhưng dùng lớp trừu tượng rất hữu dụng
- Để tạo lớp trừu tượng
 - Cần một hay nhiều hàm ảo rỗng ("pure")
 - Định nghĩa hàm với khởi tạo = 0
 - virtual void draw() const = 0;**
 - Hàm ảo thông thường
 - có chi tiết cài đặt, có thể thực hiện nạp chồng
 - Hàm ảo rỗng
 - không có cài đặt, PHẢI thực hiện nạp chồng
 - Lớp trừu tượng có thể có dữ liệu và phương thức cụ thể
 - nhưng yêu cầu phải có một hay nhiều hàm ảo rỗng

Lớp trừu tượng

- Con trỏ lớp trừu tượng
 - hữu dụng cho tính đa hình
- Ví dụ
 - Lớp trừu tượng **Shape**
 - Định nghĩa **draw** như hàm ảo rỗng
 - **Circle, Triangle, Rectangle** kế thừa từ **Shape**
 - Mỗi lớp, định nghĩa một hàm **draw**
 - Screen manager mỗi đối tượng có hành động vẽ riêng

Ví dụ

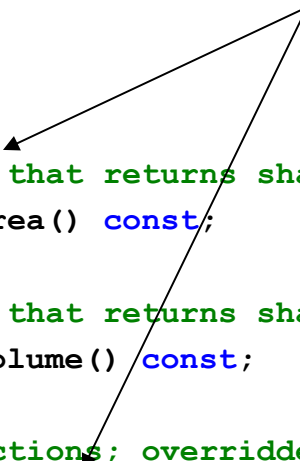
- Lớp trừu tượng **Shape**
 - Hàm ảo rỗng (phải được cài đặt sau)
 - **getName, print**
 - cài đặt mặc định là vô nghĩa
 - Hàm ảo (có thể định nghĩa lại)
 - **getArea, getVolume**
 - ban đầu return 0.0
 - Nếu không được định nghĩa lại, sử dụng định nghĩa ở lớp cơ sở
 - Lớp dẫn xuất **Point, Circle, Cylinder**

Ví dụ

	getArea	getVolume	getName	print
Shape	0.0	0.0	= 0	= 0
Point	0.0	0.0	"Point"	[x,y]
Circle	πr^2	0.0	"Circle"	center=[x,y]; radius=r
Cylinder	$2\pi r^2 + 2\pi rh$	$\pi r^2 h$	"Cylinder"	center=[x,y]; radius=r; height=h

```
1  // Fig. 10.12: shape.h
2  // Shape abstract-base-class definition.
3  #ifndef SHAPE_H
4  #define SHAPE_H
5
6  #include <string>  // C++ standard string class
7
8  using std::string;
9
10 class Shape {
11
12 public:
13
14     // virtual function that returns shape area
15     virtual double getArea() const;
16
17     // virtual function that returns shape volume
18     virtual double getVolume() const;
19
20     // pure virtual functions; overridden in derived classes
21     virtual string getName() const = 0; // return shape name
22     virtual void print() const = 0;     // output shape
23
24 }; // end class Shape
25
26 #endif
```

Hàm ảo và hàm ảo rỗng.





```
1  // Fig. 10.13: shape.cpp
2  // Shape class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "shape.h" // Shape class definition
8
9  // return area of shape; 0.0 by default
10 double getArea() const
11 {
12     return 0.0;
13
14 } // end function getArea
15
16 // return volume of shape; 0.0 by default
17 double getVolume() const
18 {
19     return 0.0;
20
21 } // end function getVolume
```


point.h (1 of 2)

```
1 // Fig. 10.14: point.h
2 // Point class definition represents an x-y coordinate pair.
3 #ifndef POINT_H
4 #define POINT_H
5
6 #include "shape.h" // Shape class definition
7
8 class Point : public Shape {
9
10 public:
11     Point( int = 0, int = 0 ); // default constructor
12
13     void setX( int ); // set x in coordinate pair
14     int getX() const; // return x from coordinate pair
15
16     void setY( int ); // set y in coordinate pair
17     int getY() const; // return y from coordinate pair
18
19     // return name of shape (i.e., "Point" )
20     virtual string getName() const;
21
22     virtual void print() const; // output Point object
23
```

Point chỉ định nghĩa lại **getName** và **print**, vì **getArea** và **getVolume** là zero (dùng cài đặt mặc định).



point.h (2 of 2)

```
24 private:
25     int x; // x part of coordinate pair
26     int y; // y part of coordinate pair
27
28 }; // end class Point
29
30 #endif
```



```
1  // Fig. 10.15: point.cpp
2  // Point class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "point.h"    // Point class definition
8
9  // default constructor
10 Point::Point( int xValue, int yValue )
11     : x( xValue ), y( yValue )
12 {
13     // empty body
14
15 } // end Point constructor
16
17 // set x in coordinate pair
18 void Point::setX( int xValue )
19 {
20     x = xValue; // no need for validation
21
22 } // end function setX
23
```



```
24 // return x from coordinate pair
25 int Point::getX() const
26 {
27     return x;
28
29 } // end function getX
30
31 // set y in coordinate pair
32 void Point::setY( int yValue )
33 {
34     y = yValue; // no need for validation
35
36 } // end function setY
37
38 // return y from coordinate pair
39 int Point::getY() const
40 {
41     return y;
42
43 } // end function getY
44
```



```
45 // override pure virtual function getName: return name of Point
46 string Point::getName() const
47 {
48     return "Point";
49 }
50 // end function getName
51
52 // override pure virtual function print: output Point object
53 void Point::print() const
54 {
55     cout << '[' << getX() << ", " << getY() << ']' ;
56 }
57 // end function print
```

Phải nạp chồng hàm ảo rỗng
getName và **print**.



```
1  // Fig. 10.16: circle.h
2  // Circle class contains x-y coordinate pair and radius.
3  #ifndef CIRCLE_H
4  #define CIRCLE_H
5
6  #include "point.h"  // Point class definition
7
8  class Circle : public Point {
9
10 public:
11
12     // default constructor
13     Circle( int = 0, int = 0, double = 0.0 );
14
15     void setRadius( double );    // set radius
16     double getRadius() const;    // return radius
17
18     double getDiameter() const;    // return diameter
19     double getCircumference() const; // return circumference
20     virtual double getArea() const; // return area
21
22     // return name of shape (i.e., "Circle")
23     virtual string getName() const;
24
25     virtual void print() const;    // output Circle object
```



```
26
27 private:
28     double radius; // Circle's radius
29
30 }; // end class Circle
31
32 #endif
```



```
1  // Fig. 10.17: circle.cpp
2  // Circle class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "circle.h"    // Circle class definition
8
9  // default constructor
10 Circle::Circle( int xValue, int yValue, double radiusValue )
11     : Point( xValue, yValue )    // call base-class constructor
12 {
13     setRadius( radiusValue );
14
15 } // end Circle constructor
16
17 // set radius
18 void Circle::setRadius( double radiusValue )
19 {
20     radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
21
22 } // end function setRadius
23
```




```
24 // return radius
25 double Circle::getRadius() const
26 {
27     return radius;
28
29 } // end function getRadius
30
31 // calculate and return diameter
32 double Circle::getDiameter() const
33 {
34     return 2 * getRadius();
35
36 } // end function getDiameter
37
38 // calculate and return circumference
39 double Circle::getCircumference() const
40 {
41     return 3.14159 * getDiameter();
42
43 } // end function getCircumference
44
```



```
45 // override virtual function getArea: return area of Circle
46 double Circle::getArea() const
47 {
48     return 3.14159 * getRadius() * getRadius();
49 } // end function getArea
50
51
52 // override virtual function getName
53 string Circle::getName() const
54 {
55     return "Circle";
56 } // end function getName
57
58
59 // override virtual function print: output Circle object
60 void Circle::print() const
61 {
62     cout << "center is ";
63     Point::print(); // invoke Point's print function
64     cout << "; radius is " << getRadius();
65 } // end function print
```

Nạp chồng **getArea** vì giờ
nó được áp dụng với Circle.



```
1  // Fig. 10.18: cylinder.h
2  // Cylinder class inherits from class Circle.
3  #ifndef CYLINDER_H
4  #define CYLINDER_H
5
6  #include "circle.h"  // Circle class definition
7
8  class Cylinder : public Circle {
9
10 public:
11
12     // default constructor
13     Cylinder( int = 0, int = 0, double = 0.0, double = 0.0 );
14
15     void setHeight( double );  // set Cylinder's height
16     double getHeight() const;  // return Cylinder's height
17
18     virtual double getArea() const;  // return Cylinder's area
19     virtual double getVolume() const;  // return Cylinder's volume
20
```



```
21 // return name of shape (i.e., "Cylinder" )
22 virtual string getName() const;
23
24 virtual void print() const; // output Cylinder
25
26 private:
27     double height; // Cylinder's height
28
29 }; // end class Cylinder
30
31 #endif
```

cylinder.cpp
(1 of 3)

```
1  // Fig. 10.19: cylinder.cpp
2  // Cylinder class inherits from class Circle.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "cylinder.h"    // Cylinder class definition
8
9  // default constructor
10 Cylinder::Cylinder( int xValue, int yValue, double radiusValue,
11     double heightValue )
12     : Circle( xValue, yValue, radiusValue )
13 {
14     setHeight( heightValue );
15
16 } // end Cylinder constructor
17
18 // set Cylinder's height
19 void Cylinder::setHeight( double heightValue )
20 {
21     height = ( heightValue < 0.0 ? 0.0 : heightValue );
22
23 } // end function setHeight
```



```
24
25 // get Cylinder's height
26 double Cylinder::getHeight() const
27 {
28     return height;
29
30 } // end function getHeight
31
32 // override virtual function getArea: return Cylinder area
33 double Cylinder::getArea() const
34 {
35     return 2 * Circle::getArea() +           // code reuse
36         getCircumference() * getHeight();
37
38 } // end function getArea
39
40 // override virtual function getVolume: return Cylinder volume
41 double Cylinder::getVolume() const
42 {
43     return Circle::getArea() * getHeight(); // code reuse
44
45 } // end function getVolume
46
```



cylinder.cpp

(3 of 3)

```
47 // override virtual function getName: return name of Cylinder
48 string Cylinder::getName() const
49 {
50     return "Cylinder";
51
52 } // end function getName
53
54 // output Cylinder object
55 void Cylinder::print() const
56 {
57     Circle::print(); // code reuse
58     cout << "; height is " << getHeight();
59
60 } // end function print
```

fig10_20.cpp
(1 of 5)

```
1  // Fig. 10.20: fig10_20.cpp
2  // Driver for shape, point, circle, cylinder hierarchy.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8
9  #include <iomanip>
10
11 using std::setprecision;
12
13 #include <vector>
14
15 using std::vector;
16
17 #include "shape.h"    // Shape class definition
18 #include "point.h"    // Point class definition
19 #include "circle.h"   // Circle class definition
20 #include "cylinder.h" // Cylinder class definition
21
22 void virtualViaPointer( const Shape * );
23 void virtualViaReference( const Shape & );
24
```




```
25  int main()
26  {
27      // set floating-point number format
28      cout << fixed << setprecision( 2 );
29
30      Point point( 7, 11 );           // create a
Point
31      Circle circle( 22, 8, 3.5 );   // create a
Circle
32      Cylinder cylinder( 10, 10, 3.3, 10 ); // create a
Cylinder
33
34      cout << point.getName() << ": "; // static binding
35      point.print();                   // static binding
36      cout << '\n';
37
38      cout << circle.getName() << ": "; // static binding
39      circle.print();                   // static binding
40      cout << '\n';
41
42      cout << cylinder.getName() << ": "; // static binding
43      cylinder.print();                   // static binding
44      cout << "\n\n";
45
```

```

46 // create vector of three base-class pointers
47 vector< Shape * > shapeVector( 3 );
48
49 // aim shapeVector[0] at derived-class P
50 shapeVector[ 0 ] = &point;
51
52 // aim shapeVector[1] at derived-class C
53 shapeVector[ 1 ] = &circle;
54
55 // aim shapeVector[2] at derived-class C
56 shapeVector[ 2 ] = &cylinder;
57
58 // loop through shapeVector and call vir
59 // to print the shape name, attributes, area and volume
60 // of each object using dynamic binding
61 cout << "\nVirtual function calls made off "
62      << "base-class pointers:\n\n";
63
64 for ( int i = 0; i < shapeVector.size(); i++ )
65     virtualViaPointer( shapeVector[ i ] );
66

```

Tạo một vector con trỏ **Shape**, cho chúng trỏ vào các đối tượng khác nhau .

Hàm **virtualViaPointer** gọi hàm ảo (**print**, **getName**,...) sử dụng con trỏ lớp cơ sở. Kiểu cụ thể được xác định “động” lúc chạy chương trình.

fig10_20.cpp
 (4 of 5)

 Use references instead of
 pointers, for the same effect.

 Gọi hàm ảo; hàm của lớp cụ
 thể sẽ được gọi trong quá
 trình chạy chương trình.

```

67 // loop through shapeVector and call virtualViaReference
68 // to print the shape name, attributes, area and volume
69 // of each object using dynamic binding
70 cout << "\nVirtual function calls made
71     << "base-class references:\n\n";
72
73 for ( int j = 0; j < shapeVector.size(); j++ )
74     virtualViaReference( *shapeVector[ j ] );
75
76 return 0;
77
78 } // end main
79
80 // make virtual function calls off a base-class
81 // using dynamic binding
82 void virtualViaPointer( const Shape *baseClass
83 {
84     cout << baseClassPtr->getName() << ": ";
85
86     baseClassPtr->print();
87
88     cout << "\narea is " << baseClassPtr->getArea()
89         << "\nvolume is " << baseClassPtr->getVolume()
90         << "\n\n";
91
92 } // end function virtualViaPointer
93
  
```



fig10_20.cpp
(5 of 5)

```
94 // make virtual function calls off a base-class reference
95 // using dynamic binding
96 void virtualViaReference( const Shape &baseClassRef )
97 {
98     cout << baseClassRef.getName() << ": ";
99
100    baseClassRef.print();
101
102    cout << "\narea is " << baseClassRef.getArea()
103         << "\nvolume is " << baseClassRef.getVolume() << "\n\n";
104
105 } // end function virtualViaReference
```



fig10_20.cpp
output (1 of 2)

```
Point: [7, 11]
Circle: center is [22, 8]; radius is 3.50
Cylinder: center is [10, 10]; radius is 3.30; height is 10.00
```

Virtual function calls made off base-class pointers:

```
Point: [7, 11]
area is 0.00
volume is 0.00
```

```
Circle: center is [22, 8]; radius is 3.50
area is 38.48
volume is 0.00
```

```
Cylinder: center is [10, 10]; radius is 3.30; height is 10.00
area is 275.77
volume is 342.12
```



fig10_20.cpp
output (2 of 2)

Virtual function calls made off base-class references:

Point: [7, 11]

area is 0.00

volume is 0.00

Circle: center is [22, 8]; radius is 3.50

area is 38.48

volume is 0.00

Cylinder: center is [10, 10]; radius is 3.30; height is 10.00

area is 275.77

volume is 342.12

Đa hình, hàm ảo và kết nối động

- Polymorphism has overhead
 - Not used in STL (Standard Template Library) to optimize performance
- Bảng hàm ảo **virtual** (vtable)
 - Tất cả các lớp sử dụng hàm ảo thì có vtable
 - Với mỗi hàm ảo, vtable có một con trỏ trỏ tới hàm proper
 - Nếu lớp dẫn xuất có hàm cùng tên với tên hàm lớp cơ sở
 - con trỏ hàm gọi tới hàm lớp cơ sở
 - (Fig. 10.21)

Hàm huỷ ảo (Virtual Destructors)

- Con trỏ cơ sở trỏ tới đối tượng dẫn xuất
 - nếu huỷ dùng **delete**, không xác định phương thức
- Cách sửa đơn giản
 - khai báo hàm huỷ ảo của lớp cơ sở
 - tạo hàm huỷ ảo của lớp dẫn xuất
 - Từ giờ, khi dùng **delete**, hàm huỷ tương ứng sẽ được gọi
- Khi đối tượng lớp dẫn xuất bị huỷ
 - Hàm huỷ của lớp dẫn xuất sẽ thực hiện trước
 - Hàm huỷ của lớp cơ sở thực hiện sau đó
- Hàm tạo không thể là hàm ảo