Illustrations List *(Main Page)*

```
1  // Fig. 10.1: employ2.h
2  // Abstract base class Employee
3  #ifndef EMPLOY2_H
4  #define EMPLOY2_H
5
6  #include <iostream.h>
7
8  class Employee {
9  public:
10    Employee( const char *, const char * );
11    ~Employee();   // destructor reclaims memory
12    const char *getFirstName() const;
13    const char *getLastName() const;
14
15    // Pure virtual function makes Employee abstract base class
16    virtual double earnings() const = 0;   // pure virtual
17    virtual void print() const;            // virtual
18 private:
19    char *firstName;
20    char *lastName;
21 };
22
23 #endif
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 1 of 13).

```
24 // Fig. 10.1: employ2.cpp
25 // Member function definitions for
26 // abstract base class Employee.
27 // Note: No definitions given for pure virtual functions.
28 #include <string.h>
29 #include <assert.h>
30 #include "employ2.h"
31
32 // Constructor dynamically allocates space for the
33 // first and last name and uses strcpy to copy
34 // the first and last names into the object.
35 Employee::Employee( const char *first, const char *last )
36 {
37    firstName = new char[ strlen( first ) + 1 ];
38    assert( firstName != 0 );    // test that new worked
39    strcpy( firstName, first );
40
41    lastName = new char[ strlen( last ) + 1 ];
42    assert( lastName != 0 );     // test that new worked
43    strcpy( lastName, last );
44 }
45
46 // Destructor deallocates dynamically allocated memory
47 Employee::~Employee()
48 {
49    delete [] firstName;
50    delete [] lastName;
51 }
52
53 // Return a pointer to the first name
54 // Const return type prevents caller from modifying private
55 // data. Caller should copy returned string before destructor
56 // deletes dynamic storage to prevent undefined pointer.
57 const char *Employee::getFirstName() const
58 {
59    return firstName;   // caller must delete memory
60 }
```

```
61
62  // Return a pointer to the last name
63  // Const return type prevents caller from modifying private
64  // data. Caller should copy returned string before destructor
65  // deletes dynamic storage to prevent undefined pointer.
66  const char *Employee::getLastName() const
67  {
68     return lastName;   // caller must delete memory
69  }
70
71  // Print the name of the Employee
72  void Employee::print() const
73     { cout << firstName << ' ' << lastName; }
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 2 of 13).

```
74  // Fig. 10.1: boss1.h
75  // Boss class derived from Employee
76  #ifndef BOSS1_H
77  #define BOSS1_H
78  #include "employ2.h"
79
80  class Boss : public Employee {
81  public:
82     Boss( const char *, const char *, double = 0.0 );
83     void setWeeklySalary( double );
84     virtual double earnings() const;
85     virtual void print() const;
86  private:
87     double weeklySalary;
88  };
89
90  #endif
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 3 of 13).

```
91  // Fig. 10.1: boss1.cpp
92  // Member function definitions for class Boss
93  #include "boss1.h"
94
95  // Constructor function for class Boss
96  Boss::Boss( const char *first, const char *last, double s )
97     : Employee( first, last )  // call base-class constructor
98  { setWeeklySalary( s ); }
99
100 // Set the Boss's salary
101 void Boss::setWeeklySalary( double s )
102    { weeklySalary = s > 0 ? s : 0; }
103
104 // Get the Boss's pay
105 double Boss::earnings() const { return weeklySalary; }
106
107 // Print the Boss's name
108 void Boss::print() const
109 {
110    cout << "\n             Boss: ";
111    Employee::print();
112 }
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 4 of 13).

```
// Fig. 10.1: commis1.h
// CommissionWorker class derived from Employee
#ifndef COMMIS1_H
#define COMMIS1_H
#include "employ2.h"

class CommissionWorker : public Employee {
public:
   CommissionWorker( const char *, const char *,
                     double = 0.0, double = 0.0,
                     int = 0 );
   void setSalary( double );
   void setCommission( double );
   void setQuantity( int );
   virtual double earnings() const;
   virtual void print() const;
private:
   double salary;       // base salary per week
   double commission;   // amount per item sold
   int quantity;        // total items sold for week
};

#endif
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 5 of 13).

```
// Fig. 10.1: commis1.cpp
// Member function definitions for class CommissionWorker
#include <iostream.h>
#include "commis1.h"

// Constructor for class CommissionWorker
CommissionWorker::CommissionWorker( const char *first,
        const char *last, double s, double c, int q )
   : Employee( first, last )  // call base-class constructor
{
   setSalary( s );
   setCommission( c );
   setQuantity( q );
}

// Set CommissionWorker's weekly base salary
void CommissionWorker::setSalary( double s )
   { salary = s > 0 ? s : 0; }

// Set CommissionWorker's commission
void CommissionWorker::setCommission( double c )
   { commission = c > 0 ? c : 0; }

// Set CommissionWorker's quantity sold
void CommissionWorker::setQuantity( int q )
   { quantity = q > 0 ? q : 0; }

// Determine CommissionWorker's earnings
double CommissionWorker::earnings() const
   { return salary + commission * quantity; }

// Print the CommissionWorker's name
void CommissionWorker::print() const
{
   cout << "\nCommission worker: ";
   Employee::print();
```

```
172 }
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 6 of 13).

```
173 // Fig. 10.1: piece1.h
174 // PieceWorker class derived from Employee
175 #ifndef PIECE1_H
176 #define PIECE1_H
177 #include "employ2.h"
178
179 class PieceWorker : public Employee {
180 public:
181    PieceWorker( const char *, const char *,
182                 double = 0.0, int = 0);
183    void setWage( double );
184    void setQuantity( int );
185    virtual double earnings() const;
186    virtual void print() const;
187 private:
188    double wagePerPiece; // wage for each piece output
189    int quantity;        // output for week
190 };
191
192 #endif
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 7 of 13).

```
193 // Fig. 10.1: piece1.cpp
194 // Member function definitions for class PieceWorker
195 #include <iostream.h>
196 #include "piece1.h"
197
198 // Constructor for class PieceWorker
199 PieceWorker::PieceWorker( const char *first, const char *last,
200                           double w, int q )
201    : Employee( first, last )  // call base-class constructor
202 {
203    setWage( w );
204    setQuantity( q );
205 }
206
207 // Set the wage
208 void PieceWorker::setWage( double w )
209    { wagePerPiece = w > 0 ? w : 0; }
210
211 // Set the number of items output
212 void PieceWorker::setQuantity( int q )
213    { quantity = q > 0 ? q : 0; }
214
215 // Determine the PieceWorker's earnings
216 double PieceWorker::earnings() const
217    { return quantity * wagePerPiece; }
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 8 of 13).

```
218
219 // Print the PieceWorker's name
220 void PieceWorker::print() const
221 {
222    cout << "\n     Piece worker: ";
```

```
223    Employee::print();
224 }
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 9 of 13).

```
225 // Fig. 10.1: hourly1.h
226 // Definition of class HourlyWorker
227 #ifndef HOURLY1_H
228 #define HOURLY1_H
229 #include "employ2.h"
230
231 class HourlyWorker : public Employee {
232 public:
233    HourlyWorker( const char *, const char *,
234                  double = 0.0, double = 0.0);
235    void setWage( double );
236    void setHours( double );
237    virtual double earnings() const;
238    virtual void print() const;
239 private:
240    double wage;   // wage per hour
241    double hours;  // hours worked for week
242 };
243
244 #endif
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 10 of 13).

```
 1 // Fig. 10.1: hourly1.cpp
 2 // Member function definitions for class HourlyWorker
 3 #include <iostream.h>
 4 #include "hourly1.h"
 5
 6 // Constructor for class HourlyWorker
 7 HourlyWorker::HourlyWorker( const char *first,
 8                             const char *last,
 9                             double w, double h )
10    : Employee( first, last )   // call base-class constructor
11 {
12    setWage( w );
13    setHours( h );
14 }
15
16 // Set the wage
17 void HourlyWorker::setWage( double w )
18    { wage = w > 0 ? w : 0; }
19
20 // Set the hours worked
21 void HourlyWorker::setHours( double h )
22    { hours = h >= 0 && h < 168 ? h : 0; }
23
24 // Get the HourlyWorker's pay
25 double HourlyWorker::earnings() const
26 {
27    if ( hours <= 40 ) // no overtime
28       return wage * hours;
29    else               // overtime is paid at wage * 1.5
30       return 40 * wage + ( hours - 40 ) * wage * 1.5;
31 }
32
33 // Print the HourlyWorker's name
```

```
34  void HourlyWorker::print() const
35  {
36     cout << "\n    Hourly worker: ";
37     Employee::print();
38  }
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 11 of 13).

```
39  // Fig. 10.1: fig10_01.cpp
40  // Driver for Employee hierarchy
41  #include <iostream.h>
42  #include <iomanip.h>
43  #include "employ2.h"
44  #include "boss1.h"
45  #include "commis1.h"
46  #include "piece1.h"
47  #include "hourly1.h"
48
49  void virtualViaPointer( const Employee * );
50  void virtualViaReference( const Employee & );
51
52  int main()
53  {
54     // set output formatting
55     cout << setiosflags( ios::fixed | ios::showpoint )
56          << setprecision( 2 );
57
58     Boss b( "John", "Smith", 800.00 );
59     b.print();                                  // static binding
60     cout << " earned $" << b.earnings();    // static binding
61     virtualViaPointer( &b );          // uses dynamic binding
62     virtualViaReference( b );         // uses dynamic binding
63
64     CommissionWorker c( "Sue", "Jones", 200.0, 3.0, 150 );
65     c.print();                                  // static binding
66     cout << " earned $" << c.earnings();    // static binding
67     virtualViaPointer( &c );          // uses dynamic binding
68     virtualViaReference( c );         // uses dynamic binding
69
70     PieceWorker p( "Bob", "Lewis", 2.5, 200 );
71     p.print();                                  // static binding
72     cout << " earned $" << p.earnings();    // static binding
73     virtualViaPointer( &p );          // uses dynamic binding
74     virtualViaReference( p );         // uses dynamic binding
75
76     HourlyWorker h( "Karen", "Price", 13.75, 40 );
77     h.print();                                  // static binding
78     cout << " earned $" << h.earnings();    // static binding
79     virtualViaPointer( &h );          // uses dynamic binding
80     virtualViaReference( h );         // uses dynamic binding
81     cout << endl;
82     return 0;
83  }
84
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 12 of 13).

```
85  // Make virtual function calls off a base-class pointer
86  // using dynamic binding.
87  void virtualViaPointer( const Employee *baseClassPtr )
88  {
89     baseClassPtr->print();
90     cout << " earned $" << baseClassPtr->earnings();
91  }
92
93  // Make virtual function calls off a base-class reference
94  // using dynamic binding.
95  void virtualViaReference( const Employee &baseClassRef )
96  {
97     baseClassRef.print();
98     cout << " earned $" << baseClassRef.earnings();
99  }
```

```
              Boss: John Smith earned $800.00
              Boss: John Smith earned $800.00
              Boss: John Smith earned $800.00
  Commission worker: Sue Jones earned $650.00
  Commission worker: Sue Jones earned $650.00
  Commission worker: Sue Jones earned $650.00
       Piece worker: Bob Lewis earned $500.00
       Piece worker: Bob Lewis earned $500.00
       Piece worker: Bob Lewis earned $500.00
      Hourly worker: Karen Price earned $550.00
      Hourly worker: Karen Price earned $550.00
      Hourly worker: Karen Price earned $550.00
```

Fig. 10.1 Demonstrating polymorphism with the **Employee** class hierarchy (part 13 of 13).

```
1   // Fig. 10.2: shape.h
2   // Definition of abstract base class Shape
3   #ifndef SHAPE_H
4   #define SHAPE_H
5   #include <iostream.h>
6
7   class Shape {
8   public:
9      virtual double area() const { return 0.0; }
10     virtual double volume() const { return 0.0; }
11
12     // pure virtual functions overridden in derived classes
13     virtual void printShapeName() const = 0;
14     virtual void print() const = 0;
15  };
16
17  #endif
```

Fig. 10.2 Definition of abstract base class **Shape** (part 1 of 10).

```
18  // Fig. 10.2: point1.h
19  // Definition of class Point
20  #ifndef POINT1_H
21  #define POINT1_H
22  #include "shape.h"
23
24  class Point : public Shape {
25  public:
```

```
26     Point( int = 0, int = 0 );  // default constructor
27     void setPoint( int, int );
28     int getX() const { return x; }
29     int getY() const { return y; }
30     virtual void printShapeName() const { cout << "Point: "; }
31     virtual void print() const;
32  private:
33     int x, y;   // x and y coordinates of Point
34  };
35
36  #endif
```

Fig. 10.2 Definition of class **Point** (part 2 of 10).

```
37  // Fig. 10.2: point1.cpp
38  // Member function definitions for class Point
39  #include "point1.h"
40
41  Point::Point( int a, int b ) { setPoint( a, b ); }
42
43  void Point::setPoint( int a, int b )
44  {
45     x = a;
46     y = b;
47  }
48
49  void Point::print() const
50     { cout << '[' << x << ", " << y << ']'; }
```

Fig. 10.2 Member function definitions for class **Point** (part 3 of 10).

```
51  // Fig. 10.2: circle1.h
52  // Definition of class Circle
53  #ifndef CIRCLE1_H
54  #define CIRCLE1_H
55  #include "point1.h"
56
57  class Circle : public Point {
58  public:
59     // default constructor
60     Circle( double r = 0.0, int x = 0, int y = 0 );
61
62     void setRadius( double );
63     double getRadius() const;
64     virtual double area() const;
65     virtual void printShapeName() const { cout << "Circle: "; }
66     virtual void print() const;
67  private:
68     double radius;   // radius of Circle
69  };
70
71  #endif
```

Fig. 10.2 Definition of class **Circle** (part 4 of 10).

```
72  // Fig. 10.2: circle1.cpp
73  // Member function definitions for class Circle
74  #include "circle1.h"
75
76  Circle::Circle( double r, int a, int b )
77     : Point( a, b )  // call base-class constructor
78  { setRadius( r ); }
79
```

```
80  void Circle::setRadius( double r ) { radius = r > 0 ? r : 0; }
81
82  double Circle::getRadius() const { return radius; }
83
84  double Circle::area() const
85     { return 3.14159 * radius * radius; }
86
87  void Circle::print() const
88  {
89     Point::print();
90     cout << "; Radius = " << radius;
91  }
```

Fig. 10.2 Member function definitions for class **Circle** (part 5 of 10).

```
1   // Fig. 10.2: cylindr1.h
2   // Definition of class Cylinder
3   #ifndef CYLINDR1_H
4   #define CYLINDR1_H
5   #include "circle1.h"
6
7   class Cylinder : public Circle {
8   public:
9      // default constructor
10     Cylinder( double h = 0.0, double r = 0.0,
11               int x = 0, int y = 0 );
12
13     void setHeight( double );
14     double getHeight() const;
15     virtual double area() const;
16     virtual double volume() const;
17     virtual void printShapeName() const {cout << "Cylinder: ";}
18     virtual void print() const;
19  private:
20     double height;   // height of Cylinder
21  };
22
23  #endif
```

Fig. 10.2 Definition of class **Cylinder** (part 6 of 10).

```
24  // Fig. 10.2: cylindr1.cpp
25  // Member and friend function definitions for class Cylinder
26  #include "cylindr1.h"
27
28  Cylinder::Cylinder( double h, double r, int x, int y )
29     : Circle( r, x, y )  // call base-class constructor
30  { setHeight( h ); }
31
32  void Cylinder::setHeight( double h )
33     { height = h > 0 ? h : 0; }
34
35  double Cylinder::getHeight() const { return height; }
36
37  double Cylinder::area() const
38  {
39     // surface area of Cylinder
40     return 2 * Circle::area() +
41            2 * 3.14159 * getRadius() * height;
42  }
43
44  double Cylinder::volume() const
45     { return Circle::area() * height; }
```

```
46
47 void Cylinder::print() const
48 {
49    Circle::print();
50    cout << "; Height = " << height;
51 }
```

Fig. 10.2 Member function definitions for class **Cylinder** (part 7 of 10).

```
52 // Fig. 10.2: fig10_02.cpp
53 // Driver for shape, point, circle, cylinder hierarchy
54 #include <iostream.h>
55 #include <iomanip.h>
56 #include "shape.h"
57 #include "point1.h"
58 #include "circle1.h"
59 #include "cylindr1.h"
60
61 void virtualViaPointer( const Shape * );
62 void virtualViaReference( const Shape & );
63
64 int main()
65 {
66    cout << setiosflags( ios::fixed | ios::showpoint )
67         << setprecision( 2 );
68
```

Fig. 10.2 Driver for point, circle, cylinder hierarchy (part 8 of 10).

```
69     Point point( 7, 11 );                  // create a Point
70     Circle circle( 3.5, 22, 8 );           // create a Circle
71     Cylinder cylinder( 10, 3.3, 10, 10 );  // create a Cylinder
72
73     point.printShapeName();    // static binding
74     point.print();             // static binding
75     cout << '\n';
76
77     circle.printShapeName();   // static binding
78     circle.print();            // static binding
79     cout << '\n';
80
81     cylinder.printShapeName(); // static binding
82     cylinder.print();          // static binding
83     cout << "\n\n";
84
85     Shape *arrayOfShapes[ 3 ];  // array of base-class pointers
86
87     // aim arrayOfShapes[0] at derived-class Point object
88     arrayOfShapes[ 0 ] = &point;
89
90     // aim arrayOfShapes[1] at derived-class Circle object
91     arrayOfShapes[ 1 ] = &circle;
92
93     // aim arrayOfShapes[2] at derived-class Cylinder object
94     arrayOfShapes[ 2 ] = &cylinder;
95
96     // Loop through arrayOfShapes and call virtualViaPointer
97     // to print the shape name, attributes, area, and volume
98     // of each object using dynamic binding.
99     cout << "Virtual function calls made off "
100         << "base-class pointers\n";
101
102    for ( int i = 0; i < 3; i++ )
103       virtualViaPointer( arrayOfShapes[ i ] );
104
105    // Loop through arrayOfShapes and call virtualViaReference
106    // to print the shape name, attributes, area, and volume
107    // of each object using dynamic binding.
108    cout << "Virtual function calls made off "
109         << "base-class references\n";
110
111    for ( int j = 0; j < 3; j++ )
112       virtualViaReference( *arrayOfShapes[ j ] );
113
114    return 0;
115 }
116
```

Fig. 10.2 Driver for point, circle, cylinder hierarchy (part 9 of 10).

```
117 // Make virtual function calls off a base-class pointer
118 // using dynamic binding.
119 void virtualViaPointer( const Shape *baseClassPtr )
120 {
121    baseClassPtr->printShapeName();
122    baseClassPtr->print();
123    cout << "\nArea = " << baseClassPtr->area()
124         << "\nVolume = " << baseClassPtr->volume() << "\n\n";
125 }
126
127 // Make virtual function calls off a base-class reference
128 // using dynamic binding.
129 void virtualViaReference( const Shape &baseClassRef )
130 {
131    baseClassRef.printShapeName();
132    baseClassRef.print();
133    cout << "\nArea = " << baseClassRef.area()
134         << "\nVolume = " << baseClassRef.volume() << "\n\n";
135 }
```

```
Point: [7, 11]
Circle: [22, 8]; Radius = 3.50
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00

Virtual function calls made off base-class pointers
Point: [7, 11]
Area = 0.00
Volume = 0.00

Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00

Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12

Virtual function calls made off base-class references
Point: [7, 11]
Area = 0.00
Volume = 0.00

Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00

Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
```

Fig. 10.2 Driver for point, circle, cylinder hierarchy (part 10 of 10).

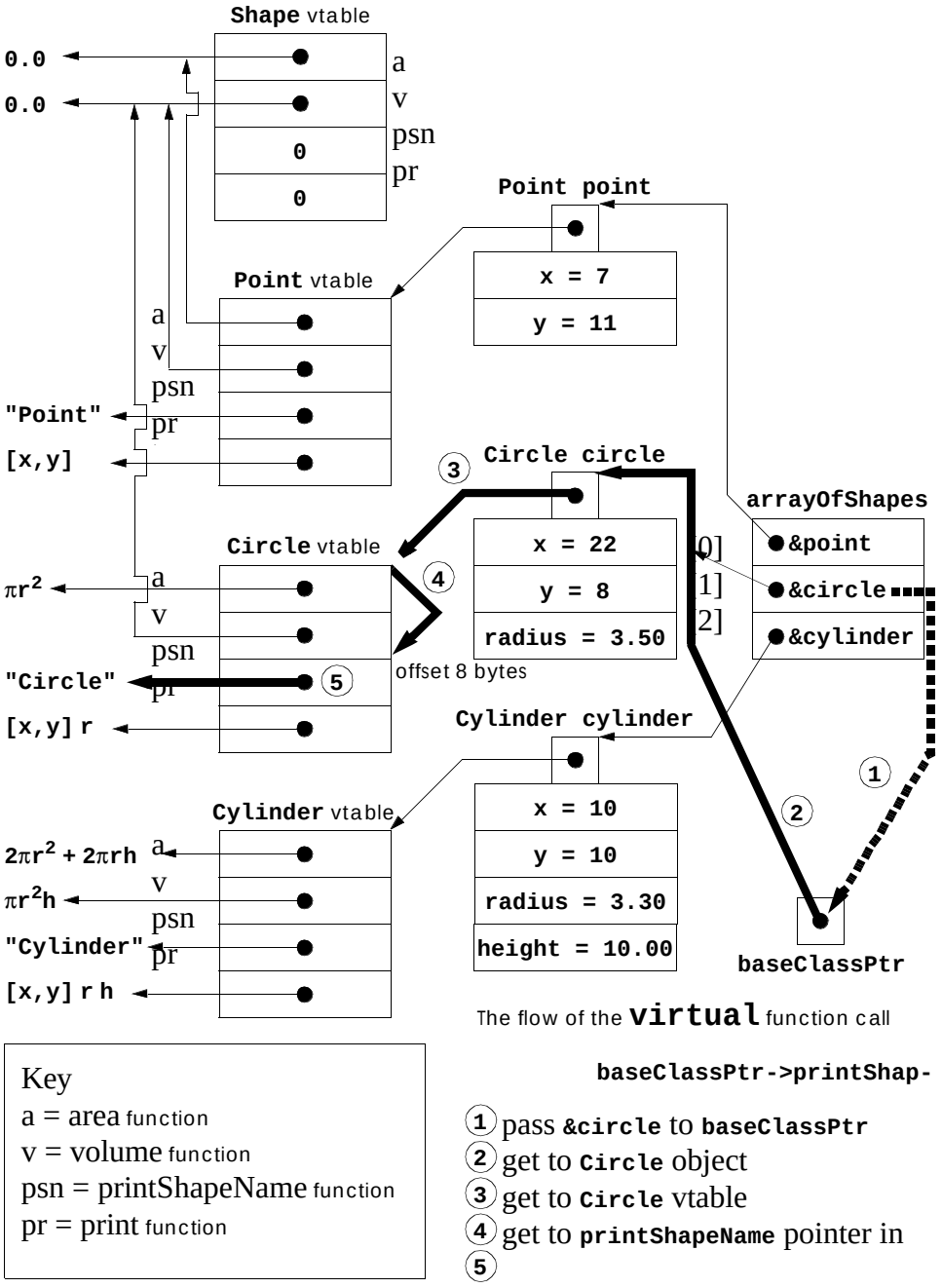

Fig. 10.3 Flow of control of a `virtual` function call.