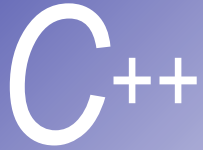


C++

Ders 5

Miras Alma



Inheritance (Miras)

- Eski sınıflardan yeni sınıflar yaratılmasıdır.
- Yeni (türetilmiş) sınıf, eski (kök) sınıfın özellik ve davranışlarını gösterir.
- Türetilmiş (Derived) sınıf, kök (base) sınıfın üye fonksiyon ve değişkenlerini miras alır.
- Bir sınıf, bir veya birden çok kök sınıftan türetilbilir.
- Miras Çeşitleri:
 - **public**: Türetilmiş nesnelere, kök sınıftan nesnelerce erişilebilir.
 - **private**: Türetilmiş nesnelere, kök sınıftan nesneler erişemez.
 - **protected**: Türetilmiş sınıflar ve friend sınıflar, kök sınıfın protected üyelerine erişebilirler.

C++

Inheritance (Miras): Kök ve Türetilmiş sınıflar

- Sıklıkla, bir nesne hem bir nesneye göre Türetilmiş sınıf (subclass), hem de bir başka nesne için aynı zamanda kök sınıfıdır (superclass).
- Bir dikdörtgen, çokgenler sınıfından türetilmişken, aynı zamanda kare sınıfının köküdür.
- Miras alma örnekleri:

| Kök (Base) sınıf | Türetilmiş (Derived) sınıf |
|------------------|---|
| Öğrenci | ÜniversiteÖğrencisi İlkokulÖğrencisi |
| Şekil | Çember Üçgen |
| Kredi | ArabaKredisi EvKredisi |
| Çalışan | FakülteÇalışanı MemurÇalışanlar |
| Hesap | ÇekHesabı MevduatHesabı |

C++

Inheritance (Miras): Kök ve Türetilmiş sınıflar

- Bir **public** miras alma tanımlanması:

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

- **CommissionWorker** sınıfı, **Employee** sınıfından türetilmiştir.
- **friend** fonksiyonlar miras alınamaz.
- Kök sınıfının Private üyelerine türetilmiş sınıfından ulaşılamaz.

C++

Protected Üyeler

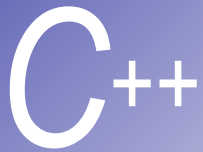
- **Protected** erişim, **public** ve **private** arasında orta seviye bir koruma sağlar.
- Türetilmiş-sınıf üyeleri, kök sınıfın **public** ve **protected** üyelerine sadece isimlerini kullanarak erişebilirler.
- Aslında, **protected** erişim, nesnenin veri korumasını (encapsulation) kırmış olur.

C++

Kök-sınıfı Pointer' larını Türetilmiş Sınıf Pointer' larına Çevirmek

- Türetilmiş bir sınıfın nesnesine kök sınıfının bir nesnesi gibi davranılabilir, fakat tersi doğru değildir.
- Ters yönde dönüşüm:
 - Kök sınıf pointer' inden türetilmiş sınıf pointer' ına bir dönüşüm ifadesi ile bu dönüşüm sağlanabilir.
 - Fakat burada pointer henüz bir nesneyi göstermiyorsa, bir nesne atandığında uygun bir nesne olması programcının sorumluluğundadır.
 - **derivedPtr = static_cast< DerivedClass * > basePtr;**

upcasting a pointer →



Kök-sınıfı Pointer' larını Türetilmiş Sınıf Pointer' larına Çevirmek

- Şimdi bir örnekle kök sınıf pointer' larından türetilmiş sınıf pointer' larına bir dönüşümün nasıl yapılacağını görelim:
- Circle sınıfı Point sınıfından türetilmiştir.
- Burada Point sınıfından bir pointer Circle sınıfından bir nesneyi gösteriyor.
- Ayrıca, Circle sınıfından bir pointer, Point sınıfından bir nesneyi göstermekte.

C++

Örnek

(pointer dönüştürme)

```
1 // Fig. 9.4: 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 v; }      // get v coordinate
17 protected:                             // accessible by derived classes
18     int x, v;                             // x and v coordinates of the Point
19 };
20
21 #endif
```


C++

Örnek (devamı)

```
22 // Fig. 9.4: 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 }
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 }
```

C++

Örnek (devamı)

```
44 // Fig. 9.4: circle.h
45 // Definition of class Circle
46 #ifndef CIRCLE_H
47 #define CIRCLE_H
48
49 #include <iostream>
50
51 using std::ostream;
52
53 #include <iomanip>
54
55 using std::ios;
56 using std::setiosflags;
57 using std::setprecision;
58
59 #include "point.h"
60
61 class Circle : public Point { // Circle inherits from Point
62     friend ostream &operator<<( ostream &, const Circle & );
63 public:
64     // default constructor
```

Circle sınıfı **Point** sınıfından public olarak türetildiği için onun **public** ve **protected** fonksiyon ve datalarına sahip olacaktır.

C++

Örnek (devamı)

```
65     Circle( double r = 0.0, int x = 0, int y = 0 );
66
67     void setRadius( double );    // set radius
68     double getRadius() const;    // return radius
69     double area() const;        // calculate area
70 protected:
71     double radius;
72 };
73
74 #endif
75 // Fig. 9.4: circle.cpp
76 // Member function definitions for class Circle
77 #include "circle.h"
78
79 // Constructor for Circle calls constructor for Point
80 // with a member initializer then initializes radius.
```

C++

Örnek (devamı)

```

81 Circle::Circle( double r, int a, int b )
82     : Point( a, b )           // call base-class constructor
83 { setRadius( r ); }
84
85 // Set radius of Circle
86 void Circle::setRadius( double r )
87     { radius = ( r >= 0 ? r : 0 ); }
88
89 // Get radius of Circle
90 double Circle::getRadius() const { return radius; }
91
92 // Calculate area of Circle
93 double Circle::area() const
94     { return 3.14159 * radius * radius; }
95
96 // Output a Circle in the form:
97 // Center = [x. y]: Radius = # ##
98 ostream &operator<<( ostream &output, const Circle &c )
99 {
100     output << "Center = " << static_cast< Point >( c )

```

Circle point' den türetildiği için point' in data üyelerine sahiptir.

Point' in consructor' u çağrılarak **point** data üyelerine başlangıç değeri atanıyor.

C++

Örnek (devamı)

```
101         << " : Radius = "
102         << setiosflags( ios::fixed | ios::showpoint )
103         << setprecision( 2 ) << c.radius;
104
105     return output;    // enables cascaded calls
106}
107// Fig. 9.4: fig09_04.cpp
108// Casting base-class pointers to derived-class pointers
109#include <iostream>
110
111using std::cout;
112using std::endl;
113
114#include <iomanip>
115
116#include "point.h"
117#include "circle.h"
118
119int main()
120{
121    Point *pointPtr = 0, p( 30, 50 );
```

C++

Örnek (devamı)

pointPtr bir **Circle *** dönüşür, ve **circlePtr** 'a atanır. **circlePtr** **Circle** olan **c** gibi davranır.

```
122 Circle *circlePtr = 0, c( 2.7, 120, 89 );
123
124 cout << "Point p: " << p << "\nCircle c: " << c << '\n';
125
```

Point p: [30, 50]

```
// Treat a Circle as a Point
pointPtr = &c; // assign address of c to pointPtr
```

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

```
cout << "\nCircle c (via *pointPtr): "
      << *pointPtr << '\n';
```

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

```
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) " << circlePtr->area() << '\n';
136
137
138 // DANGEROUS: Treat a Point as a Circle
```

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

C++

Program çıktısı

```

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 }

```

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

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

Bir **Point** nesnesi atanmış olan **pointPtr** türetilmiş-sınıf hakkında hiçbir bilgiye sahip değildir.
circlePtr'ye **Circle *** dönüşümü ile türetilmiş-sınıf bilgisi taşımayan gerçek bir kök-sınıf nesnesi tahsis edildiğinden tehlikelidir.

C++

Türetilmiş sınıfta, kök-sınıfı üyelerini değiştirmek

- Türetilmiş sınıfta aynı prototip yapısına sahip yeni bir fonksiyon oluşturulabilir. Yani aynı isimli ama içeriği değişik bir fonksiyon.
- Bu fonksiyon ismi türetilmiş sınıfta kullanılınca türetilmiş sınıftaki hali çağırılır.
- Kök sınıftaki orijinal fonksiyonu ::(scope-resolution) operatörü ile türetilmiş sınıftan kullanabiliriz.

C++

Örnek

```
1 // Fig. 9.5: emplov.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 };
15
16 #endif
17 // Fig. 9.5: employ.cpp
18 // Member function definitions for class Employee
19 #include <iostream>
20
21 using std::cout;
```

C++

Örnek(devamı)

```
22
23 #include <cstring>
24 #include <cassert>
25 #include "employ.h"
26
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 }
40
41 // Output employee name
```

C++

Örnek (devamı)

```

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 }
51 // Fig. 9.5: 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:

```

HourlyWorker public olarak **Employee** 'den türetiliyor.

HourlyWorker print fonksiyonunu yeniden tanımlayacak.

C++

Örnek (devamı)

```
64     double wage:           // wage per hour
65     double hours:          // hours worked for week
66 };
67
68 #endif
69 // Fig. 9.5: 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
78 using std::ios;
79 using std::setiosflags;
80 using std::setprecision;
81
82 #include "hourly.h"
83
```

C++

Örnek (devamı)

```

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 }
93
94 // Get the HourlyWorker's pay
95 double HourlyWorker::getPav() const { return wage * hours; }
96
97 // Print the HourlyWorker's name and pay
98 void HourlyWorker::print() const
99 {
100     cout << "HourlyWorker::print() is execut
101     Employee::print(); // call base-class
102

```

Print fonksiyonu **HourlyWorker** sınıfında tekrar yazılıyor olmasına rağmen yeni fonksiyonda hala **::** operatörü kullanılarak eskisi çağrılabilir.

C++

Program çıktısı

```
103     cout << " is an hourly worker with pay of $"
104         << setiosflags( ios::fixed | ios::showpoint )
105         << setprecision( 2 ) << getPay() << endl;
106
107// Fig. 9.5: fig09_05.cpp
108// Overriding a base-class member function in a
109// derived class.
110#include "hourly.h"
111
112int main()
113{
114     HourlyWorker h( "Bob", "Smith", 40.0, 10.00 );
115     h.print();
116     return 0;
117}
```

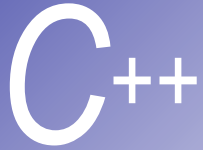
HourlyWorker::print() is executing

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

C++

public, private, ve protected miras

| Kök sınıfı üye erişim anahtar kelimesi | Miras Çeşitleri | | |
|--|--|--|--|
| | public miras | protected miras | private miras |
| Public | Derived sınıfta public 'dir. Friend fonksiyonlar, üye olmayan fonksiyonlar ve herhangi bir statik olmayan fonksiyon tarafından ulaşılabilir. | Derived sınıfta protected 'dir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından ulaşılabilir. | Derived sınıfta private 'dir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından ulaşılabilir. |
| Protected | Derived sınıfta protected 'dir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından ulaşılabilir. | Derived sınıfta protected 'dir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından ulaşılabilir. | Derived sınıfta private 'dir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından ulaşılabilir. |
| Private | Derived sınıfta erişilemezdir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından base sınıfının public veya protected üye fonksiyonları aracılığıyla ulaşılabilir. | Derived sınıfta erişilemezdir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından base sınıfının public veya protected üye fonksiyonları aracılığıyla ulaşılabilir. | Derived sınıfta erişilemezdir. Friend fonksiyonlar, herhangi bir statik olmayan fonksiyon tarafından base sınıfının public veya protected üye fonksiyonları aracılığıyla ulaşılabilir. |



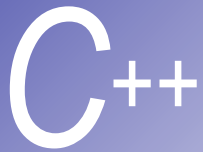
Direkt ve direkt olamayan kök sınıfı

- Direkt kök sınıfı: türetilmiş sınıf bildirimi yapılırken türetilmiş sınıfın header'ında kök sınıf (:) iki nokta kullanılarak açıkça belirtilir.

```
class HourlyWorker : public Employee  
//Employee HourlyWorker'ın direkt kök sınıfıdır.
```

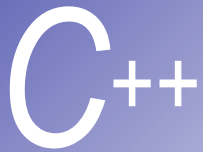
- Direkt olmayan kök sınıfı: türetilmiş sınıf header'ında gözükmeyen ama iki veya daha fazla alt sınıfın miras ile kök sınıf olmasıdır.

```
class MinuteWorker : public HourlyWorker  
//Employee MinuteWorker'ın direkt olmayan bir kök sınıfıdır.
```

Türetilmiş sınıflarda Constructor ve Destructor kullanımı

- ‘**member-initializer**’ yapısı kullanılır.
- Türetilmiş sınıf constructor’ unda kök-sınıfı constructor’ unu çağırmak için gerekir, aksi halde benimsenmiş olarak kök-sınıfı constructor’ u çağrılır.
- Kök-sınıfı constructor ve kök-sınıfı atama operatörü **derived** sınıfıyla türetilmez ama çağrılabilir.



Türetilmiş sınıflarda Constructor ve Destructor kullanımı

- Türetilmiş-sınıf constructor' u çağrılırken ilk olarak kök-sınıf constructor' u çağrılır. Eğer türetilmiş-sınıf constructor' u tanımlanmamışsa benimsenmiş olarak kök-sınıf constructor' u çağrılır.
- Kök sınıf Destructor' ü, constructor' un tam tersi şeklinde, türetilmiş-sınıf Destructor' ünden daha sonra çağırılır.

C++

Örnek

```
1 // Fig. 9.7: 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 };
13
14 #endif
15 // Fig. 9.7: point2.cpp
16 // Member function definitions for class Point
17 #include <iostream>
18
19 using std::cout;
20 using std::endl;
21
```

C++

Örnek (devamı)


```
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 }
33
34 // Destructor for class Point
35 Point::~~Point()
36 {
37     cout << "Point destructor: "
38         << '[' << x << ", " << y << ']' << endl;
39 }
```

C++

Örnek

```
40 // Fig. 9.7: 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 };
56
57 #endif
```

Circle
Point'den
türetiliyor.



C++

Örnek (devamı)

```
58 // Fig. 9.7: 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 << "]"
74 }
```

İlk değer verme yapısı kullanılıyor. ilk olarak Circle constructor Point constructor' u çağırır.

C++

Örnek (devamı)

```
75
76 // Destructor for class Circle
77 Circle::~Circle()
78 {
79     cout << "Circle destructor:  radius is "
80         << radius << " [" << x << ", " << y << "]"
81 }
82 // Fig. 9.7: fig09 07.cpp
83 // Demonstrate when base-class and derived-class
84 // constructors and destructors are called.
85 #include <iostream>
86
87 using std::cout;
88 using std::endl;
89
90 #include "point2.h"
91 #include "circle2.h"
92
```

Son olarak Circle destructor' u Point' in destructor' unu çağırır.

C++

Örnek (devamı)

```

93 int main()
94 {
95     // Show constructor and destructor call
96     {
97         Point p( 11, 22 );
98     }
99
100    cout << endl;
101    Circle circle1( 4.5, 72, 29 );
102    cout << endl;
103    Circle circle2( 10, 5, 5 );
104    cout << endl;
105    return 0;
106}

```

Point constructor: [11, 22]

Point destructor: [11, 22]

Hatırlatma: **Point** constructor **circle** nesnesi içinde **circle** constructordan önce çağrılır.

Point constructor: [72, 29]

Circle constructor: radius is 4.5 [72, 29]

Point destructor **Circle** destructor 'dan sonra çağrılır. (outside in).

Point constructor: [5, 5]

Circle constructor: [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]

C++

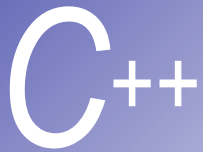
Program çıktısı

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

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

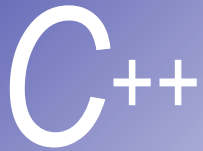
Point constructor: [5, 5]
Circle constructor: radius is 10 [5, 5]

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



Türetilmiş sınıf nesnesinin kök sınıfı nesneye dönüşümü

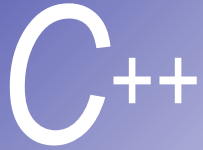
- Türetilmiş-sınıf, kök-sınıfından farklı olmasına rağmen türetilmiş-sınıfı nesne, kök-sınıfı bir nesne gibi davranabilir.
 - Çünkü türetilmiş-sınıf kök-sınıfın bütün üyelerine sahiptir. Ayrıca yeni üyelere de sahip olabilir.
 - Türetilmiş sınıftan bir nesne, Kök sınıftan bir nesneye atanabilir.
- Kök-sınıfı nesnelerse türetilmiş-sınıf gibi davranamaz.
 - Çünkü türetilmiş sınıf kök sınıftan fazla üyeye sahip olabilir.
 - Kök sınıftan bir nesne türetilmiş sınıftan bir nesneye atanamaz.
 - Ancak atama operatörünü bu atamaya izin verecek şekilde aşırı yüklememiz mümkündür.



Türetilmiş sınıf nesnesinin kök sınıfı nesneye dönüşümü

Public türü miras ile, kök ve türetilmiş sınıfı pointer ve nesneleri karıştırmanın ve eşleştirmenin dört yolu vardır:

1. Bir kök-sınıfı pointer ile bir kök-sınıfı nesneye işaret etmeye izin verilir
2. Bir türetilmiş-sınıfı pointer ile bir kök-sınıfı nesneye işaret etmeye izin verilir
3. Bir kök-sınıfı pointer ile bir türetilmiş-sınıfı nesneye işaret etmek muhtemel syntax hatası oluşturur veya kod sadece kök-sınıfı üyelerini ifade eder aksi takdirde syntax hatası oluşur
4. Bir türetilmiş-sınıfı pointer ile bir türetilmiş-sınıfı nesneye işaret etmek syntax hatası oluşturur veya türetilmiş-sınıfı pointer ilk olarak kök-sınıfına dönüştürülmelidir



Inheritance(miras) ile Yazılım Mühendisliği

- Sınıflar genelde birbirlerine benzer özellikler taşır:
 - Ortak özellikleri kök sınıfa yerleştirilir.
 - Miras alma yolu ile yeni davranışlar (behaviors) ve özellikler (attributes) türetilmiş sınıflar üzerinde oluşturulur.
- Kök-sınıfı üzerinde değişiklik yapmak:
 - Türetilmiş sınıflar, public ve protected arayüz değiştirilmediği sürece bir değişiklik gerektirmezler.
 - Ancak tekrar derlenmeye ihtiyaç duyabilirler.

Bu güçlü özellik ISV (Independent Software Vendors) için oldukça çekicidir. Böylelikle özel sınıflar lisanslı olarak üretilip nesne-programı formatında kullanıcılara sunulur

C++

Miras' a karşılık Birleştirme

Sınıflar:

Employee
BirthDate

TelephoneNumber

Yanlış:

Employee BirthDate' e
akrabadır, veya
Employee Telephone-
Number' a akrabadır

Doğru:

Employee bir
BirthDate' e sahiptir,
veya Employee bir
TelephoneNumber' a
sahiptir

- (Miras) “Bir akraba **dır**” :
Bir sınıfın diğer bir sınıftan türetilmesi ile
oluşan akrabalıktır.
[Inheritance]
- (Birleştirme) “Bir akrabalığa **sahiptir**” :
Bir sınıfın diğer bir sınıfı bir üye olarak
içermesi sonucu akrabalıktır.
[Composition]

C++

Örnek

Point



Circle



Cylinder

```

1 // Fig. 9.8: 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 };
20
21 #endif

```

Point data üyeleri
Circle 'dan ulaşılabilir
olması için
protected' dırlar

C++

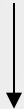
Örnek (devamı)

```
22 // Fig. 9.8: 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 }
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 }
```

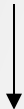
C++

Örnek

Point



Circle



Cylinder

```

1 // Fig. 9.9: 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 };
23
24 #endif
  
```

Circle data üyeleri
Cylinder 'dan
ulaşılabilir olması için
protected dırlar

C++

Örnek (devamı)

```
25 // Fig. 9.9: 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 ); }
```

C++

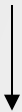
Örnek (devamı)

```
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
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 }
```

C++

Örnek (devamı)

Point



Circle

**Cylinder**

```
1 // Fig. 9.10: 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 }
```

C++

Örnek (devamı)

```
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 };
28
29 #endif
30 // Fig. 9.10: 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
```

C++

Örnek (devamı)

```
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 }
53
54 // Calculate volume of Cylinder
55 double Cylinder::volume() const
56     { return Circle::area() * height; }
57
58 // Output Cylinder dimensions
59 ostream &operator<<( ostream &output, const Cylinder &c )
60 {
```

Circle::area()
yeniden yazıldı.

C++

Örnek (devamı)

```
61     output << static_cast< Circle >( c )
62         << "; Height = " << c.height;
63
64     return output;    // enables cascaded calls
65 }
66 // Fig. 9.10: fig09 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 );
```

C++

Örnek (devamı)

```

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";
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 "t
100 cout << "\nCylinder printed as a Point is: "
101     << pRef << "\n\n";
102

```

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

pRef cyl 'in bir **Point** olduğunu sanıyor. Bu sebeple point gibi ekrana yazılır.

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]

C++

Örnek

```

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()
107
108 return 0;
109}

```

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

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

The new location, radius, and height of cyl
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

Circle nesnesine Referans
circleRef cyl 'in bir Circle olduğunu sanıyor. Bu sebele point gibi ekrana yazılır

C++

Çoklu Miras

- Türetilmiş-sınıfın bir çok kök-sınıftan miras almasıdır
- Yazılımın yeniden kullanımı açısından faydalıdır, ama nesnenin karmaşık bir yapı almasına neden olabilir

C++

Örnek

```
1 // Fig. 9.11: base1.h
2 // Definition of class Base1
3 #ifndef BASE1_H
4 #define BASE1_H
5
6 class Base1 {
7 public:
8     Base1( int x ) { value = x; }
9     int getData() const { return value; }
10 protected:    // accessible to derived classes
11     int value;  // inherited by derived class
12 };
13
14 #endif
15 // Fig. 9.11: base2.h
16 // Definition of class Base2
17 #ifndef BASE2_H
18 #define BASE2_H
```

C++

Örnek (devamı)

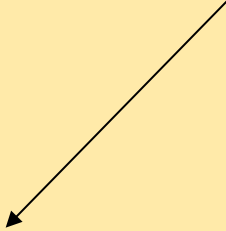
```
19
20 class Base2 {
21 public:
22     Base2( char c ) { letter = c; }
23     char getData() const { return letter; }
24 protected:      // accessible to derived classes
25     char letter;  // inherited by derived class
26 };
27
28 #endif
29 // Fig. 9.11: derived.h
30 // Definition of class Derived which inherits
31 // multiple base classes (Base1 and Base2).
32 #ifndef DERIVED_H
33 #define DERIVED_H
34
35 #include <iostream>
36
```

C++

Örnek (devamı)

```
37 using std::ostream;
38
39 #include "base1.h"
40 #include "base2.h"
41
42 // multiple inheritance
43 class Derived : public Base1, public Base2 {
44     friend ostream &operator<<( ostream &, const Derived & );
45
46 public:
47     Derived( int, char, double );
48     double getReal() const;
49
50 private:
51     double real;    // derived class's private data
52 };
53
54 #endif
```

Derived, Base1 ve Base2 den türetilmiştir.



C++

Örnek (devamı)

```
55 // Fig. 9.11: derived.cpp
56 // Member function definitions for class Derived
57 #include "derived.h"
58
59 // Constructor for Derived calls constructors for
60 // class Base1 and class Base2.
61 // Use member initializers to call base-class constructors
62 Derived::Derived( int i, char c, double f )
63     : Base1( i ), Base2( c ), real ( f ) { }
64
65 // Return the value of real
66 double Derived::getReal() const { return real; }
67
68 // Display all the data members of Derived
69 ostream &operator<<( ostream &output, const Derived &d )
70 {
71     output << "    Integer: " << d.value
72           << "\n Character: " << d.letter
73           << "\nReal number: " << d.real;
74
75     return output;    // enables cascaded calls
76 }
```

C++

Örnek (devamı)

```
77 // Fig. 9.11: fig09_11.cpp
78 // Driver for multiple inheritance example
79 #include <iostream>
80
81 using std::cout;
82 using std::endl;
83
84 #include "base1.h"
85 #include "base2.h"
86 #include "derived.h"
87
88 int main()
89 {
90     Base1 b1( 10 ), *base1Ptr = 0; // create Base1 object
91     Base2 b2( 'Z' ), *base2Ptr = 0; // create Base2 object
92     Derived d( 7, 'A', 3.5 );      // create Derived object
93
94     // print data members of base class objects
95     cout << "Object b1 contains integer " << b1.getData()
96         << "\nObject b2 contains character " << b2.getData()
97         << "\nObject d contains:\n" << d << "\n\n";
```

C++

Örnek (devamı)

Object b1 contains integer 10
Object b2 contains character Z
Object d contains:
 Integer: 7
 Character: A
 Real number: 3.5

```

98
99 // print data members of derived class object
100 // scope resolution operator resolves getData ambiguity
101 cout << "Data members of Derived can be"
102     << " accessed individually:"
103     << "\n    Integer: " << d.Base1::getData()
104     << "\n    Character: " << d.Base2::getData()
105     << "\nReal number: " << d.getReal() << "\n\n";
106
107 cout << "Derived can be treated as an "
108     << "object of either base class:\n"
109
110 // treat Derived as a Base1 object
111 base1Ptr = &d;
112 cout << "base1Ptr->getData() yields "
113     << base1Ptr->getData() << '\n';
114
115 // treat Derived as a Base2 object
116 base2Ptr = &d;

```

Data members of Derived can be accessed individually:
 Integer: 7
 Character: A
 Real number: 3.5

d nesnesi base1 nesnesi gibi davranır

d base2 gibi davranan bir nesnedir

base1Ptr->getData() 7 yi verir

Derived can be treated as an object of either base class:

C++

Örnek

```
117     cout << "base2Ptr->getData() yields "  
118         << base2Ptr->getData() << endl;  
119  
120     return 0;  
121 }
```

base2Ptr->getData() yields A

Object b1 contains integer 10
Object b2 contains character Z
Object d contains:

Integer: 7
Character: A
Real number: 3.5

Data members of Derived can be accessed individually:

Integer: 7
Character: A
Real number: 3.5

Derived can be treated as an object of either base class:

base1Ptr->getData() yields 7
base2Ptr->getData() yields A

C++

Sanal Fonksiyonlar ve Çoklubenzeşim

- Daha kolay genişletilebilen sistemleri tasarlamayı ve gerçekleştirmeyi sağlar.
- Hiyerarşide var olan bütün sınıf nesnelerince kullanılabilen genel yazılmış programları kullanarak, algoritmanın çalışması esnasında hiç çaba sarf etmeden veya çok küçük ilavelerle sınıflar ilave etmeyi mümkün kılar.

C++

Sanal Fonksiyonlar

- Sanal fonksiyonlar:
- **Daire, Üçgen, Dörtgen, Kare** vs. gibi bir şekiller sınıfının kümesini gözönüne alalım.
- Her şekil kendisine ait tek çizim fonksiyonuna sahiptir, ama bunları çağırmak **Shape** kök-sınıfının çizim fonksiyonu olan *draw* fonksiyonunu çağırarak mümkün olabilir.
 - Derleyici hangisini çağıracağını dinamik olarak belirler (yani program çalışırken belirlenir)
- Kök-sınıfında *draw* fonksiyonunu sanal olarak tanımlayalım.
- Her türetilmiş sınıf içinde *draw* fonksiyonunu tekrar tanımlayalım (override).

C++

Sanal Fonksiyonlar

- Bir Sanal Fonksiyon deklarasyonu:
 - Kök-sınıfındaki fonksiyon prototiplerinden önce

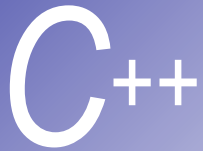
```
virtual void draw() const;
```

Shape kök-sınıfında tanımlanır.

- Bir türetilmiş sınıf nesnesine işaret eden bir kök-sınıf pointer'ı doğru **draw** fonksiyonunu çağırarak:

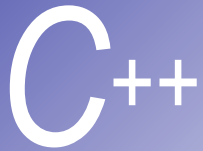
```
ShapePtr->draw() ;
```

- Eğer türetilmiş sınıfta bir sanal fonksiyon tanımlı değilse, fonksiyon kök-sınıftan miras alınır.



Sanal Fonksiyonlar

- Sanal bir fonksiyona “.” üye-seçim-operatörü ve spesifik nesne adıyla bir referans yapıldığında, referans derleme esnasında tanımlanır ve aranan sanal fonksiyon bu nesnenin sınıfı için tanımlanır. Buna “statik-bağlantı” denir
- **ShapePtr->draw()** ;
 - Derleyici dinamik bağlantı uygular.
 - Fonksiyon çalışma sırasında belirlenir.
 - Kök-sınıfı pointer kullanarak türetilmiş-sınıf nesnesi referans edersek, draw fonksiyonu devreye sokulmuş olur ve program türetilmiş- sınıflar için dinamik olarak doğru sınıfın fonksiyonunu seçecektir
- **ShapeObject.draw()** ;
 - Derleyici statik bağlantı uygular.
 - Fonksiyon derleme sırasında belirlenir.



Soyut ve Somut Sınıflar

Programcının herhangi bir şekilde bir nesneyi oluşturmaya gerek kalmaksızın soyut sınıflar üzerinden bu işlem gerçekleştirilir

Soyut kök sınıfları:

TwoDimensionalShape
ThreeDimensionalShape

Somut Sınıflar:

Square, Circle, Triangle
Cube, Sphere, Cylinder

- **Soyut (Abstract) sınıflar**
 - Tek amaç diğer sınıflar için bir kök sınıfı sağlamaktır.
 - Gerçek nesneler tanımlamak için çok geneldirler. Bir nesne türetebilmek için daha spesifik olmak gerekir.
 - Pointer ve referansları olabilir.
- **Somut (Concrete) Sınıflar**
 - Örnek nesneleri olan sınıflardır.
 - Gerçek nesneler oluşturmak için nesneye yönelik özellikler sağlarlar.
- **Soyut sınıf oluşturma**
 - Bir sınıfın bir yada birden fazla sanal fonksiyonunu sifıra eşitleyerek saf (pure) fonksiyon tanımlanabilir.
 - Bir saf sanal fonksiyon örneği:

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