KALITIM

Binnur Kurt
kurt@ce.itu.edu.tr

Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi

Sınıf Yapısı

4 Kalıtım

Çok Şekillilik

4

KALITIM

C++'ın yazılan kodun yeniden kullanılabilir olmasını sağlayan mekanizması kalıtımdır. Yeniden kullanılabilirlikten, bir sınıfın alınıp bir başka yazılım uygulamasında da kullanılabilmesini anlıyoruz. Bu özellik yazılım geliştirme çevrimini kısaltırken aynı zamanda yazılımın daha gürbüz olmasını sağlayacaktır.

Tarihçe :

• Kopyala ve Yapıştır + Uyarla + Hata Ayıkla,

• Tekrar tekrar kullanılan fonksiyonlar için kütüphaneler oluştur,

• Yeni yazılım projesi ≡

Kütüphane Fonksiyonları (Uyarla + Hata Ayıkla)

4

KALITIM

Çözüm :

• Sınıf Kütüphaneleri

Problemleri daha iyi modellediklerinden yeni bir proje için kullanılmak istenildiklerinde daha az değiştirilme ihtiyacı duyarlar.

• C++ bir sınıfın kodunu değiştirmeden eklentiler yapmamıza olanak tanır. Bu kalıtım yolu ile bir temel sınıftan yeni bir sınıf türetilmesi şeklinde olur. **türetilen sınıf** ile **temel sınıf** arasında "is a" şeklinde bir hiyerarşik ilişki sözkonusudur.

• Bir temel sınıftan türetilen sınıfı belirtmek için türetilen sınıf adından sonra ":" konup temel sınıf adı yazılır.

4

KALITIM

```
class teacher {       // temel sınıf
  public:
    char *Name;
    int Age,numberOfStudents;
    void setName (char *newName){Name=newName;}
};


class principal : public teacher {  // turetilmis sinif
    char *schoolName;
    int numberOfTeachers;
  public:
    void setSchool(char *name){schoolName=name;}
};
```


4

KALITIM

principal is a teacher

*temel sınıf ile türetilmiş sınıf arasında **is a** ilişkisi vardır*

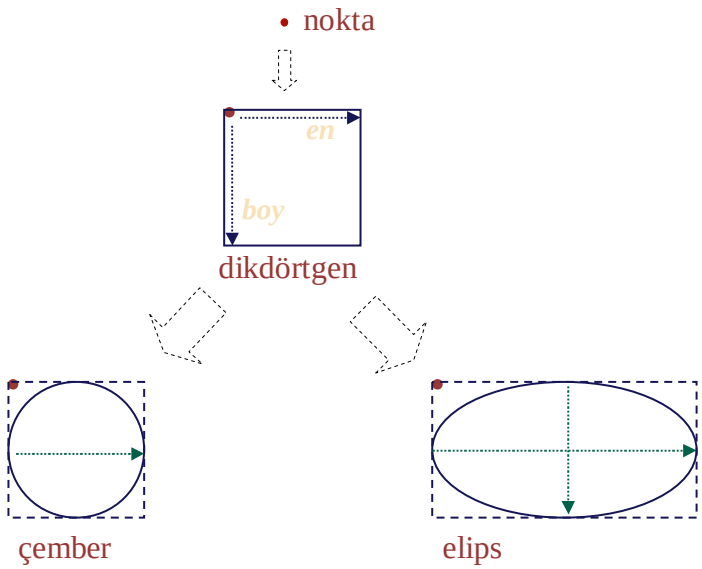
principal *(türetilmiş sınıf)*

teacher *(temel sınıf)*
- Name,
- Age,
- numberOfStudents
- setName(char *)

SchoolName
numberOfTeachers
setSchool(char *)

```
void main()
{
  teacher t1;
  principal p1;
  p1.setName(" Principal 1");
  t1.setName(" Teacher 1");
  p1.setSchool(" Elementary School");
}
```

4 KALITIM



4 KALITIM

```
class point {       // temel sınıf
    int x,y;
  public:
    void setPoint(int X,int Y){x=X;y=Y;}
};

class rectangle : public point {  // turetilmis sinif
     int Width,Height ;
  public:
     void setSize(int w,int h){Width=w;Height=h;}
};
class circle : public rectangle {  // turetilmis sinif
  public:
    void setRadius(int r){Width=Height=r;}
};
```

4 KALITIM

C++'ın yazılan kodun *yeniden kullanılabilir* olmasını sağlayan mekanizma *kalıtım*dır. Yeniden kullanılabilirlikten, bir sınıfın alınıp bir başka yazılım uygulamasında da (*aynen yada değişikliklerle birlikte*) kullanılabilmesini anlıyoruz. Bu özellik yazılım geliştirme sürecini kısaltırken aynı zamanda yazılımın daha *gürbüz* olmasını sağlayacaktır:

İstemlerin Analizi
Sistem Analizi
Tasarım
Kodlama
Test
Bakım

4 KALITIM

Türetilmiş Sınıfta Üyelerin Yeniden Tanımlanabilmesi

Bazı durumlarda, temel sınıftaki bir fonksiyonu, türetilmiş sınıfta yeniden tanımlamak gerekebilir:

```
class teacher{       // Base class
  public:
    char *Name;
    int Age,numberOfStudents;
    void setName (char *new_name){Name=new_name;}
    void print();
};

void teacher::print()          // Print method of teacher class
{
   cout <<"Name: "<< Name<<"  Age: "<< age<< endl;
   cout << "Number of Students: " <<numberOfStudents << endl;
}
```

4 KALITIM

example13.cpp

```
class principal : public teacher{  // Derived class
  public:
    char *schoolName;
    int numberOfTeachers;
    void setSchool(char *s_name){schoolName=s_name;}
    void print();      // Print function of principal class
};

void principal::print()          // Print method of principal class
{
   cout <<"Name: "<< Name<<"  Age: "<< Age<< endl;
   cout << Number of Students: " <<numberOfStudents << endl;
   cout <<"Name of the school: "<< schoolName << endl;
}
```

Bu durumda principal sınıfı içinde tanımladığımız yeni print() fonksiyonu temel sınıfta tanımlı print() fonksiyonu üzerine yazacaktır. Eğer temel sınıftaki print() fonksiyonuna erişilmek istenirse :: operatörü kullanılarak teacher::print() yazılır.

4 KALITIM

Örnek

```
class A{
 public:
   int ia1,ia2;
   void fa1();
   int fa2(int);
};
```

```
class B: public A{
 public:
   float ia1;            // overrides ia1
   float fa1(float);   // overrides fa1
};
```

```
void main(){
    B b;
    int j=b.fa2(1);
    b.ia1=4;                    // B::ia1
    b.ia2=3;                    // A::ia2   if  ia2 is public in A
    float y=b.fa1(3.14);  // B::fa1
    b.fa1();  // ERROR  fa1 function in B hides  the function of A
    b.A::fa1();    // OK
    b.A::ia1=1;   // OK
}
```

4 KALITIM

Erişim Denetimi

Hatırlatma: Bir sınıf üyesi (*sınıf içerisindeki*) diğer tüm üyelere erişebilir. O sınıftan bir nesne ise sadece **public** ile tanımlı üyelere erişebilir.

Kalıtım mekanizmasında, türetilmiş sınıf üyelerinin, temel sınıf üyelerine erişimi nasıl denetlenebilir?

Kural : türetilmiş sınıf üyeleri temel sınıfın

public ve **protected**

ile tanımlanmış üyelerine erişebilir.

4 KALITIM

Erişim	Sınıf İçinden Erişim	Türetilmiş sınıftan Erişim	Dışarıdan Erişim
public	evet	evet	evet
protected	evet	evet	hayır
private	evet	hayır	hayır

Genel olarak, üyeleri **private** tanımlamak uygun olacaktır. Böylelikle dışarıdan bir fonksiyonun yanlışlıkla üyenin değerini değiştirmesi olasılığı ortadan kaldırılmış olur. Temel sınıf tasarlanırken olabildiğince **protected** kullanılmasından kaçınılmalıdır. Yeni sınıflar kalıtım yoluyla türetilerek genişletildikçe, üst sınıfların temel sınıf üyelerine erişimi (*karmaşıklığı*) önlenmiş olur. Böylelikle daha kararlı ve güvenilir sınıflar gerçekleştirilebilinir.

KALITIM

```
class teacher{       // Base class
  private:
    char *Name;
  protected:
    int Age,numberOfStudents;
  public:
    void setName (char *name){Name=name;}
    void print();   };

class principal : public teacher{  // Derived class
    char *schoolName;
    int numberOfTeachers;
  public:
    void setSchool(char *name){schoolName=name;}
    void print();
    int getAge(){ return Age;}
    char* getName(){ return Name;}  };
```


KALITIM

```
void main()
{
   teacher t1;
   principal p1;

    t1.numberOfStudents=100;
    t1.setName("Sema Catir");
    p1.setSchool("Halide Edip Adivar Lisesi");
 }
```

public Kalıtım

Kalıtım ile bir sınıf türetilirken, genellikle public takısı kullanılır:

```
class Base
  { };
class Derived : public Base {
```

Bu şekilde türetilen bir sınıfta temel sınıfın üyelik tanımlamaları değişmez. Örneğin temel sınıfın public üyeleri aynı zamanda türetilen sınıfın da public üyeleri olacaktır.

private Kalıtım

```
class Base
  { };
class Derived : private Base {
```

Buna *private kalıtım* denir. Temel sınıfın public üyeleri türetilen sınıfın private üyeleri olur. Bunun sonucu olarak türetilmiş sınıfa ait nesneler temel sınıfın hiçbir elemanına erişemezler. Türetilen sınıfın üyeleri temel sınıfın public and protected tanımlı üyelerine erişebilir.

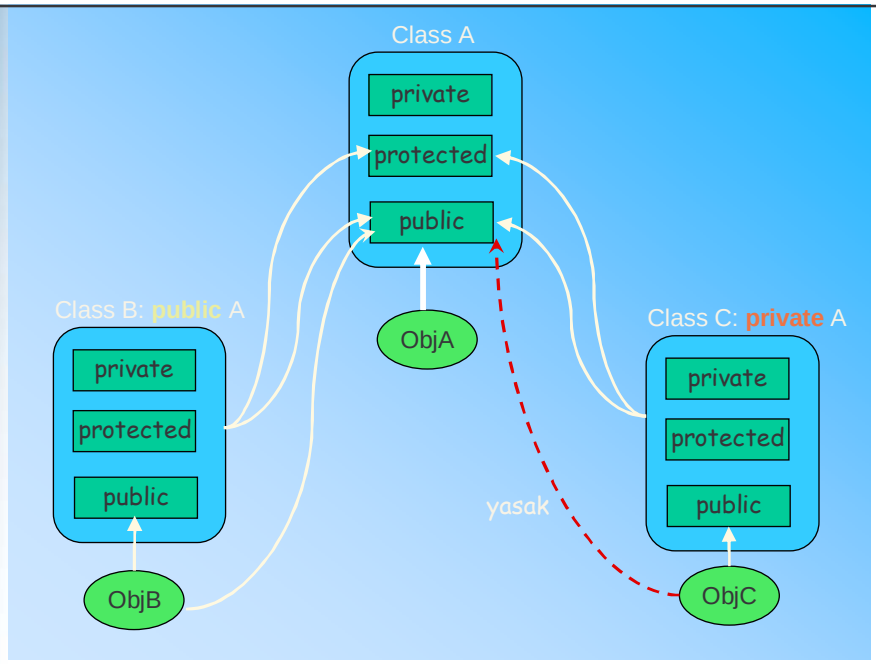
4 KALITIM

Türetilmiş Sınıfta Erişimin Yeniden Tanımlanması

Temel sınıfın public tanımlı üyelerine erişim türetilmiş sınıfta yeniden tanımlanabilir.

```
Class Base{
    public:
      void f();
};
class Derived : private Base{   // All members of Base are private now
   int i;
 public:
   Base::f();          //   f() is public again
   void fb1();
};
```


4 KALITIM



Kalıtım ile Aktarılamayan Metodlar

Temel sınıfta tanımlı bir metod, eğer işlev yükleme yapılmamış ise, otomatik olarak türetilen sınıf üyelerinin kullanımına aktarılır. Ancak bazı özel fonksiyonlar, kalıtım ile türetilmiş sınıfa aktarılmazlar:

- İşlev yüklenmiş = operatörü

  İşlev yüklenmiş atama operatörünün amacını hatırlayınız !

- Kurucu Fonksiyonlar

  Temel sınıfın kurucu fonksiyonu türetilmiş sınıfın kurucu fonksiyonu değildir.

- Yokedici Fonksiyonlar

  Temel sınıfın yokedici fonksiyonu türetilmiş sınıfın yokedici fonksiyonu değildir.

Kurucu Fonksiyonlar ve Kalıtım

Türetilmiş sınıftan bir nesne yaratıldığında, temel sınıfa ait kurucu fonksiyon türetilmiş sınıfa ait kurucu fonksiyondan önce çağrılır. Temel sınıf üyeleri türetilmiş sınıfın bir alt parçası olduğundan üst parça oluşturulmadan önce alt parçalara ait üyelerin yapılandırılması zorunluluğu vardır.

example15.cpp

```
class teacher{       // turetilmis sinif
   char *Name;
   int Age,numberOfStudents;
 public:
   teacher(char *newName){Name=newName;}  // temel sinif kurucusu
};
```

```
class principal : public teacher{  // turetilmis sinif
   int numberOfTeachers;
 public:
   principal(char *, int );     // // turetilmis sinif kurucusu
};
```

```
principal::principal(char *newName,int numOT):teacher(newName)
{
    numOfTeachers=numOT;
}

void main() {
   principal p1("Sema Catir",20);
}
```

Eğer temel sınıf, parametre alan bir kurucu fonksiyona sahip ise türetilmiş sınıfa ait kurucu fonksiyon, temel sınıf kurucu fonksiyonunu, uygun parametreler ile çağıracak bir kurucuya sahip olmalıdır.

example16.cpp

Yokedici Fonksiyonlar ve Kalıtım

Yokedici fonksiyonlar nesnenin erimi dışına çıkıldığında otomatik olarak çağırılırlar. Kalıtım ile türetilmiş sınıflarda yokedici fonksiyonların çağırılış sırası kurucu fonksiyonların çağırılış sırasının tersi şeklindedir.

Bu durumda ilk olarak türetilmiş sınıfın yokedici fonksiyonu çağırılacaktır.

example17.cpp

```
#include <iostream.h>
class B {
  public:
    B() { cout << "B constructor" << endl; }
   ~B() { cout << "B destructor" << endl; }
};
class C : public B {
  public:
    C() { cout << "C constructor" << endl; }
   ~C() { cout << "C destructor" << endl; }
};
void main(){
   cout << "Start" << endl;
   C ch;      // create a C object
   cout << "End" << endl;
}
```


Atama İşlevi ve Kalıtım

Temel sınıfın atama işlevi türetilen sınıfın atama işlevi olamaz.

```
class IntegerArray {
 protected:
   int size;
   int *contents;
 public:
   IntegerArray & operator=(const IntegerArray &);  // atama islevi
   :       // diger metodlar
};
IntegerArray & IntegerArray::operator=(const IntegerArray &s){
   size = s.size;
   contents = new int[size]; delete[] contents ;
   memcpy(contents,s.contents.contents,size*sizeof(int));
   return *this;
}
```

4 KALITIM

```
class ExtArray : public IntegerArray {
  int     size;
  int    *dim2;
 public:
  ExtArray & operator=(const ExtArray &);
  :       // DigerMetodlar
};

ExtArray & ExtArray::operator=(const ExtArray &s) {
   size = s.size; delete[] dim2 ;
   dim2 = new int[size];
   memcpy(dim2,s.dim2,size*sizeof(int));
   IntegerArray::size = s.IntegerArray::size; delete[] contents ;
   contents = new int[IntegerArray::size];
   memcpy(contents,s.contents,IntegerArray::size*sizeof(int));
   return *this;
 }
```


Örnek: Kurucu Zinciri

```
class A {
  private:
    int x;
    float y;
  public:
    A(int i, float f) :
    x(i), y(f)        // initialize A
    { cout << "Constructor A" << endl;  }
    void display() {
     cout << x << ", " << y << "; "; }
};
```

```
class B : public A {
  private:
    int v;
    float w;
  public:
    B(int x, float y, int i2, float f2) :
    A(x, y),          // initialize A
    v(i2), w(f2)      // initialize B
    { cout << "Constructor B" << endl; }
    void display(){
      A::display();
      cout << v << ", " << w << "; ";    }
};
```

```
class C : public B {
  private:
    int z;
    float t;
  public:
  C(int x,float y, int v,float w,int i3,float f3) :
    B(i1, f1, i2, f2),              // initialize B
      z(i3), t(f3)                  // initialize C
    { cout << "Constructor C" << endl; }
    void display() {
      B::display();
      cout << z << ", " << t;
      }
};
```

```
void main() {
  C c(1, 1.1, 2, 2.2, 3, 3.3);
  cout << "\nData in c = ";
  c.display();
}
```

4 KALITIM

Örnek: Sınıf ve Kurucu Zinciri — Açıklama

C sınıfı B sınıfından ve B sınıfı da A sınıfından türetilmiştir. Her sınıf kendi ve alt sınıflardaki kurucu fonksiyonlarına uygun sayıda parametre almakta ve aktarmaktadır: A sınıfı kurucu fonksiyonu iki, B sınıfı kurucu fonksiyonu dört (ikisi A sınıfı için) ve C sınıfı kurucu fonksiyonu (A ve B sınıfları kurucuları için ikişer parametre) altı parametre almaktadır.

main() fonksiyonunda C sınıfından c adında bir nesne tanımlayıp 6 adet başlangıç değeri verdik. Böylelikle tüm alt sınıflara uygun başlangıç değeri verildi.

```
C(int x, float y,int v, float w, int i3, float f3) :
                A(x, y),        // error: can't initialize A
                y(i3), z(f3)    // initialize C
 { }
```


4 KALITIM

int ve float gibi basit veri tipine sahip sınıf üyelerine aşağıdaki gibi başlangıç değeri verilebilir :

```
class  A {
  int i1,i2;
  A(int new1, int new2): i1(new1),i2(new2) {
     ...
   }
};
```

Ancak bu bir kalıtım uygulaması değildir.

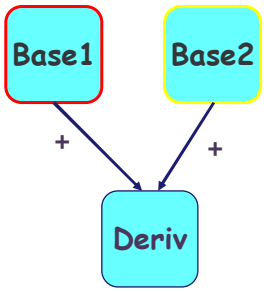
Çoklu Kalıtım

bir sınıfın birden fazla temel sınıftan türetilmesi

```
class Base1{      // Base 1
 public:
   int a;
   void fa1();
   char *fa2(int);
};
```

```
class Base2{      // Base 2
 public:
   int a;
   char *fa2(int, char);
   int fc();
};
```

```
class Deriv : public Base1 , public Base2{
public:
    int a;
    float fa1(float);
    int fb1(int);
};
```




```
void main()
{
 Deriv d;
 d.a=4;                    //Deriv::a
 float y=d.fa1(3.14);      //Deriv::fa1
 int i=d.fc();             // Base2::fc
}
```

```
  char * c=d.fa2(1);
```

Ataması geçerli değildir.
Kalıtım ile yeniden tanımlanan fonksiyonlara işlev yüklenemez.
Geçerli kullanım :

```
  char * c=d.Base1::fa2(1);
```

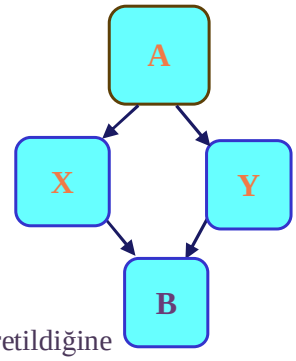
yada

```
  char * c=d.Base2::fa2(1,"Hello");
```

4 KALITIM

Tekrarlı Kalıtım

```
class A
  {  };
class X : public A
  {  };
class Y : public A
  {  };
class B : public X, public Y
  {  };
```



Hem **X** hem de **Y** sınıflarının **A** sınıfından türetildiğine dikkat ediniz. **B** sınıfı ise çoklu kalıtım ile **X** ve **Y** sınıflarından türetilmiştir.

Bu durumda **A** sınıfı hem **X** ve hem de **Y** sınıflarında ortak olduğundan **B** sınıfı iki adet **A** alt sınıfına sahiptir – biri **X** diğeri **Y** sınıflarından.

4 KALITIM

Ayrıca **A** sınıfında aşağıda verildiği gibi bir int tipinde üyeye sahip olsun.

```
class A {
  protected:
    int data;
};
class B : public X, public Y  {
  public:
    void Cfunc() {
      int temp = data;  // error: ambiguous
     }
};
```

Derleyici, "X sınıfından gelen **data**'yı mı? yoksa Y sınıfından gelen **data**'yı kullanmalı?" belirsizliği nedeni ile hata verecektir.

4 KALITIM

Çözüm : Sanal Sınıflar

Bu problem **virtual** anahtar sözcüğü kullanılarak çözülebilir.

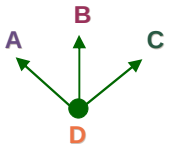
```
class A
  {  };
class X : virtual public A
  {  };
class Y : virtual public A
  {  };
class B : public X, public Y
  {  };
```

virtual anahtar sözcüğü derleyiciye kalıtım ile alt sınıflardan türetilen alt nesnelerden sadece birinin kullanmasını söyler. Ancak bu çözümde burada detaylı olarak duramayacağımız bazı karmaşık durumlarda yeni belirsizlikler getirebilmektedir.

Genel olarak çoklu kalıtımdan kaçınmalısınız. Ancak C++'da deneyimli iseniz, çoklu kalıtımın gerekli olduğu durumlarda kullanmanız çözümü kolaylaştıracaktır.

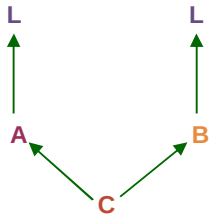
```
class Base
{
   public:
      int a,b,c;
 };
class Derived : public Base
 {
   public:
      int b;
 };
class Derived2 : public Derived
 {
   public:
      int c;
 };
```


```
class A {
   ...
};
class B {
   ...
};
class C {
   ...
 };
class D : public A, public B, private C {
   ...
 };
```



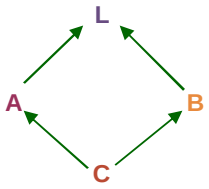
4 KALITIM

```
class L {
    public:
      int next;
};
class A : public L {
    ...
};
class B : public L {
    ...
};
class C : public A, public B {
   void f() ;
   ...
 };
```



4 KALITIM

```
class L {
    public:
      int next;
};
class A : virtual public L {
    ...
};
class B : virtual public L {
    ...
};
class C : public A, public B {
   ...
 };
```



4

KALITIM

```
class B {
     ...
};
class X : virtual public B {
    ...
};
class Y : virtual public B {
    ...
};
class Z : public B {
   ...
 };
class AA : public X, public Y , public Z {
   ...
 };
```