MALATYA
TURGUT ÖZAL ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ
ALGORİTMA VE PROGRAMLAMA 2

ÇOK BİÇİMLİLİK ve SANAL SINIFLAR

TÜRETİLMİŞ SINIFTA TEMEL SINIF ÜYELERİNİN YENİDEN TANIMLANMASI

- Türetilmiş sınıfta temel sınıfın elemanları yeniden tanımlanabilir. Bu durumda öncelik türetilmiş sınıfı elemanına geçer.

```cpp
#include <iostream>
#include <string>
using namespace std;

class calisan{
    public:
        string ad,soyad;
        calisan(){};
        void isim(string a,string b){
            ad=a;
            soyad=b;
        }
};

class muhendis:private calisan{
    int maas;
    public:
        muhendis(){};
        void muhendis_maas(int m){
            maas=m;
        }

        void isim(string x,string y){
            calisan::isim(x,y);
        }

        void muhendis_listele(){
            cout<<"Ad:"<<ad<<endl;
            cout<<"Soyad:"<<soyad<<endl;
            cout<<"Maas:"<<maas<<endl;
            cout<<"----------------"<<endl;
        }
};
```

```cpp
int main() {
    muhendis muh1,muh2;
    muh1.isim("Asiye","Kahraman");
    muh2.isim("Mehmet","Dogan");
    muh1.muhendis_maas(3500);
    muh2.muhendis_maas(3300);
    muh1.muhendis_listele();
    muh2.muhendis_listele();

    return 0;
}
```

```
C:\Users\monste
Ad:Asiye
Soyad:Kahraman
Maas:3500
----------------
Ad:Mehmet
Soyad:Dogan
Maas:3300
----------------
```

- İsim() fonksiyonu calisan sinifinin elemanıdır ve biz bunu mühendis sınıfı içinde yeniden tasarladık. Fonksiyonu aynı isimde mühendislik sınıfı içindede kullandık.
- Muhendislik içindeki isim() fonksiyonu içinde ise gelen parametrelerle beraber **calisan::isim(x,y)** şeklinde temel sınıfın isim() fonksiyonunda gönderdik.
- İlla calisan sinifinin isim fonksiyonuna göndermek şart değil. İstersek mühendislik içindeki isim fonksiyonunu istediğimiz gibi doldurabilirdik.
- ÖRNEĞİN:

```cpp
void isim(string x,string y){
    ad=x;
    soyad=y;
}
```

SANAL FONKSİYONLAR VE ÇOK BİÇİMLİLİK

- Kalıtımda birbirleriyle ilişkili sınıfların her birinde yeniden tanımlamış fonksiyonlarının tek bir sinifin nesnesiyle çağrılmasına **çok biçimlilik** denir. Burada çok biçimliliğin önemi; temel sınıftaki bir üye fonksiyonun türetilmiş sınıflarda yeniden tanımlanmasındadır.
- Bir anlamda temel sınıftaki bir fonksiyonunun türetilmiş diğer siniflar için değişik biçimlerde tanımlanmıştır. Çok biçimlilikte esas olan temel sınıftaki fonksiyondur ve çağrılan da odur.
- Çok biçimlilikte fonksiyonu çağıran nesnenin türü ne ise çağrılan fonksiyon o sınıfın fonksiyonudur.
- Çok biçimlilikte temel sınıfta ve bu sınıftan türetilmiş diğer sınıflarda bulunan ortak fonksiyonun çağrılmasında, **temel sınıf nesne göstericisine (işaretçi-pointer)** yapılan atama işlemi çok önemlidir.

SANAL FONKSİYONLAR VE ÇOK BİÇİMLİLİK

- Bu şekilde sınıf tanımlaması yapılan A temel sınıfından B sınıf türetilmiştir. Ana fonksiyon içerisinde tanımlanan B nesnesinin adresinin temel sınıf **göstericisine(pointer)** atanması ve daha sonra bu gösterici ile fonk() fonksiyonunun çağrılması çok biçimliliğin bir kullanımıdır.
- Peki, **ptr->fonk()** şeklinde bir komut, A sinifinin fonk() fonksiyonunu mu çağırır, yoksa B sınıfının fonk() fonksiyonunu mu çağırır? Burada A sinifinin **ptr** göstericisiyle B sınıfının fonk() fonksiyonu çağrılmak istenebilir.
- C++'ta işte tam bu noktada sanal fonksiyonlar devreye girer. Sanal fonksiyonun bildirimi temel sınıfta yapılır ve türetilmiş sınıflarda tekrar tanımlanır.
- Temel sinifta ve diğer sınıflarda ortak olarak bulunan bir fonksiyonun temel siniftaki tanımlanmasının önüne **virtual** anahtar sözcüğü getirilerek sanal fonksiyon oluşturulmuş olur. **Üstteki** örnekte kullanılan **fonk()** fonksiyonunun tanımını **virtual** olarak yapsak sorunumuz çözülmüş olur. Böylece çağırma işlemi göstericinin türüne değil içeriğine göre yapılır

```
4  class A{
5      public:
6          void fonk(){
7              ...
8          }
9  };
10
11 class B:public A{
12     void fonk(){
13         ...
14     }
15 };
16
17 int main() {
18     B nesne1;
19     A *ptr =&nesne1;
20     ptr->fonk();
21
22     return 0;
23 }
```

```
1  #include <iostream>
2  using namespace std;
3
4  class A{
5      public:
6          virtual void fonk(){
7              ...
8          }
9  };
10
11 class B:public A{
12     void fonk(){
13         ...
14     }
15 };
16
17 int main() {
18     B nesne1;
19     A *ptr =&nesne1;
20     ptr->fonk();
21
22     return 0;
23 }
24
```

ÖRNEK-2

```cpp
#include <iostream>
using namespace std;

class anasinif{
    int t;
    public:
        anasinif(){}

        virtual void goster(){
            cout<<t<<endl;
        }
};
class A:public anasinif{
    int a,b;
    public:
        A(){}

        A(int x,int y){
            a=x;
            b=y;
        }

        void goster(){
            cout<<a<<" "<<b<<endl;
        }
};

class B:public anasinif{
    int c,d;
    public:
        B(){}

        B(int x,int y){
            c=x;
            d=y;
        }

        void goster(){
            cout<<c<<" "<<d<<endl;
        }
};
```

```cpp
int main() {
    A nesne1(10,20);
    B nesne2(30,40);
    anasinif *p;

    p=&nesne1;
    p->goster();

    p=&nesne2;
    p->goster();

    return 0;
}
```

```
C:\Users\monster\Deskto
10 20
30 40
```

Örneğimizde **anasinif** adlı temel sınıfını oluşturduk ve bu temel sınıftan **A** ve **B** sınıflarını türettik.

anasinif temel sinifi içerisinde **sanal** olarak bir **goster()** fonksiyonu tanımladık ve bu fonksiyonu çok biçimliliğe uygun olarak türetilmiş sınıflarda yeniden tanımladık. Görüldüğü gibi **virtual** anahtar sözcüğü sadece temel sınıfta kullanılmıştır, türetilmiş sınıflarda tekrar kullanmaya gerek yoktur.

main içerisinde **A** ve **B** sınıflarından iki nesne oluşturduk. **anasinif** sınıfından oluşturduğumuz bir nesne göstericisine sırasıyla **A** sınıfından oluşturduğumuz nesnenin adresini ve **B** sınıfından oluşturduğumuz nesnenin adresini atayıp **p->goster()** şeklinde temel sınıf fonksiyonunu çağırdık.

Derleyici yürütme anında **p** göstericisinin içeriğine göre hangi türetilmiş sınıfın **goster()** fonksiyonunun çağrılacağına karar verdi.

Peki, hangi fonksiyonunun çağrılacağına nasıl karar veriyor?

Örneğimizde de görüldüğü gibi sanal fonksiyonlar çağrılırken nesne işaretçileri kullanıldı. Nesne işaretçileri yerine daha önce diğer üye fonksiyonları çağırırken kullandığımız nokta operatörü de kullanılabilirdi. Fakat temel sınıf göstericileri yardımıyla türetilmiş sınıflara ait fonksiyonları çağırmak, çok biçimliliği desteklediği için sanal fonksiyonlar çağrılırken nesne işaretçileri kullanılır. Tabii burada önemli olan çok biçimlilikte hangi fonksiyonun çağrılmak istendiğidir. Derleyici derleme anında hangi fonksiyonun çağrıldığını ayırt edemez. Fakat yürütme anında derleyici programa bir kod parçası ekleyerek göstericinin işaret ettiği türe göre hangi sınıfın üye fonksiyonunun çağrılacağını belirler.

ÖRNEK-3

```cpp
#include <iostream>
using namespace std;

class dortgen{
    public:
        int kenar1,kenar2;
        dortgen(){}

        virtual void alan(){
            cout<<"Dortgenin Alani="<<kenar1*kenar2<<endl;
        }
};

class kare:public dortgen{
    public:
        kare(){}

        kare(int x){
            kenar1=x;
        }

        void alan(){
            cout<<"Karenin Alani="<<kenar1*kenar1<<endl;
        }
};

class dikdortgen:public dortgen{
    public:
        dikdortgen(){}

        dikdortgen(int x,int y){
            kenar1=x;
            kenar2=y;
        }

        void alan(){
            cout<<"Dikdortgenin Alani="<<kenar1*kenar2<<endl;
        }
};
```

```cpp
int main() {
    kare sekil1(4);
    dikdortgen sekil2(4,5);
    dortgen *p;

    p=&sekil1;
    p->alan();

    p=&sekil2;
    p->alan();

    return 0;
}
```

```
C:\Users\monster\Desktop\ders sla
Karenin Alani=16
Dikdortgenin Alani=20
```

Örneğimizde **Dortgen** sınıfından **kare** ve **Dikdortgen** sınıflarını türettik.

Dortgen temel sınıfını tanımlarken alan fonksiyonunu **virtual** olarak bildirdik. Bu sanal fonksiyonu **Kare** ve **Dikdortgen** sınıflarında tekrar tanımladık.

main içerisinde **Kare** ve **Dikdortgen** sınıflarından birer nesne oluşturduk ve yapıcı fonksiyonlarına parametreleri verdik. **Kare** sınıfının yapıcı fonksiyonu kalıtım ile **Dortgen** sınıfından kendisine geçen **kenar1** değişkenine parametresini atadi. Aynı şekilde **Dikdortgen** sınıfının yapıcı fonksiyonu da parametrelerini **kenar1** ve **kenar2** değişkenlerine atadi.

main içerisinde tanımladığımız **Dortgen** sınıfından bir nesne göstericisi olan **p**'ye ilk olarak daha önce tanımladığımız **Kare** sınıfının nesnesi olan **sekil1** nesnesinin adresini atadık. Böylece temel sınıfın göstericisi türetilmiş sınıfın göstericisi olmuş oldu. **p->goster()** şeklinde temel sınıfın göster() fonksiyonunu çağırdık. Derleyici yürütme zamanında programa bir kod ekleyerek bu kod sayesinde **Kare** sınıfının **goster()** fonksiyonunu çağırdı ve ekrana karenin alanı yazıldı.

Ikinci atamada **p** ye **Dikdortgen** sınıfının nesnesi olan **sekil2** nesnesinin adresi atandı ve **p->goster()** şeklinde temel sınıfın göster() fonksiyonu çağrıldı. Yine yürütme zamanında derleyici programa bir kod ekleyerek bu kod yardımıyla **Dikdortgen** sınıfının **goster()** fonksiyonunu çağırdı ve ekrana dikdörtgenin alanı yazdırıldı.