NESNE MODELLERİ : SINIFLAR

Bu bölümünden itibaren C++ programlama dilinin nesneye dayalı programlamaya yönelik özellikleri tanıtılacaktır.

Bu özellikler yazılımların kalitesini yükseltmek amacıyla geliştirilmişlerdir.

Bilgisayar programı yazmanın, aslında gerçek dünyadaki unsurların bilgisayarda birer modellerinin oluşturulması anlamına gelir.

Gerçek dünya nesnelerden oluşmaktadır. Bu nedenle bilgisayar programları da bu nesnelerin modellerinden oluşturulmaktadır.

Nesne Nedir?

Gerçek dünyada bir nesne iki kısımdan oluşur:

1. Nitelikler. Buna durumlar ya da özellikler de denir.

2. Davranışlar (yetenekler).

Nesnelerin bilgisayarda kurulan modellerinde ise aşağıda gösterildiği gibi nitelikleri belirtmek için veriler, davranışları belirtmek için de fonksiyonlar kullanılacaktır.

Gerçek dünyada nesne = Nitelikler + Davranışlar

Yazılım dünyasında nesne = Veriler + Fonksiyonlar

Sınıflar ve Nesneler

C++'da nesnelerin modellerini oluşturmak için sınıf (*class*) adı verilen bir yapı kullanılmaktadır.

Bu yapı bir nesneyi oluşturacak olan verileri ve fonksiyonları birlikte barındırmaktadır.

Sınıflar, nesnelerin modeli diğer bir deyişle şablonudur. Programda bir kez sınıf yazılıp şablon oluşturulduktan sonra o sınıftan gerektiği kadar nesne yaratılabilir.

Örnek: Grafik programlarında kullanılacak nokta nesnelerini tanımlamak üzere bir model.
Noktalar iki boyutlu düzlemde yer alacağından özellik olarak iki adet koordinat bilgisine sahiptirler:

- x ve y koordinatları. Bu özellikler tamsayı değişkenler ile ifade edilebilirler.

Örnek programımızda noktaların sahip olması gereken yetenekler (davranışlar) ise şunlardır:

- Noktalar, düzlemde herhangi bir yere konumlanabilmeli: git fonksiyonu
- Noktalar bulundukları koordinatları ekrana çıkartabilmeli: goster fonksiyonu
- Noktalar, sıfır (0,0) koordinatında olup olmadıkları sorusunu yanıtlayabilmeli: sifir_mi fonksiyonu

Nokta sınıfı:

```
class Nokta{                    // Nokta Sınıfı
   int x,y;                     // Nitelikler: x ve y koordinatları
 public:
   void git(int, int);          // Noktanın hareket etmesini sağlayan fonksiyon
   void goster();               // Noktanın koordinatlarını ekrana çıkartır
   bool sifir_mi();             // Noktanın (0,0) koordinatlarında olup olmadığı
};
```

Bildirim class sözcüğü ile başlar, daha sonra sınıfın ismi gelir (Nokta).

Yukarıdaki örnekte önce veriler sonra fonksiyonlar yazılmıştır. Bu sıra ters de olabilir.

Sınıfın içindeki veri ve fonksiyonlara o sınıfın **üye**leri (*member*) denir.

Sınıf bildirimi noktalı virgül (;) ile bitirilir.

Örnek sınıf bildiriminin içinde fonksiyonların sadece prototipleri yer almaktadır.

Fonksiyonların gövdeleri ise ayrı bir yerde tanımlanabilir.

Üye fonksiyonlara o sınıfın **metot**ları da (*method*) denilmektedir.

```
// ***** Üye Fonksiyonların Gövdeleri *****

// Noktanın hareket etmesini sağlayan fonksiyon
void Nokta::git(int yeni_x, int yeni_y)
{
   x = yeni_x;                                  // x koordinatına yeni değer atandı
   y = yeni_y;                                  // y koordinatına yeni değer atandı
}

// Noktanın koordinatlarını ekrana çıkaran fonksiyon
void Nokta::goster()
{
   cout << "X= " << x << ", Y= " << y << endl;
}

// Noktanın (0,0) koordinatlarında olup olmadığını belirten fonksiyon
bool Nokta::sifir_mi()
{
   return (x == 0) && (y == 0);          // x=0 VE y=0 ise doğru
}
```

Üye fonksiyonların gövdeleri de yazıldıktan sonra Nokta modeli (şablon) tamamlanmıştır. Artık bu sınıftan gerektiği kadar nokta nesnesi yaratılabilir.

```
int main()
{
    Nokta n1,n2;                // 2 adet nesne tanımlandı n1 ve n2
    n1.git(100,50);             // n1 (100,50)'ye gönderiliyor
    n1.goster();                // n1'in koordinatları ekrana çıkartılıyor
    n1.git(20,65);              // n1 (20,65)'e gönderiliyor
    n1.goster();                // n1'in koordinatları ekrana çıkartılıyor
    if(n1.sifir_mi())           // n1 sıfır da mı?
      cout << "n1 şu anda sıfır noktasındadır." << endl;
    else
      cout << "n1 şu anda sıfır noktasında değildir." << endl;
    n2.git(0,0);                // n2 (0,0)'a gönderiliyor
    if(n2.sifir_mi())           // n2 sıfır da mı?
      cout << "n2 şu anda sıfır noktasındadır." << endl;
    else
      cout << "n2 şu anda sıfır noktasında değildir." << endl;
    return 0;
}
```

Bkz Örnek o31.cpp

Programda iki adet nesne (n1 ve n2) tanımlanmıştır. Bu nesnelerin üye fonksiyonları çağırılarak nesnelerin belli davranışlarda bulunmaları sağlanmıştır. Nesnelerin metotlarının canlandırılmasına o nesneye **mesaj** göndermek denir.

Değerlendirme:

Sınıflar ve nesneler hakkında bu aşamaya kadar öğrendiklerimizi değerlendirdiğimizde şu sonuçlara varabiliriz:

Sınıflar 'aktif' veri tipleridir. Bu tiplerden nesneler tanımlanır. Nesnelere mesajlar gönderildiğinde nesneler bir davranışta bulunarak cevap verirler.

Sınıflar hem verileri hem de bu veriler üstünde işlem yapan fonksiyonları aynı paket içinde barındırırlar (*encapsulation*). Bunun sonucu olarak:

- Nesneler gerçek dünyadakine benzer şekilde modellenebilir,
- Program daha kolay okunabilir, çünkü birbirleriyle ilgili olan veriler ve fonksiyonlar bir aradadır,
- Hataları bulmak kolay olur, çünkü bu veriler üzerinde işlem yapabilen fonksiyonlar bellidir.

Metotların inline Fonksiyon (Makro) Olarak Tanımlanmaları:

Örnek 3.1'deki Nokta sınıfının bildiriminde metotların (üye fonksiyonların) sadece prototipleri yer almaktadır. Fonksiyonların gövdeleri ise sınıfın dışında tanımlanmıştı. Metotların gövdeleri de sınıf bildiriminin içine yazılabilir. Bu durumda o fonksiyon makro (inline) olarak tanımlanmış olur.

Aşağıdaki örnekte sifir_mi adlı üye fonksiyon sınıfın içine yazılarak makro olarak tanımlanmıştır.

```
class Nokta{              // Nokta Sınıfı
   int x,y;               // Nitelikler: x ve y koordinatları
  public:
   void git(int, int);    // Noktanın hareket etmesini sağlayan fonksiyon
   void goster();         // Noktanın koordinatlarını ekrana çıkartır
   bool sifir_mi()        // Noktanın (0,0) koordinatlarında olup olmadığı (makro)
   {
       return (x == 0) && (y == 0);
   }
};
```

Nesnelerin Dinamik Olarak Yaratılması:

Derleyicinin hazır veri tipleri (int, float, char vs.) veri tanımlamak için nasıl kullanılıyorsa sınıflar da nesne tanımlamak için aynı şekilde kullanılabilir. Örneğin nesnelere işaret edebilen işaretçiler tanımlanabilir ve bu işaretçiler için dinamik bellekten yer alınabilir ve daha önce alınmış olan yerler geri verilebilir.

Aşağıdaki programda nesnelere işaret edebilen iki işaretçi (pn1 ve pn2) tanımlanmakta, dinamik bellekte bu işaretçiler tarafından işaret edilen nesneler yaratılmaktadır.

```
int main()
{
    Nokta*pn1 = new Nokta;     //pn1'in gösterdiği nesne için bellekten yer alındı
    Nokta*pn2 = new Nokta;     //pn2'nin gösterdiği nesne için bellekten yer alındı
    pn1->git(50,50);           //pn1'in gösterdiği nesneye git mesajı
    pn1->goster();             //pn1'in gösterdiği nesneye goster mesajı
    pn2->git(100,150);         //pn2'nin gösterdiği nesneye git mesajı
    if(pn2->sifir_mi())        // pn2'nin gösterdiği nesne  sıfır da mı?
        cout << "pn2'nin nesnesi şu anda sıfır noktasındadır." << endl;
    else
        cout << "pn2'nin nesnesi şu anda sıfır noktasında değildir." << endl;
    delete pn1;                 // Dinamik bellekten alınan yerler geri veriliyor
    delete pn2;
    return 0;
}
```

Nesnelerin Dizi Olarak Yaratılması:

Sınıflardan, nesne dizileri de tanımlamak mümkündür. Aşağıdaki örnekte Nokta sınıfından, adı dizi olan 10 elemanlı bir nesne dizisi yaratılmış ve kullanılmıştır.

```
int main()
{
    Nokta   dizi[10];                  // 10 elemanlı dizi yaratılıyor.
    dizi[0].git(15,40);                // dizinin ilk elemanına (0 indisli) git mesajı
    dizi[1].git(75,35);                // dizinin ikinci elemanına (1 indisli) git mesajı
     :                                 // Diğer elemanlar da konumlandırılır
    for (int i= 0; i < 10; i++)        // dizideki tüm nesneler ekrana çıkartılıyor
        dizi[i].goster();
    return 0;
}
```

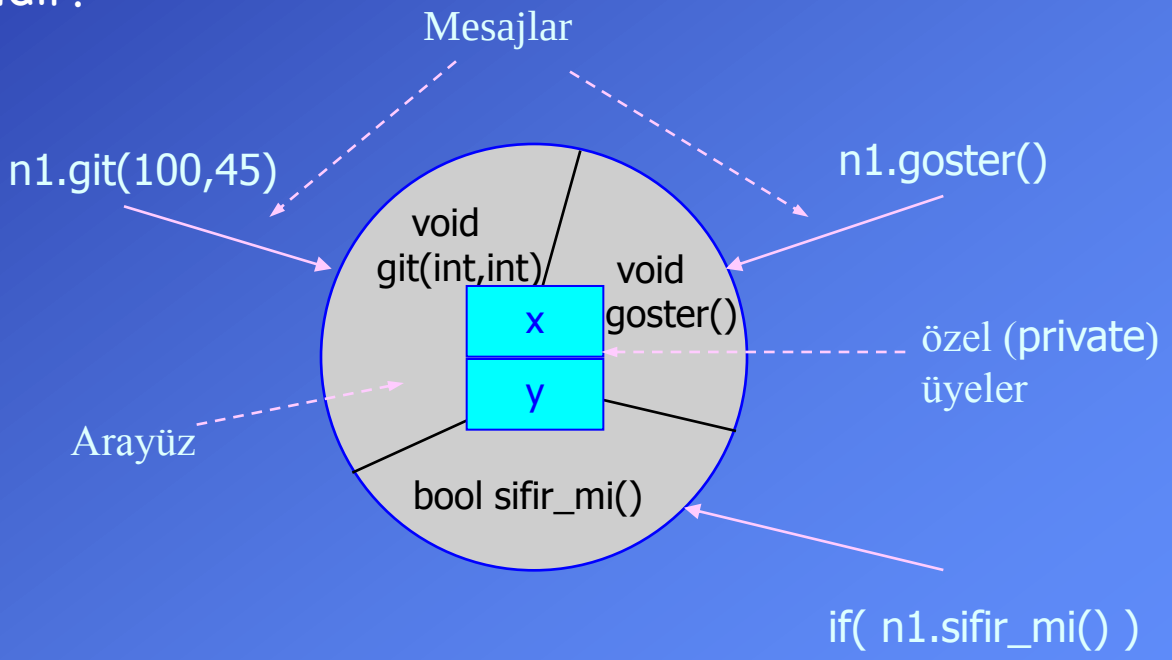
Sınıf Üyelerine Erişimin Denetlenmesi

Programcılar, sınıfların oluşturulması ve kullanılması açısından ikiye ayrılırlar. Birincisi sınıfları yazan programcılardır. İkincisi ise sınıfları kullanarak nesneler yaratan ve bu nesnelere mesajlar göndererek onları canlandıran programcılardır.
Sınıfı oluşturan programcı sınıf yapısının doğru çalışmasından sorumludur. Sınıfı nesne tanımlamak için kullanan programcı sınıfın iç yapısını bilmek zorunda değildir. Bilmesi gereken tek şey nesnelere nasıl mesaj yollanacağıdır.

Bu özellikleri sağlamak için, sınıfı yazan programcı, sınıfın bazı üyelerini gizleyerek (*data hiding*) onlara sınıf dışından erişilmesini engelleyebilir. Bir sınıfın içindeki üyelere erişimi denetleyen üç farklı etiket bulunmaktadır: **public** (açık), **private** (özel) ve **protected** (korumalı).

Açık (public) olarak tanımlanan üyelere, bu sınıftan yaratılan tüm nesneler yoluyla erişilebilir.
Özel (private) olarak tanımlanan üyelere (veri ve fonksiyon) ise sadece sınıfın içindeki fonksiyonlar erişebilir. Bu sınıftan yaratılan nesneler üzerinden özel üyelere erişmek mümkün değildir.
Korumalı (protected) üyeler ise kalıtım (*inheritance*) özelliği ile birlikte kullanılmaktadır. Kalıtım konusu anlatılırken korumalı erişim yöntemi de açıklanacaktır.

Bir sınıfın içindeki açık elemanlar o sınıfın dışarıya verdiği hizmetleri (*services*) tanımlarlar. Bunların tümüne sınıfın **arayüzü** (*interface*) denir. Özel üyeler sınıfın **gerçeklenmesi**yle (*implementation*) ilgili olduklarından sınıfın kullanıcılarını ilgilendirmezler.
Bir sınıftaki üyelerin erişim tipi normalde özeldir (private). Sınıfın yazarı dışarıya açmak istediği üyelerin tanımından önce public: etiketini koyar. Bu nedenle örnekteki Nokta sınıfındaki ilk iki üye (x , y) özeldir ve dışarıya kapalıdır. Ana programda yaratılan nesneler üzerinden bu verilere erişmek mümkün değildir.



Örnek olarak Nokta sınıfından üretilen noktaların hareket alanlarının x ekseninde 0-500, y ekseninde ise 0-300 aralığında olması ve noktalar bu alanın dışına çıkamamaları istensin.

```
class Nokta{                    // Nokta Sınıfı
   int x,y;                     // özel (private) Nitelikler: x ve y koordinatları
  public:                       // açık (public) üyeler
   bool git(int, int);          // Noktanın hareket etmesini sağlayan fonksiyon
   void goster();               // Noktanın koordinatlarını ekrana çıkartır
   bool sifir_mi();             // Noktanın (0,0) koordinatlarında olup olmadığı
};

// Noktanın hareket etmesini sağlayan fonksiyon
bool Nokta::git(int yeni_x, int yeni_y)
{
   if( yeni_x > 0 &&  yeni_x < 500 &&        // yeni_x 0-500 aralığında ise
       yeni_y > 0 &&  yeni_y < 300)          // yeni_y 0-300 aralığında ise
   {
          x = yeni_x;                        // x koordinatına yeni değer atandı
          y = yeni_y;                        // y koordinatına yeni değer atandı
          return true;                       // atama yapıldı, hata yok
   }
   return false;                             // Giriş değerleri kabul edilmedi
}
```

Yeni yazılan git fonksiyonu geriye bir mantıksal değer döndürmektedir. Bu değer sayesinde fonksiyona gönderilen koordinatların kabul edilip edilmediği anlaşılmaktadır.

```
int main()
{
    Nokta n1;          // n1 nesnesi yaratıldı
    int x,y;           // Tuş takımından değer okumak için iki değişken tanımlandı
    cout << "Noktanın gideceği koordinatları veriniz ";
    cin >> x >> y;          // Tuş takımından iki değer okunuyor
    if( n1.git(x,y) )       // git fonksiyonu çağırılıyor ve sonuç değerlendiriliyor
       n1.goster();          // Atama yapıldıysa ekrana yeni koordinatlar çıkacak
    else
      cout << "\nGirilen değerler kabul edilmemiştir";
}
```

Erişim Etiketlerinin Geçerlilik Alanları:

Bir erişim etiketi yazıldığı yerden aşağıya doğru geçerli olur. Bu etiketin altında yer alan tüm veriler ve fonksiyonlar etikette belirtilen erişim türüne sahip olurlar. Bir etiketin geçerlilik alanı, başka bir etiket yazıldığında ya da sınıfın sonuna gelindiğinde sona erer.

C++'da struct Yapısı:

C++'da struct yapısı da class ile aynı işlevi yerine getirmekte ve sınıf bildirimleri için kullanılmaktadır.
struct ile class arasındaki tek fark üyelerin erişim haklarında ortaya çıkmaktadır.
class bildirimlerinde aksi bildirilmedikçe üyelerin erişim tipi özeldir (private).
struct bildirimlerinde ise aksi bildirilmedikçe üyelerin erişim tipi açıktır (public).

Sınıflarda Arkadaşlık (friend) İlişkisi

Bir A sınıfı oluşturulurken B sınıfı A'nın arkadaşı olarak bildirilebilir. Bu durumda B'nin tüm üye fonksiyonları erişim tipleri ne olursa olsun A'nın tüm üyelerine (public, private, protected) erişebilirler.

```
class A{
  friend class B;                    // B sınıfı A'nın arkadaşıdır
  private:                      // A'nın özel üyeleri
     int i;
     float f;
  public:                            // A'nın açık üyeleri
     void fonk1(char *c);
};

class B{                             // B sınıfı
    int j;
  public:
     void fonk2(A &s){cout << s.i;}          // B, A'nın özel üyesi i'ye erişebiliyor
};
```

Sınıflar arası arkadaşlık ilişkisi tek yönlüdür. Yukarıdaki örnekte B, A'nın arkadaşıdır ve onun tüm üyelerine erişmektedir. Ancak A, B'nin arkadaşı değildir.

Bir fonksiyon da bir sınıfın arkadaşı olarak bildirilebilir. Fonksiyonlar arkadaşları oldukları sınıfların tüm üyelerine (public, private, protected) erişebilirler.

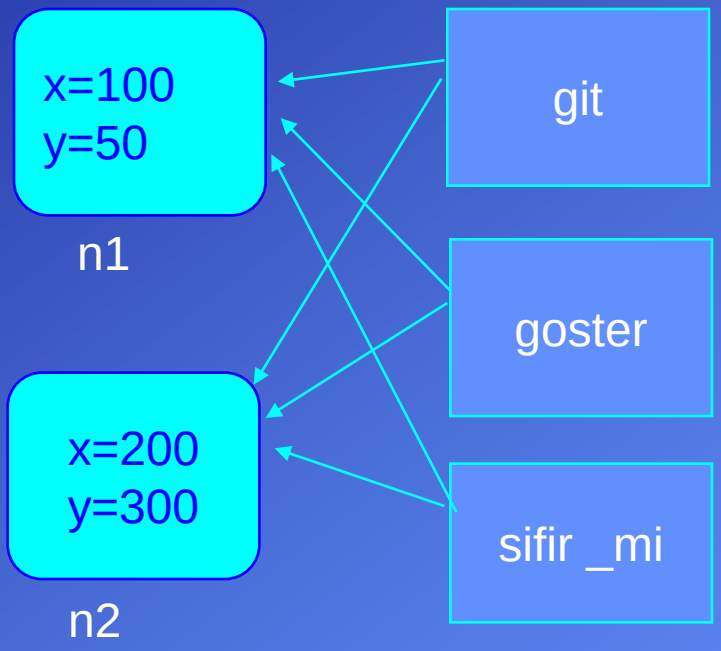
```
class Nokta{                          // Nokta Sınıfı
   friend void sifirla(Nokta &);      // Nokta sınıfının arkadaşı olan sifirla fonksiyonu
   int x,y;                           // özel (private) Nitelikler: x ve y koordinatları
 public:                              // açık (public) üyeler
   bool git(int, int);                // Noktanın hareket etmesini sağlayan fonksiyon
   void goster();                     // Noktanın koordinatlarını ekrana çıkartır
   bool sifir_mi();                   // Noktanın (0,0) koordinatlarında olup olmadığı
};

void sifirla (Nokta &n)               // Hiçbir sınıfın üyesi değil
{
   n.x = 0;                           // n'nin x elemanı sıfırlanıyor
   n.y = 0;                           // n'nin x elemanı sıfırlanıyor
}
```

this İşaretçisi

Bir sınıftan yaratılan her nesnenin sadece verileri için bellekte ayrı yerler ayrılır. Sınıfın fonksiyonları ise bir kere derlenir ve tüm nesneler aynı fonksiyonları paylaşırlar.

Örnek Nokta sınıfından n1 ve n2 nesneleri yaratıldığında bu nesneler bellekte verileri kadar yer kaplarlar. Nokta sınıfında iki adet tamsayı veri (x ve y) bulunduğundan bu sınıftan tanımlanan nesneler de sadece iki tamsayı içerirler.



Fonksiyonlar tüm nesneler tarafından ortak olarak kullanıldığında göre bir fonksiyon çağırıldığında hangi nesnedeki veriye erişeceği nasıl belirlenmektedir?

Aynı fonksiyonun farklı nesneler üzerinde çalışmasını sağlamak için C++ derleyicileri tarafından her programda **this** adında bir işaretçi tanımlanmaktadır. Bir sınıfın üye fonksiyonu hangi nesne için çağırılırsa this işaretçisine o nesnenin adresi atanır. Örneğin n1.goster(); çağrısı yapıldığında this, n1'in adresini taşır, n2.goster(); çağrısı yapıldığında ise n2'nin adresini taşır.

Programcılar da gerek duyduklarında this işaretçisini programlarında kullanabilirler. Aşağıda Nokta sınıfına uzak adında bir fonksiyon eklenmiştir. Bu fonksiyon parametre olarak Nokta tipinden başka bir nesne almakta ve hangi nesnenin (noktanın) sıfır noktasına (0,0) daha uzak olduğunu hesaplamaktadır. Daha uzak olan nesnenin adresi geriye gönderilmektedir.

```
// Hangi noktanın (0,0)'a daha uzak olduğunu bulan fonksiyon
Nokta *Nokta::uzak(Nokta &n)
{
   unsigned long x1 = x*x;
   unsigned long y1 = y*y;
   unsigned long x2 = n.x * n.x;
   unsigned long y2 = n.y * n.y;
   if ( (x1+y1) > (x2+y2) ) return this;      // Kendi nesnesinin adresi
   else return &n;                            // Parametre olarak gelen nesne
}
```

Bkz Örnek o32.cpp