Nesne Yönelimli Programlamaya Giriş

Bu hafta

- Nesne Yönelimli Programlamanın Temelleri
- Sınıflar ve Nesneler
- Sınıf Tanımı ve Nesne Oluşturma
- Erişim Belirleyicileri: `public`, `private`, `protected`
- Yapıcı (Constructor) ve Yıkıcı (Destructor) Fonksiyonlar

Nesne Yönelimli Programlama

Nesne Yönelimli Programlama (OOP), C++ dilinde programların yalnızca fonksiyonlar ve mantık üzerinden değil, gerçek dünyadaki nesneler etrafında yapılandırıldığı bir yaklaşım veya programlama modelidir. Bu model, **veriyi** ve bu veriyi işleyen **fonksiyonları** bir araya getirerek, yazılım bileşenlerini birer "**nesne**" olarak ele alır.

OOP, verileri temel olarak iki ayrı bellek alanına ayırır: birincisi veri **üyeleri** (properties), ikincisi ise bu verilere erişim veya bu veriler üzerinde işlem yapmayı sağlayan **fonksiyonlar** (methods). Bu ayrım, kodun hem esnek hem de modüler olmasını sağlar.

Nesneler arasındaki ilişkilendirmeler ve sınıfların kullanımı, büyük ve karmaşık yazılımların daha kolay bir şekilde organize edilmesine ve yönetilmesine olanak tanır. Aynı zamanda OOP, yeniden kullanılabilirlik, kodun okunabilirliği ve sürdürülebilirliği gibi yazılım geliştirme süreçlerinde kritik öneme sahip avantajlar sunar.

Nesne Yönelimli Programlama Nedir?

Nesne Yönelimli Programlama, nesneleri ve sınıfları kullanarak programlama yapılan bir yaklaşımdır. Bu yaklaşım, gerçek dünyadaki varlıkların dijital dünyaya aktarılmasını hedefler. OOP, kalıtım (inheritance), çok biçimlilik (polymorphism), veri gizleme (data hiding) gibi gerçek dünya kavramlarına dayanır. Amacı, veriyi ve bu veri üzerinde çalışan fonksiyonları bir araya getirerek tek bir yapı içinde toplamak ve bu yapıların dışarıdan kontrolsüz bir şekilde kullanılmasını engellemektir.

Kısaca Nesne Yönelimli Programlama; her şeyin bir nesne olarak temsil edildiği bir programlama paradigmasıdır. OOP'ler, gerçek dünya varlıklarını nesneler biçiminde uygular. OOP'nin temel amacı, verileri ve bunlar üzerinde çalışan işlevleri birlikte düzenlemektir, böylece programın başka hiçbir bölümü bu verilere o işlev dışında erişemez.

Nesne Yönelimli Programlama (OOP) Kavramları - C++

C++'da Nesne Yönelimli Programlama (OOP) sekiz temel prensip üzerine kuruludur:

1. **Class (Sınıf)**

- Kullanıcı tanımlı veri tipi.
- Veri (attributes) ve davranışları (methods) bir arada tutar.

2. **Object (Nesne)**

- Sınıfın bir örneği (instance).
- Sınıftaki özellik ve yöntemlerin somut temsili.

3. **Inheritance (Kalıtım)**

- Bir sınıfın başka bir sınıftan türetilmesi.
- Kodun yeniden kullanılabilirliğini artırır.

4. **Polymorphism (Çok Biçimlilik)**

- Aynı işlevin farklı şekillerde çalışabilmesi.
- Örnek: Overloading ve virtual functions.

5. **Abstraction (Soyutlama)**
   - Karmaşıklığı gizleme, yalnızca gerekli bilgileri gösterme.
   - Örnek: Arabirimin (interface) kullanımı.
6. **Encapsulation (Kapsülleme)**
   - Veriyi ve fonksiyonları bir arada tutma.
   - Veri gizliliğini sağlar (`private` erişim belirleyici).

7. **Dynamic Binding (Dinamik Bağlama)**

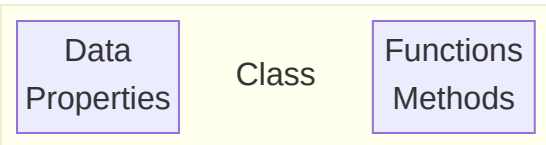
- Çalışma zamanında (runtime) yöntemin çağrılması.
- Örnek: Virtual functions ile polimorfizm.

8. **Message Passing (Mesaj Gönderimi)**

- Nesneler arası iletişim mekanizması.
- Örnek: Metot çağrıları yoluyla veri alışverişi.

1. Class (Sınıf)

C++'da sınıf, Nesne Yönelimli Programlamanın (OOP) temel kavramlarından biridir. Gerçek dünyadaki varlıkları temsil eden nesneleri oluşturmak için bir plan veya şablon görevi görür. Bir sınıf, veri üyelerini (nitelikler) ve veriler üzerinde çalışan üye işlevlerini (yöntemler) kapsülleyerek karmaşık sistemleri modellemek için bir yol sağlar.



Sınıfın Temel Özellikleri

Kapsülleme (Encapsulation)

- **Tanım**: Sınıf, verileri (özellikler/attributes) ve fonksiyonları (metotlar/methods) bir araya getirir.
- **Avantajları**:
  - Nesnenin bazı bileşenlerine doğrudan erişimi kısıtlar.
  - Veri bütünlüğünü korur.

Yeniden Kullanılabilirlik (Reusability)

- **Tanım**: Bir sınıf tanımlandıktan sonra, aynı sınıf birden fazla nesne oluşturmak için yeniden kullanılabilir.
- **Faydaları**:
  - Tekrarlayan kod yazımını azaltır.
  - Modülerliği artırır.

Soyutlama (Abstraction)

- **Tanım**: Sınıflar, yalnızca gerekli özellikleri açığa çıkararak uygulama ayrıntılarını gizler.
- **Avantajları**:
  - Karmaşıklığı azaltır.
  - Daha temiz ve anlaşılır bir tasarım sağlar.

Kalıtım (Inheritance)

- **Tanım**: Sınıflar, başka sınıflardan özellik ve metot miras alabilir.
- **Avantajları**:
  - Kod tekrarını önler.
  - Hiyerarşik ilişkiler kurar.

Çok Biçimlilik (Polymorphism)

- **Tanım**: Bir sınıf, türetilmiş sınıflarda yeniden tanımlanabilen metotlar tanımlayabilir.
- **Avantajları**:
  - Dinamik davranışlar sağlar.
  - Esnek ve genişletilebilir kod yazılmasına olanak tanır.

C++'da Sınıf Yapısı

Bir Sınıfın Tipik Bileşenleri

1. **Veri Üyeleri (Data Members)**:
   - Sınıfın durumunu (state) tutan değişkenlerdir.
2. **Üye Fonksiyonlar (Member Functions)**:
   - Sınıfın davranışlarını tanımlayan fonksiyonlardır.
3. **Erişim Belirleyicileri (Access Specifiers)**:
   - Verilerin ve metotların erişilebilirliğini kontrol eden anahtar kelimeler.
   - Türleri:
     - `public` : Genel erişim.
     - `private` : Yalnızca sınıf içi erişim.
     - `protected` : Türetilmiş sınıflar için erişim.

Sınıf Söz Dizimi (Syntax)

C++'da Sınıf Tanımı

Bir sınıfın genel yapısı şu şekildedir:

```
class ClassName {
private:
    // Özel veri üyeleri ve metotlar
    // (Yalnızca sınıf içinde erişilebilir)

public:
    // Genel veri üyeleri ve metotlar
    // (Sınıf dışından erişilebilir)

    // Yapıcı Fonksiyon (Constructor): Nesneleri başlatır
    ClassName() {
        // Başlatma kodu
    }

    // Yıkıcı Fonksiyon (Destructor): Kaynakları temizler
    ~ClassName() {
        // Temizlik kodu
    }
};
```

C++'da Sınıf Oluşturma ve Kullanımı

Örnek: Bir Sınıf Tanımlama ve Kullanma

Kod

Verilen örnek, bir `Car` sınıfı tanımlayıp bu sınıfın nesnelerini kullanarak işlemler yapmayı göstermektedir:

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

// Sınıf Tanımı
class Araba {
private:
    string marka; // Marka bilgisi
    int hiz;    // Hız bilgisi

public:
    // Yapıcı Fonksiyon (Constructor)
    Araba(string ArabaMarkasi, int arabaHizi) {
        marka = ArabaMarkasi;
        hiz = arabaHizi;
    }

    // Araba bilgilerini gösteren metot
    void goster() {
        cout << "Araba Markası: " << marka << ", Hız: " << hiz << " km/s" << endl;
    }

    // Hız güncelleme metodu
    void hizGuncelle(int yeniHiz) {
        hiz = yeniHiz;
        cout << "Hız " << hiz << " km/s olarak güncellendi." << endl;
    }
};
```

```cpp
int main() {
    // Car sınıfından nesneler oluşturma
    Araba araba1("Toyota", 120);
    Araba araba2("Honda", 140);

    // Araba sınıfı metotlarına erişim
    araba1.goster();
    araba2.goster();

    araba1.hizGuncelle(150);
    araba1.goster();

    return 0;
}
```

Örnekteki Temel Özellikler

Kapsülleme (Encapsulation)

- **Özellikler**: `marka` ve `hiz` değişkenleri özel (private) olarak tanımlanmıştır.
- **Erişim**: Bu değişkenlere yalnızca sınıfın genel (public) metotları aracılığıyla erişilebilir veya değiştirilebilir. Bu sayede veri güvenliği sağlanır.

Yeniden Kullanılabilirlik (Reusability)

- **Özellikler**: `Araba` sınıfı, birden fazla nesne oluşturmak için kullanılabilir.
- **Örnek**: `araba1` ve `araba2` nesneleri, aynı sınıf yapısını kullanarak farklı araba bilgilerini taşır. Bu, sınıfın yeniden kullanılabilirliğini sağlar.

Veri Koruma (Data Protection)

- **Özellikler**: `marka` ve `hiz` özelliklerine doğrudan erişim engellenmiştir.
- **Faydalar**:
  - Yanlışlıkla veri değiştirme veya zarar verme ihtimali ortadan kalkar.
  - Verinin tutarlılığı korunur.

Gerçek Hayat Analojisi

- **Sınıf**: Bir sınıf, bir araba için hazırlanan bir plan (blueprint) gibidir.
  - Plan, arabaların sahip olacağı özellikleri (örneğin, renk, hız, marka) ve davranışları (örneğin, hızlanma, fren yapma) tanımlar.
- **Nesneler**: Bu plan (sınıf) kullanılarak birden fazla araba (nesne) üretilebilir.
  - Her araba (nesne), aynı planı takip eder, ancak her birinin kendine özgü özellikleri olabilir (örneğin, birinin markası Toyota, diğerininki Honda olabilir).

Bir sınıfın amacı, bir nesnenin temel yapı taşlarını tanımlamak ve bu taşları kullanarak çok sayıda nesne oluşturulmasını sağlamaktır. Bu, aynı yapı ve işlevselliği paylaşan ama kendi özelliklerine sahip nesneler yaratma imkanı tanır.

2. Nesne (Object)

Nesne Nedir?

- Nesne, Nesne Yönelimli Programlama (OOP) içinde gerçek dünya varlıklarını temsil eden bir kavramdır.
- Her nesne, belirli bir **davranış** ve **durum** (state) ile tanımlanır.
- Nesneler, **fiziksel** veya **mantıksal** olabilir.

C++'da Nesne

- C++ programlama dilinde, bir nesne, bir sınıfın (class) örneğidir (instance).
- Bir sınıf tanımlandıktan sonra, bu sınıftan nesneler oluşturulabilir.
- **Bellek**, yalnızca bir nesne oluşturulduğunda ayrılır.

Nesnelerin Bellekteki Yeri

- Nesneler, **bellekte** bir alan kaplar ve bu alan, nesnenin **durumunu** (state) tutar.
- Nesneler, tanımlanan işlemlerle (operasyonlarla) üzerinde işlem yapılabilen birimlerdir.
- Her nesne, **veri** (data) ve bu veriyi manipüle eden **kod** (fonksiyonlar/metodlar) içerir.

Nesne Yapısı

1. **Veri (Data)**: Nesnenin durumunu belirler. Örneğin, bir banka hesabı nesnesi, bakiye gibi bilgileri tutar.
2. **Kod (Code)**: Nesnenin üzerinde işlem yapılmasını sağlayan fonksiyonlar/metotlar içerir. Örneğin, bir vektör nesnesi, eleman ekleme veya çıkarma gibi işlemleri yapabilir.

Örnek: Banka Hesabı Nesnesi

Verilen örnekte, bir banka hesabını temsil eden `BankaHesabi` sınıfı ve onun nesneleri kullanılmıştır.

- Banka Hesabı /1
- Banka Hesabı /2

Ana fonksiyonda kullanımı ;

```
int main() {
    // Banka hesabı nesneleri oluşturma
    BankaHesabi hesap1("Ahmet", 1000);
    BankaHesabi hesap2("Ayşe", 1500);

    // Nesnelerin metodlarını kullanma
    hesap1.bakiyeGoster();
    hesap2.bakiyeGoster();

    hesap1.paraYatir(500);
    hesap2.paraCek(200);

    return 0;
}
```