Nesne Yönelimli Programlamaya Giriş

Bu hafta

- Veri Kapsülleme Kavramı
- Getter ve Setter Fonksiyonları
- Veri Gizliliği
- Soyutlama Nedir? Nesnelerle İşlemler
- Uygulama: Basit Bir Sınıf Tasarımı

Veri Kapsülleme ve Soyutlama

Veri Kapsülleme (Encapsulation)

- **Veri kapsülleme**, nesne yönelimli programlamanın temel prensiplerinden biridir.
- Bir sınıfın verilerini (**data members**) ve bu verilere erişimi sağlayan işlemleri (**member functions**) bir arada tutar.
- Kapsülleme, **veri gizliliği** ve **kontrollü erişim** sağlar.

Getter ve Setter Fonksiyonları

- **Getter fonksiyonları**: Özel (`private`) verilere dışarıdan **okuma erişimi** sağlar.
- **Setter fonksiyonları**: Özel verilere dışarıdan **yazma erişimi** sağlar.
- Bu yöntem, verilerin **kontrollü bir şekilde değiştirilmesine** olanak tanır.

Örnek:

```cpp
#include <iostream>
using namespace std;
class Kisi {
private:
    string isim;
    int yas;
public:
    // Setter fonksiyonları
    void setIsim(string yeniIsim) {
        isim = yeniIsim;
    }
    void setYas(int yeniYas) {
        if (yeniYas > 0) { // Geçerli bir yaş kontrolü
            yas = yeniYas;
        } else {
            cout << "Geçersiz yaş!" << endl; }
    }
    // Getter fonksiyonları
    string getIsim() {
        return isim;
    }

    int getYas() {
        return yas;
```

```cpp
int main() {
    Kisi kisi;

    kisi.setIsim("Ahmet");
    kisi.setYas(25);

    cout << "İsim: " << kisi.getIsim() << endl;
    cout << "Yaş: " << kisi.getYas() << endl;

    return 0;
}
```

Veri Gizliliği (Data Privacy)

- Kapsülleme, sınıfın içindeki verilerin **dışarıdan doğrudan erişimini** sınırlar.
- Veriler genellikle `private` olarak tanımlanır.
- Dışarıdan erişim, yalnızca kontrollü bir şekilde **getter** ve **setter** fonksiyonlarıyla sağlanır.
- Avantajları:
  - **Veri bütünlüğü** korunur.
  - Verilere yapılan tüm işlemler kontrol edilebilir.

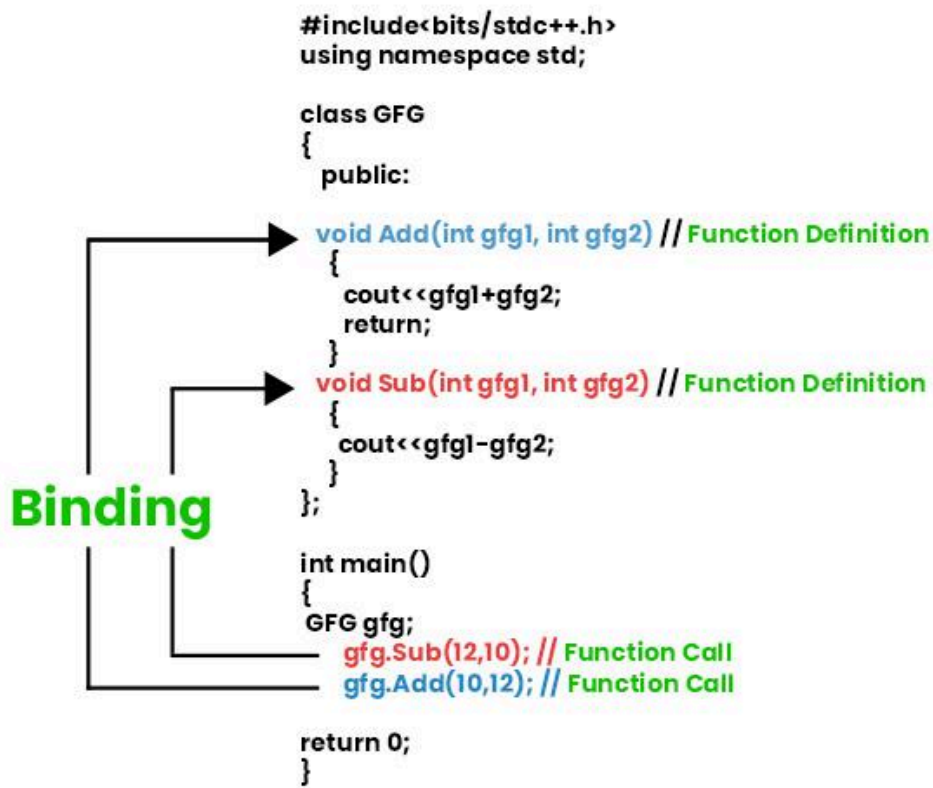
Soyutlama (Abstraction)

- **Soyutlama**, karmaşık sistemlerin yalnızca önemli ayrıntılarını gösterme, gereksiz ayrıntıları gizleme işlemidir.
- Nesneler, yalnızca ilgili işlemleri sağlayan bir arabirim sunar.
- Örneğin:
  - Bir arabanın motoru hakkında bilgi sahibi olmadan, yalnızca gaz pedalı ve direksiyon kullanılarak sürülebilir.
  - Programlamada, bir nesne belirli bir işlevsellik sunarken iç detaylarını gizler.

C++'da Dinamik Bağlama

Dinamik Bağlama Nedir?

- **Dinamik bağlama**, bir fonksiyon çağrısını ilgili fonksiyon tanımıyla çalışma zamanında bağlar.
- **Statik bağlamanın** oluşturduğu sorunları önler:
  - Statik bağlama, fonksiyon çağrısını ve tanımını derleme zamanında bağlar.
- Dinamik bağlama daha esnektir ve bu esneklik, statik bağlamanın sınırlamalarını ortadan kaldırır.


#include<bits/stdc++.h>
using namespace std;
class GFG
{
public:
void Add(int gfg1, int gfg2) // Function Definition
{
cout<<gfg1+gfg2;
return;
}
void Sub(int gfg1, int gfg2) // Function Definition
{
cout<<gfg1-gfg2;
}
};
int main()
{
GFG gfg;
gfg.Sub(12,10); // Function Call
gfg.Add(10,12); // Function Call
return 0;
}
Binding

Basit Tanım

- **Dinamik bağlama**, fonksiyon bildirimi ve çağrısı arasındaki bağlantıyı ifade eder.
- Çalışma zamanına kadar hangi fonksiyonun çalıştırılacağına karar verilmesi anlamına gelir.
- Çağrılan fonksiyon, nesnenin türüne bağlı olarak seçilir.

Dinamik Bağlamanın Kullanımı

- Tek bir fonksiyon adı kullanılarak birden fazla nesneyi işlemek mümkündür.
- Kodun **hata ayıklaması** ve hataların giderilmesi kolaylaşır.
- **Karmaşıklık azaltılır**.

Statik Bağlama ve Dinamik Bağlama Karşılaştırması

Statik Bağlama	Dinamik Bağlama
Derleme zamanında gerçekleşir (**erken bağlama**)	Çalışma zamanında gerçekleşir (**geç bağlama**)
Statik bağlama, gerekli tüm bilgilere sahip olduğu için daha hızlıdır.	Dinamik bağlama daha yavaştır çünkü fonksiyon çağrısı çalışma zamanına kadar çözülmez.
Normal fonksiyon çağrıları, operatör aşırı yükleme ve fonksiyon aşırı yükleme kullanılarak gerçekleşir.	Sanal fonksiyonlar kullanılarak gerçekleşir.
Gerçek nesneler statik bağlama kullanmaz.	Gerçek nesneler dinamik bağlama kullanır.

Polimorfizm (Polymorphism)

“Polimorfizm” kelimesi birçok forma sahip olmak anlamına gelir. Basit bir ifadeyle polimorfizmi bir mesajın birden fazla biçimde gösterilme yeteneği olarak tanımlayabiliriz. Polimorfizmin gerçek hayattaki bir örneği, aynı zamanda farklı özelliklere sahip olabilen bir kişidir. Erkek aynı zamanda hem baba hem koca hem de çalışandır. Yani aynı kişi farklı durumlarda farklı davranışlar sergiliyor. Buna polimorfizm denir. Polimorfizm, Nesneye Yönelik Programlamanın önemli özelliklerinden biri olarak kabul edilir.

Polimorfizm Türleri

C++ dilinde iki ana polimorfizm türü vardır:

1. **Derleme Zamanı Polimorfizmi (Compile-Time Polymorphism)**

- **Statik bağlama** ile gerçekleştirilir.
- Fonksiyon aşırı yükleme (**function overloading**) ve operatör aşırı yükleme (**operator overloading**) bu tür polimorfizmi sağlar.

Örnek: Fonksiyon Aşırı Yükleme

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

class HesapMakinesi {
public:
    int topla(int a, int b) {
        return a + b;
    }

    double topla(double a, double b) {
        return a + b;
    }
};

int main() {
    HesapMakinesi hesap;

    cout << "Tam Sayı Toplamı: " << hesap.topla(5, 10) << endl;
    cout << "Ondalık Sayı Toplamı: " << hesap.topla(3.5, 2.5) << endl;

    return 0;
}
```

Sanal Fonksiyonlar

- **Sanal fonksiyon**, bir temel sınıfta bildirilen ve bir türetilmiş sınıfta yeniden tanımlanan (override edilen) üye fonksiyondur.
- Türetilmiş sınıfın sanal fonksiyonunu, temel sınıfa bir işaretçi veya referans kullanarak çağırabilirsiniz.
- Dinamik bağlama, sanal fonksiyonlarla uygulanır.

Örnek:

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

class Temel {
public:
    virtual void yazdir() {
        cout << "Temel sınıf fonksiyonu." << endl;
    }
};

class Turetilmis : public Temel {
public:
    void yazdir() override {
        cout << "Türetilmiş sınıf fonksiyonu." << endl;
    }
};
```

```
int main() {
    Temel* ptr;
    Turetilmis obj;
    ptr = &obj;

    // Dinamik bağlama ile türetilmiş sınıf fonksiyonu çağrılır.
    ptr->yazdir();

    return 0;
}
```

Sanal Fonksiyonlar (Virtual Functions)

Sanal Fonksiyon Nedir?

- **Sanal fonksiyon**, temel bir sınıfta tanımlanmış, türetilmiş bir sınıfta yeniden tanımlanabilen (override edilen) bir üye fonksiyondur.
- Sanal fonksiyonlar, çalışma zamanında (runtime) hangi fonksiyonun çağrılacağını belirler.
- Anahtar kelime: `virtual`.

Sanal Fonksiyonların Temel Özellikleri

1. **Çalışma Zamanında Bağlama (Dynamic Binding)**:
   - Sanal fonksiyonlar, çalışma zamanında çağrıyı doğru fonksiyona bağlar.
   - Bu, dinamik bağlama (late binding) olarak adlandırılır.
2. **Türetilmiş Sınıf Üzerinde Kontrol**:
   - Temel sınıfın bir işaretçisi veya referansı kullanılarak türetilmiş sınıfın kendi fonksiyonları çağrılabilir.
3. **Temel Sınıfta Deklarasyon**:
   - Sanal fonksiyonlar, temel sınıfta `virtual` anahtar kelimesiyle bildirilir.
4. **Türetilmiş Sınıfta Yeniden Tanımlama (Override)**:
   - Türetilmiş sınıflar, temel sınıfta tanımlanan bir sanal fonksiyonu kendi ihtiyaçlarına göre yeniden tanımlayabilir.
5. **Sanallık Otomatik Devam Eder**:
   - Bir sanal fonksiyon türetilmiş sınıfta yeniden tanımlandığında, bu tanım otomatik olarak sanal olur.

Sanal Fonksiyonlar ile Örnek

Kod:

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

class Hayvan {
public:
    virtual void sesCikar() {
        cout << "Hayvan bir ses çıkarıyor." << endl;
    }
};

class Kopek : public Hayvan {
public:
    void sesCikar() override {
        cout << "Köpek havlıyor." << endl;
    }
};

class Kedi : public Hayvan {
public:
    void sesCikar() override {
        cout << "Kedi miyavlıyor." << endl;
    }
};
```

```
int main() {
    Hayvan* hayvanPtr;

    Kopek kopekObj;
    Kedi kediObj;

    hayvanPtr = &kopekObj;
    hayvanPtr->sesCikar(); // Dinamik bağlama ile Köpek sınıfının sesCikar() fonksiyonu çağrılır.

    hayvanPtr = &kediObj;
    hayvanPtr->sesCikar(); // Dinamik bağlama ile Kedi sınıfının sesCikar() fonksiyonu çağrılır.

    return 0;
}
```