Nesne Yönelimli Programlamaya Giriş

C++'da Erişim Belirleyicileri (Access Specifiers)

C++ dilinde bir sınıfın üyelerine (veri üyeleri ve üye fonksiyonlar) erişimi kontrol etmek için **erişim belirleyicileri** kullanılır. Bu belirleyiciler, sınıf üyelerinin erişim düzeylerini tanımlar.

1. `public` (Genel)

- **Açıklama**:
  - Genel üyeler, sınıfın dışından erişilebilir.
  - Hem sınıfın kendisi hem de sınıfın dışındaki herhangi bir kod bu üyelere erişebilir.
- **Kullanım Alanları**:
  - Sınıfın **dış dünya** ile etkileşim kurduğu üyeleri tanımlamak için kullanılır.

Örnek:

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

class Araba {
public:
    string marka;
    void yazdir() {
        cout << "Araba Markası: " << marka << endl;
    }
};

int main() {
    Araba araba;
    araba.marka = "Toyota"; // Public üyeye erişim
    araba.yazdir();

    return 0;
}
```

2. `private` (Özel)

- **Açıklama**:
  - Özel üyeler, yalnızca sınıfın içinden erişilebilir.
  - Sınıf dışındaki kod doğrudan bu üyelere erişemez.
- **Kullanım Alanları**:
  - Verilerin gizliliğini sağlamak ve doğrudan erişimi engellemek için kullanılır.

Örnek:

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

class Araba {
private:
    string marka;

public:
    void setMarka(string yeniMarka) {
        marka = yeniMarka;
    }

    void yazdir() {
        cout << "Araba Markası: " << marka << endl;
    }
};

int main() {
    Araba araba;
    // araba.marka = "Toyota"; // Hata: Private üyeye doğrudan erişim engellenir.
    araba.setMarka("Toyota"); // Private üyeye dolaylı erişim
    araba.yazdir();

    return 0;
}
```

3. `protected` (Korunan)

- **Açıklama**:
  - Korunan üyeler, sınıfın kendisi ve türetilmiş sınıflar tarafından erişilebilir.
  - Sınıf dışındaki diğer kod tarafından erişilemez.
- **Kullanım Alanları**:
  - Sınıf hiyerarşisinde verilerin miras alınarak kullanılmasını sağlamak.

Örnek:

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

class Temel {
protected:
    string isim;
};

class Turetilmis : public Temel {
public:
    void setIsim(string yeniIsim) {
        isim = yeniIsim; // Protected üyeye türetilmiş sınıfta erişim
    }

    void yazdir() {
        cout << "İsim: " << isim << endl;
    }
};

int main() {
    Turetilmis obj;
    obj.setIsim("Ali");
    obj.yazdir();

```

Erişim Belirleyicileri Karşılaştırması

Erişim Belirleyicisi	Sınıf İçinde	Türetilmiş Sınıflar	Sınıf Dışında
`public`	Erişilebilir	Erişilebilir	Erişilebilir
`private`	Erişilebilir	Erişilemez	Erişilemez
`protected`	Erişilebilir	Erişilebilir	Erişilemez

Friend Sınıflar (Friend Classes) ve Fonksiyonlar

Friend Nedir?

- C++'da bir sınıf, başka bir sınıfı veya fonksiyonu **friend** olarak tanımlayabilir.
- Friend olarak tanımlanan sınıf veya fonksiyon, **private** ve **protected** üyelerine doğrudan erişebilir.
- **Friend**, sınıfın bir üyesi değildir ancak sınıfın iç detaylarına erişim yetkisi verilmiştir.

Friend Kavramının Kullanımı

1. **Friend Fonksiyonlar**:
   - Bir sınıfın özel üyelerine erişebilen, sınıfın dışındaki fonksiyonlardır.
   - Sınıfın içinde `friend` anahtar kelimesiyle bildirilir.

2. **Friend Sınıflar**:
   - Bir sınıfın tüm üyelerine erişim yetkisi verilen başka bir sınıftır.

Friend Fonksiyon Örneği

Kod:

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

class Kutu {
private:
    double uzunluk;
public:
    Kutu() : uzunluk(0) {}

    // Friend fonksiyon bildirimi
    friend void uzunlukAyarla(Kutu& k, double l);
};
// Friend fonksiyon tanımı
void uzunlukAyarla(Kutu& k, double l) {
    k.uzunluk = l; // Private üyeye erişim
    cout << "Uzunluk: " << k.uzunluk << endl;
}

int main() {
    Kutu kutu1;
    uzunlukAyarla(kutu1, 10.5);

```

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

class Kisi {
private:
    string isim;
    int yas;

public:
    Kisi(string isim, int yas) : isim(isim), yas(yas) {}

    // Friend sınıf bildirimi
    friend class Doktor;
};

class Doktor {
public:
    void bilgileriGoster(const Kisi& kisi) {
        // Kisi sınıfının private üyelerine erişim
        cout << "İsim: " << kisi.isim << ", Yaş: " << kisi.yas << endl;
    }
};

int main() {
    Kisi kisi("Ahmet", 30);
    Doktor doktor;
    doktor.bilgileriGoster(kisi);
```

Friend Kullanımının Avantajları

1. **Kontrollü Erişim**:
   - Sınıf üyelerine sadece belirli fonksiyonlar veya sınıfların erişmesine izin verir.
2. **Kod Modülerliği**:
   - Sınıfın veri üyelerine doğrudan erişim sağlarken kodun bölünmesini destekler.
3. **Erişim Kısıtlamalarını Esnetme**:
   - Bazı durumlarda sınıf dışındaki fonksiyonların sınıfın iç üyelerine erişmesi gerekir; friend bunu mümkün kılar.

Friend Kullanımının Dezavantajları

1. **Kapsülleme İlkesi İhlali**:
   - Private üyelerin friend sınıflar ve fonksiyonlarla erişime açılması, kapsülleme ilkesini zayıflatabilir.
2. **Bağımlılık**:
   - Friend sınıflar ve fonksiyonlar, bağımlılığı artırarak kodun bakımı zorlaştırabilir.

- Friend sınıflar ve fonksiyonlar, özel üyeleri dış sınıflara veya fonksiyonlara açmak için kullanılır.
- **Friend Fonksiyonlar**: Sınıf üyelerine erişebilen sınıf dışı fonksiyonlardır.
- **Friend Sınıflar**: Başka bir sınıfın tüm üyelerine erişim yetkisi verilen sınıflardır.
- Kullanımda dikkatli olunmalı, yalnızca ihtiyaç duyulan yerlerde tercih edilmelidir.

Yerel Sınıflar (Local Classes)

Yerel Sınıf Nedir?

- **Yerel sınıf**, bir fonksiyon içinde tanımlanan sınıflardır.
- Tanımlandıkları fonksiyonun kapsamı dışında kullanılamazlar.
- C++'da yerel sınıflar, genellikle küçük ve belirli bir işleve hizmet eden sınıflar için kullanılır.

Yerel Sınıfların Özellikleri

1. **Kapsam**:
   - Yerel sınıf, yalnızca tanımlandığı fonksiyonun içinde erişilebilir.
2. **Kısıtlamalar**:
   - Yerel sınıflar, tanımlandıkları fonksiyonun yerel değişkenlerine **doğrudan erişemez**.
   - Ancak, sabit (`const`) değişkenlere erişebilir.
3. **Kullanım Alanları**:
   - Karmaşık veri yapılarının geçici olarak işlenmesi.
   - Fonksiyon içi işlemlerde sınıf bazlı kapsülleme sağlamak.

Yerel Sınıfların Kullanımı

Örnek:

```cpp
#include <iostream>
using namespace std;
void hesaplama() {
    // Yerel sınıf tanımı
    class Hesaplayici {
    public:
        int carp(int a, int b) {
            return a * b;
        }
        int topla(int a, int b) {
            return a + b; }
    };
    // Yerel sınıf nesnesi
    Hesaplayici hesap;
    int x = 5, y = 10;

    cout << "Toplam: " << hesap.topla(x, y) << endl;
    cout << "Çarpım: " << hesap.carp(x, y) << endl;
}

int main() {
    hesaplama(); // Yerel sınıfın kullanıldığı fonksiyonu çağırma
    return 0;
}
```