C++ OOP Temelleri

OOP nedir?

OOP, Nesne Yönelimli Programlama anlamına gelir. Nesne yönelimli bir programlama dili, programlamasında nesneleri kullanır. Nesne yönelimli kavramlarla programlama, bir programda kalıtım, polimorfizm, soyutlama vb. gibi gerçek dünya kavramlarını taklit etmeyi amaçlar.

C++ dili, C diline nesne yönelimli programlama eklemek amacıyla tasarlanmıştır. Programın boyutu arttıkça, programın okunabilirliği, sürdürülebilirliği ve hatasız yapısı azalır. OOP'nin temel amacı, verileri ve bunlar üzerinde çalışan işlevleri birbirine bağlamaktır, böylece kodun başka hiçbir bölümü bu işlev dışında bu verilere erişemez.

Bu, işlevlere veya prosedürlere dayanan C gibi dillerdeki en büyük sorundu (bu nedenle prosedürel programlama dili adı). Sonuç olarak, sorunu yeterince ele almama olasılığı yüksekti. Ayrıca, veriler neredeyse ihmal ediliyordu ve veri güvenliği kolayca tehlikeye atılıyordu. Sınıfları kullanmak, programı gerçek dünya senaryosu olarak modelleyerek bu sorunu çözer.

Prosedür Yönelimli Programlama ile Nesne Yönelimli Programlama Arasındaki Fark

Prosedür Yönelimli Programlama

- Bilgisayarın izlemesi için bir dizi talimat yazmaktan oluşur
- Ana odak, veri akışı değil, işlevlerdir.
- İşlevler yerel veya genel verileri kullanabilir
- Veriler, işlevden işleve açık bir şekilde taşınır
- Nesne yönelimli programlama

Sınıflar ve nesne kavramı üzerinde çalışır

- Sınıf, nesneler oluşturmak için bir şablondur
- Verileri kritik bir öğe olarak ele alır
- Nesnelerdeki sorunu ayrıştırır ve nesnelerin çevresinde veri ve işlevler oluşturur
- Temel olarak, prosedürel programlama, verileri manipüle eden prosedürleri veya işlevleri yazmayı içerirken, nesne yönelimli programlama, hem verileri hem de işlevleri içeren nesneler oluşturmayı içerir.

Nesne Yönelimli Programlamada Temel Öğeler

- Sınıflar - Nesne oluşturmak için temel şablon. Bu, nesne yönelimli programlamanın yapı taşıdır.
- Nesneler – Temel çalışma zamanı varlıkları ve bir sınıfın örnekleri.
- Veri Soyutlama ve Kapsülleme – Verileri ve işlevleri tek bir birime sarma

- Kalıtım - Bir sınıfın özellikleri başkalarına miras alınabilir
- Polimorfizm - Birden fazla form alabilme yeteneği
- Dinamik Bağlama – Yürütülecek kod, program çalışana kadar bilinmez
- Mesaj Geçirme – mesaj (Bilgi) çağrı formatı

Nesne Yönelimli Programlamanın Faydaları

Nesne yönelimli programlamanın birçok avantajı vardır. Aşağıda listelenen birkaçıdır. - OOP içeren programların yürütülmesi daha hızlı ve kolaydır. - Nesneleri ve kalıtımı kullanarak programlar için net bir yapı sağlar ve kodun yeniden kullanılabilirliğini geliştirir. - Kodun bakımını, değiştirilmesini ve hata ayıklamasını kolaylaştırır. - Veri gizleme ilkesi, güvenli sistemler oluşturmaya yardımcı olur - Birden Fazla Nesne, herhangi bir müdahale olmaksızın bir arada var olabilir - Yazılım karmaşıklığı kolayca yönetilebilir, böylece daha az kod ve daha kısa geliştirme süresi ile tamamen yeniden kullanılabilir yazılımların oluşturulması bile mümkündür.

Class'lar (Sınıflar)

Sınıflar ve yapılar biraz aynıdır ancak yine de bazı farklılıkları vardır. Örneğin, yapılarda veri saklayamayız, bu da her şeyin halka açık olduğu ve kolayca erişilebildiği anlamına gelir ki bu da yapının önemli bir dezavantajıdır çünkü veri güvenliğinin önemli olduğu yerlerde yapılar kullanılamaz. Yapıların bir diğer dezavantajı ise onlara fonksiyon ekleyememememizdir.

Sınıflar, kullanıcı tanımlı veri türleridir ve nesneleri oluşturmak için bir şablondur. Sınıflar, sınıf üyeleri olarak da adlandırılan değişkenler ve işlevlerden oluşur.

C++'da bir sınıf tanımlamak için class anahtar sözcüğünü kullanırız.

C++'da bir sınıfın sözdizimi şöyledir:

```
class class_name
{
    //body of the class
};
```

Objects (Nesneler)

Nesneler bir sınıfın örnekleridir. Bir nesne yaratmak için, sadece sınıf adını ve ardından nesnenin adını belirtmemiz gerekir. Nesneler, sınıf tanımına bağlı olan sınıf niteliklerine ve yöntemlerine erişebilir. Nesneler tarafından kullanılmalarına izin vermek için izinlerinin daha iyi belirlenebilmesi için bu özniteliklerin ve yöntemlerin erişim değiştiricilerine yerleştirilmesi önerilir.

C++'da bir nesneyi tanımlamanın sözdizimi şöyledir:

```
class class_name
{
    //body of the class
};

int main()
{
    class_name object_name; //object
}
```

Sınıf öznitelikleri ve yöntemleri, sınıf içinde tanımlanan değişkenler ve işlevlerdir. Ayrıca tamamen sınıf üyeleri olarak bilinirler.

Sınıf niteliklerinin ne olduğunu anlamak için aşağıdaki örneği inceleyin

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

class Employee
{
    int eID;
    string eName;
    public:
};


int main()
{
    Employee Harry;
}
```

Çalışan adlı bir sınıf oluşturulur ve sınıf içinde eId ve eName olmak üzere iki üye tanımlanır. Bu iki üye değişkendir ve sınıf öznitelikleri olarak bilinirler. Şimdi main'de Harry adında bir nesne tanımlandı. Harry, nokta operatörünü kullanarak bu niteliklere erişebilir. Ama halka açıklanmadıkça Harry'nin erişimine açık değiller.

```
class Employee
{
public:
    int eID;
    string eName;
};


int main()
{
    Employee Harry;
    Harry.eID = 5;
    Harry.eName = "Harry";
    cout << "Employee having ID " << Harry.eID << " is " << Harry.eName << endl;
}
```

Çıktı

```
Employee having ID 5 is Harry
```

Sınıf yöntemleri, bir sınıfta tanımlanan veya bir sınıfa ait olan işlevlerden başka bir şey değildir. Bir sınıfa ait yöntemlere, özniteliklere eriştikleri gibi nesneleri

tarafından erişilir. Fonksiyonlar bir sınıfa ait olacak şekilde iki şekilde tanımlanabilirler.

Sınıf içinde tanımlama

Sınıflar içindeki işlevleri tanımlamayı gösteren bir örnek,

```
class Employee
{
public:
    int eID;
    string eName;

    void printName()
    {
        cout << eName << endl;
    }
};
```

Sınıf dışında tanımlama

Bir fonksiyon sınıfın dışında tanımlanabilse de, içinde bildirilmesi gerekir. Daha sonra, işlevi dışarıda tanımlamak için kapsam çözümleme operatörünü (::) kullanabiliriz.

Sınıfların dışındaki işlevleri tanımlamayı gösteren bir örnek,

```
class Employee
{
public:
    int eID;
    string eName;

    void printName();
};


void Employee::printName()
{
    cout << eName << endl;
}
```

C++'da Nesnelere Bellek Ayırma

Her ikisi de aynı sınıftan olsalar bile, sınıfın değişkenlerine ve işlevlerine bellek tahsis edilme şekli farklıdır. Bellek, yalnızca nesne oluşturulduğunda sınıfın değişkenlerine tahsis edilir.

Sınıf bildirildiğinde bellek değişkenlere tahsis edilmez. Aynı zamanda, tek değişkenler farklı nesneler için farklı değerlere sahip olabilir, bu nedenle her nesne sınıfın tüm değişkenlerinin ayrı bir kopyasına sahiptir. Ancak bellek, sınıf bildirildiğinde işleve yalnızca bir kez tahsis edilir. Bu nedenle, nesnelerin işlevlerin ayrı ayrı kopyaları yoktur, her nesne arasında yalnızca bir kopya paylaşılır.

C++'da Statik Veri Üyeleri

Statik bir veri üyesi oluşturulduğunda, sınıfın tüm nesneleri arasında paylaşılan veri üyesinin yalnızca tek bir kopyası vardır. Statik olarak belirtilmedikçe, genellikle her nesnenin özniteliklerin ayrı bir kopyası vardır.

Statik üyeler, sınıfın herhangi bir nesnesi tarafından tanımlanmaz. Kapsam çözümleme işleci kullanılarak herhangi bir işlevin dışında özel olarak tanımlanırlar.

Statik değişkenlerin nasıl tanımlandığına bir örnek

```
class Employee
{

public:
    static int count; //returns number of employees
    string eName;

    void setName(string name)
    {
        eName = name;
        count++;
    }
};

int Employee::count = 0; //defining the value of count
```

C++'da Statik Yöntemler

Statik bir yöntem oluşturulduğunda, herhangi bir nesneden ve sınıftan bağımsız hale gelirler. Statik yöntemler yalnızca statik veri üyelerine ve statik yöntemlere erişebilir. Statik yöntemlere yalnızca kapsam çözümleme operatörü kullanılarak erişilebilir. Bir programda statik yöntemlerin nasıl kullanıldığına dair bir örnek gösterilmiştir.

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

class Employee
{

public:
    static int count; //static variable
    string eName;

    void setName(string name)
    {
        eName = name;
        count++;
    }

    static int getCount()//static method
    {
        return count;
    }
};

int Employee::count = 0; //defining the value of count

int main()
{
    Employee Harry;
    Harry.setName("Harry");
    cout << Employee::getCount() << endl;
}
```

Çıktı

```
1
```

Arkadaş fonksiyonları, sınıf içinde tanımlı olmasalar bile, sınıf üyelerinin özel verilerine erişme hakkına sahip olan fonksiyonlardır. Friend fonksiyonunun prototipinin yazılması gerekmektedir.

Bir sınıf içinde bir arkadaş işlevi bildirmek, o işlevi sınıfın bir üyesi yapmaz.

Arkadaş İşlevinin Özellikleri

- Sınıfın kapsamında değil, sınıfın bir üyesi olmadığı anlamına gelir.
- Sınıfın kapsamında olmadığı için o sınıfın nesnesinden çağrılamaz.
- İster genel ister özel erişim değiştiricisi altında olsun, sınıf içinde herhangi bir yerde bildirilebilir, herhangi bir fark yaratmaz
- Üyelere doğrudan adlarıyla erişemez, herhangi bir üyeye erişmek için (nesne_adı.üye_adı) gerekir.

Bir sınıf içinde bir arkadaş işlevi bildirmek için sözdizimi şöyledir:

```
class class_name
{
    friend return_type function_name(arguments);
};

return_type class_name::function_name(arguments)
{
    //body of the function
}
```

C++ Arkadaş Sınıfları

Arkadaş sınıfları, tanımlandıkları sınıfın özel üyelerine erişim izni olan sınıflardır. Burada dikkat edilmesi gereken en önemli nokta, sınıf başka bir sınıfın arkadaşı olursa, o sınıfın tüm private üyelerine erişebilir.

Bir sınıf içinde bir arkadaş sınıfı bildirmek için sözdizimi şöyledir:

```
class class_name
{
    friend class friend_class_name;
};
```

Yapıcı kelimesi oluşturma, oluşturucu olarakda kullanılmaktadır.

Yapıcı, sınıfla aynı ada sahip özel bir üye işlevdir. Yapıcının bir dönüş türü yoktur. Yapıcılar, sınıflarının nesnelerini başlatmak için kullanılır. Yapıcılar, bir nesne oluşturulduğunda otomatik olarak çağrılır.

C++'da Yapıcıların Özellikleri

- Sınıfın genel bölümünde bir yapıcı bildirilmelidir.
- Nesne oluşturulduğunda otomatik olarak çağrılır.
- Değer döndüremezler ve dönüş türleri yoktur.
- Varsayılan bağımsız değişkenlere sahip olabilir.

Bir yapıcının nasıl kullanıldığına bir örnek,

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

class Employee
{

public:
    static int count; //returns number of employees
    string eName;

    //Constructor
    Employee()
    {
        count++; //increases employee count every time an object is defined
    }

    void setName(string name)
    {
        eName = name;
    }

    static int getCount()
    {
        return count;
    }
};

int Employee::count = 0; //defining the value of count

int main()
{
    Employee Harry1;
```

```
    Employee Harry2;
    Employee Harry3;
    cout << Employee::getCount() << endl;
}
```

Çıktı:

```
3
```

C++'da Parametreli ve Varsayılan Yapıcılar

Parametreli yapıcılar, bir veya daha fazla parametre alan yapıcılardır. Varsayılan kurucular, parametre almayan kuruculardır. Bu, yukarıdaki örnekte yalnızca tanım sırasında çalışan adını ileterek yardımcı olabilirdi. Bu, setName işlevini kaldırmalıydı.

C++'da Yapıcı Aşırı Yüklemesi

Yapıcı aşırı yükleme, işlev aşırı yüklemesine benzer bir kavramdır. Burada, bir sınıfın farklı parametrelere sahip birden çok yapıcısı olabilir. Bir örneğin tanımı sırasında, bağımsız değişkenlerin sayısı ve türüyle eşleşen yapıcı yürütülür.

Örneğin, bir program 0, 1 ve 2 argümanlı 3 kurucudan oluşuyorsa ve biz yapıcıya sadece bir argüman iletirsek, bir argüman alan kurucu otomatik olarak çalıştırılır.

C++'da Varsayılan Argümanlara Sahip Yapıcılar

Yapıcının varsayılan bağımsız değişkenleri, yapıcı bildiriminde sağlananlardır. Yapıcı çağrılırken değerler sağlanmazsa, kurucu otomatik olarak varsayılan argümanları kullanır.

Varsayılan bağımsız değişkenlerin bildirilmesini gösteren bir örnek,

```
class Employee
{

public:
    Employee(int a, int b = 9);
};
```

Yapıcıyı C++ ile Kopyala

Kopya oluşturucu, başka bir nesnenin kopyasını oluşturan bir tür oluşturucudur. Bir nesnenin başka bir nesneye benzemesini istiyorsak, bir kopya yapıcı kullanabiliriz. Programda herhangi bir kopya yapıcı yazılmamışsa, derleyici kendi kopya yapıcısını sağlayacaktır.

Bir kopya yapıcı bildirmek için sözdizimi şöyledir:

```
class class_name
{
    int a;

public:
    //copy constructor
    class_name(class_name &obj)
    {
        a = obj.a;
    }
};
```

Kapsülleme, Nesne Yönelimli Programlamanın ilk ayağıdır. Veri niteliklerini ve yöntemlerini bir araya getirmek anlamına gelir. Amaç, hassas verileri kullanıcılardan gizli tutmaktır.

Kapsülleme, ihtiyaç duyulana kadar değiştirilemez hale gelmeleri için özniteliklerin her zaman özel yapılması gereken iyi bir uygulama olarak kabul edilir. Veriler sonuçta bunun bir sonucu olarak daha güvenlidir. Üyeler gizli hale getirildikten sonra, onlara erişme veya onları değiştirme yöntemleri bildirilmelidir.

Kapsüllemenin nasıl gerçekleştirildiğine bir örnek

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

class class_name
{
private:
    int a;

public:
    void setA(int num)
    {
        a = num;
    }

    int getA()
    {
        return a;
    }
};

int main()
{
    class_name obj;
    obj.setA(5);
    cout << obj.getA() << endl;
}
```

Çıktı:

```
5
```

C++'da Genel Erişim Belirteçleri

Genel erişim değiştiricisi altında bildirilen tüm değişkenler ve işlevler herkes tarafından kullanılabilir olacaktır. Hem sınıf içinden hem de sınıf dışından erişilebilirler. Nokta (.) operatörü, programda genel veri üyelerine doğrudan erişmek için kullanılır.

C++'da Özel Erişim Belirteçleri

Özel erişim değiştiricisi altında bildirilen tüm değişkenler ve işlevler yalnızca sınıf içinde kullanılabilir. Sınıf dışında herhangi bir nesne veya işlev tarafından kullanılmasına izin verilmez.

C++'da Korumalı Erişim Belirteçleri

Korumalı erişim değiştiricileri, özel erişim değiştiricilerine benzer, ancak korumalı erişim değiştiricilerine türetilmiş sınıfta erişilebilirken, özel erişim değiştiricilerine türetilmiş sınıfta erişilemez.

Aşağıda, erişim değiştiricilerin genel, özel ve korumalı türetildiğindeki davranışını gösteren bir tablo gösterilmektedir.

	Genel Türetme	Özel Türetme	Korumalı Türetme
Özel Üyeler	Miras alınmadı	Miras alınmadı	Miras alınmadı
Korumalı Üyeler	Korumalı	Özel	Korumalı
Kamu Üyeler	Paylaşımlı	Özel	Korumalı

- Sınıf, genel modda miras alınmışsa, özel üyeleri alt sınıftan miras alınamaz.
- Sınıf genel modda miras alınırsa, korunan üyeleri korunur ve bunlara alt sınıftan erişilebilir.
- Sınıf, genel modda miras alınmışsa, genel üyeleri geneldir ve alt sınıfın içinden ve sınıfın dışından erişilebilir.
- Sınıf özel modda miras alınırsa, özel üyeleri alt sınıftan miras alınamaz.
- Sınıf özel modda miras alınırsa, korunan üyeleri özeldir ve alt sınıftan erişilemez.
- Sınıf özel modda miras alınmışsa, genel üyeleri özeldir ve alt sınıftan erişilemez.
- Sınıf korumalı modda miras alınırsa, özel üyeleri alt sınıftan miras alınamaz.
- Sınıf, korumalı modda miras alınırsa, korumalı üyeleri korunur ve bunlara alt sınıftan erişilebilir.
- Sınıf korumalı modda miras alınırsa, genel üyeleri korunur ve alt sınıftan erişilebilir.

C++'da polimorfizm

Poli, birkaç anlamına gelir ve morfizm biçim anlamına gelir. Polimorfizm, birkaç formu olan bir şeydir veya bunu tek isim ve çoklu formlar olarak da söyleyebiliriz. İki tür polimorfizm vardır: - Derleme zamanı polimorfizmi - Çalışma zamanı polimorfizmi

Derleme Zamanı Polimorfizmi

Derleme zamanı polimorfizminde hangi fonksiyonun çalışacağı zaten bilinmektedir. Derleme zamanı polimorfizmi ayrıca erken bağlama olarak da adlandırılır; bu, zaten işlev çağrısına bağlı olduğunuz ve bu işlevin çalışacağını bildiğiniz anlamına gelir.

İki tür derleme zamanı polimorfizmi vardır:

- İşlev Aşırı Yüklemesi Bu, birden fazla fonksiyon oluşturmamızı sağlayan bir özelliktir ve fonksiyonların isimleri aynıdır ancak parametrelerinin farklı olması gerekir. Bir programda işlev aşırı yüklemesi uygulandığında ve işlev çağrıları yapıldığında, derleyici hangi işlevlerin yürütüleceğini bilir.
- Operatör Aşırı Yüklemesi

Bu, bazı belirli görevler için çalışan operatörleri tanımlamamızı sağlayan bir özelliktir. + operatörünü kullanarak iki diziyi birleştirerek toplayabilir ve aynı anda aritmetik olarak iki sayısal değer ekleyebiliriz. Bu, operatörün aşırı yüklenmesidir.

Çalışma Zamanı Polimorfizmi

Çalışma zamanı polimorfizmi ile derleyicinin kod yürütüldüğünde ne olacağı hakkında hiçbir fikri yoktur. Çalışma zamanı polimorfizmi aynı zamanda geç bağlama olarak da adlandırılır. Çalışma zamanı polimorfizmi, işlev çağrılarına çalışma zamanında karar verildiği için yavaş kabul edilir. Çalışma zamanı polimorfizmi, sanal işlevlerden elde edilebilir.

C++'da Sanal İşlevler

Bir sanal anahtar sözcük kullanılarak bildirilen temel sınıftaki bir üye işleve sanal işlev denir. Türetilmiş sınıfta yeniden tanımlanabilirler. Üst sınıfta bulunan ancak alt sınıfta yeniden tanımlanan bir işleve sanal işlev denir.

Bir fonksiyonun sanal olabilmesi için statik olmaması gerekir.