BTEP243 – Ders 3

Sınıflar ve Nesneler

Nesne tabanlı programlamada, programlamanın temeli sınıflardır (class). Nesnelerin yaratılmasında "taslak" (blueprint) görevi görür.

- **Sınıflar**;
  - Nesnelerin özelliklerinin / değişkenlerin (data members) ve fonksiyonlarının (member functions / methods) nasıl ilişkilendirileceğini belirler.
  - Bilgi gizleme (information hiding) mekanizmasına sahiptir.
- **Nesneler** ise gerçek dünyadaki fiziksel varlıkları modeller.

Sınıflarda Üye Erişim Berlileyicileri (Member Access Specifiers)

- **public:** Bir sonraki üye erişim belirleyicisine kadar bütün özellikler (data members) veya üye fonksiyonlar (member functions), sınıfın nesnesini yaratmış bütün kullanıcılar tarafından erişilebilirler.
- **private:** Bir sonraki üye erişim belirleyicisine kadar bütün özellikler (data members) veya üye fonksiyonlar (member functions), ancak ayni sınıfın üye fonksiyonları tarafından erişilebilirler.
- **protected:** Bir sonraki üye erişim belirleyicisine kadar bütün özellikler (data members) veya üye fonksiyonlar (member functions), ayni sınıfın üye fonksiyonları ve bu sınıftan türetilen (inheritance) sınıfların üyeleri tarafından erişilebilirler.

Yapıcı (Constructor) Fonksiyonlar

Oluşturduğumuz sınıflar içerisinde yer alan herhangi bir değişkene başlangıç değeri verebilmek için yapıcı fonksiyonlar kullanılır. Yapıcı fonksiyonların en temel ve bilinmesi gereken özellikleri aşağıda sıralanmıştır:

- Bir sınıf için nesne oluşturulduğu anda o sınıfın ilgili yapıcı fonksiyonu otomatik olarak çağırılmaktadır.
- Yapıcı fonksiyonun ismi sınıf ismi ile aynı olmak zorundadır.
- Birden fazla yapıcı fonksiyon tanımlanabilmektedir. (Varsayılan, Parametreli ve Kopyalayıcı). Ancak yine de hepsinin ismi sınıf ismi ile aynı olmak zorundadır.
- Yapıcılar değer döndürmezler bu yüzden "return" gibi bir geri dönüş komutu içermezler. Void (boş) bile olamazlar.
- Nesne oluşturulacağı zaman derleyici tarafından sadece bir kere çalıştırılırlar.
- Parametre alabilirler.
- Herhangi bir yapıcı fonksiyon tanımlanmaz ise, derleyici otomatik olarak varsayılan bir yaratıcı oluşturup nesneler için hafızada yer tutmalarını sağlar.

"class" Yazım Kuralı:

```
class <sınıf_ismi>
{
        private:
                //değişkenler veya bazı fonksiyonlar
        public:
                //erişilmesi istenen değişkenler ve fonksiyonlar (yapıcılar, yıkıcılar ve diğerleri)
        protected:
                //türetilen sınıflar tarafından erişilmesi istenen değişkenler ve fonksiyonlar
};
```

Örnek 1:

//time.h

```
class time{
private:
        int dakika,saat;
public:
//varsayilan yapici fonksiyon (parametresiz)
        time()
        {
                dakika=0;
                saat=0;
        }
};
#include<iostream>
using namespace std;
#include"time.h"
void main()
{
        time okulsaati;

        system("pause");
}
```

Derleyici, saat sınıfı için okulsaati nesnesi yaratıldığı anda varsayılan yapıcı fonksiyonunu otomatik olarak çalıştırır ve saat ve dakika değerlerinin başlangıç değerlerini 0 olarak belirler.

Erişim Operatörü (Binary Scope Resolution Operator) " :: " ve Nokta Operatörü (Dot operator)

Üye fonksiyonlar sınıf içerisinde tanımlanırlar, fakat sınıf dışında tanımlamak daha iyidir. Bu tanımlama için erişim operatörü (::) kullanılır. Bunun yanı sıra, sınıfın üye fonksiyonlarına (public üyeler) nokta operatörü ile erişilebilir.

Aşağıdaki örnekte her iki operatörün kullanımı gösterilmiştir.

Örnek 2:

```
class time{
private:
        int dakika,saat;
public:
//fonksiyonaların prototipleri sınıf içerisinde verilmelidir.
        time();
        void goster();
};
//fonksiyonaların tanımları  sınıf dışında erişim operatörü kullanılarak yapılmalıdır.
time::time()
{
        dakika=0;
        saat=0;
}
void time::goster( )
{
        cout<<saat<<":"<<dakika<<endl;
}
```

```
#include<iostream>
using namespace std;
#include"time.h"
void main()
{
	time okulsaati;
	okulsaati.goster( );

	system("pause");
}
```

Sınıf üyelerine erişim kontrolü

Bütün değişkenler (data member) private olarak tutulmalıdır. Eğer gerekirse, bu özelliklerin değiştirilebilmesi için "**set/setter"** foksiyonları ve ulaşabilmek için de "**get/getter"** fonksiyonları kullanılabilir.

Set Fonksiyonları: Bu fonksiyon türü, herhangi bir nesnenin özelliğini değiştirebilmek için kullanılan fonksiyondur. Genellikle herhangi bir değer döndürmezler ancak yeni değeri parametre olarak kabul ederler.

Get Fonksiyonları: Bu fonksiyon türü, herhangi bir nesnenin özelliğine ulaşabilmek için kullanılan fonksiyondur. İlgili özelliği geri döndürmekle yükümlüdürler.

Örnek 3:

//time.h

```
class time{
private:
	int dakika,saat;
public:
	time()
	{
		dakika=0;
		st=0;
	}
	void setDakika(int d)
	{
		dakika=d;
	}
	void setSaat(int s)
	{
		saat=s;
	}
	int getDakika()
	{
		return dakika;
	}
	int getSaat()
	{
		return saat;
	}
	void goster()
	{
		cout<<saat<<":"<<dakika<<endl;
	} };
#include<iostream>
using namespace std;
#include"time.h"
```

```
void main()
{
	time okulsaati;
	cout<<okulsaati.getSaat()<<":"<<okulsaati.getDakika()<<endl;
	okulsaati.setDakika(30);
	okulsaati.setSaat(8);

	okulsaati.goster();

	saat uykusaati(23);
	uykusaati.goster();

	system("pause");
}
```

Diğer Yapıcı Türleri – Parametreli Yapıcılar (Parameterized / Overloaded Constructors)

Yapıcılarda aşırı yükleme (overload) kullanımı mümkündür. Bir yapıcı parametre tipi ve parametre sayısı ile diğer yapıcılardan ayırt edilebilirler.

Örnek 4:

```
class time{
private:
	int dakika,saat;
public:
	//varsayilan yapıcı fonksiyon
	time()
	{
		dakika=0;
		saat=0;
	}
	//parametreli yapıcı fonskiyonlar
	time(int s,int d)
	{
		dakika=d;
		saat=s;
	}
	time(int s)
	{
		saat=s;
		dakika=0;
	}
	void setDakika(int d)
	{
		dakika=d;
	}
	void setSaat(int s)
	{
		saat=s;
	}
	int getDakika()
	{
		return dakika;
	}
	int getSaat()
	{
		return saat;
```

```
	}
	void goster()
	{
		cout<<saat<<":"<<dakika<<endl;
	}
	void gosterS()
	{
		cout<<saat%12<<":"<<dakika;
		if(saat>12)
			cout<<"p.m.";
		else
			cout<<"a.m.";
	}
};
#include<iostream>
using namespace std;
#include"saat.h"
void main()
{
	time okulsaati , yemeksaati(12,30), uykusaati(23);
	okulsaati.goster();
	okulsaati.gosterS();
	cout<<endl;

	yemeksaati.goster();
	yemeksaati.gosterS();
	cout<<endl;

	uykusaati.goster();
	uykusaati.gosterS();
	cout<<endl;

	system("pause");
}
```

Yukarıdaki örnekte yaratılan 3 nesne farklı yapıcıların çalıştırılmasını sağlamaktadır.

- **okulsaati** nesnesi varsayılan yapıcıyı (time( ))
- **yemeksaati(12,30)** nesnesi 2 parametre içerdiğinden dolayı, iki parametreli yapıcıyı (**time(int s,int d)**)
- **uykusaati(23)** nesnesi ise tek parametre içerdiğinden, tek parametreli yapıcıyı (**time(int s)**)

kullanmaktadır.

Yapıcılarda Varsayılan (Default) değer kullanımı

Eğer bir yapıcı parametre içermezse, bütün varsayılan değerler güvenli değerlere atanır. Bu tip yapıcılara varsayılan yapıcılar denir. Her bir sınıfın sadece bir varsayılan yapıcısı olabilir.

Örnek 5:

```
classtime{
private:
	int dakika,saat;
public:
	time(int s, int d=0)
	{
		saat=s;
		dakika=d;
	}
};
```

Önemli Not:
Parametreli yapıcı fonksiyonda eğer tüm parametrelere varsayılan değer atanırsa, bu varsayılan yapıcı fonksiyon olarak da kabul edilir. Ayrıca parametresiz bir yapıcı fonksiyon yaratmak hataya yol açar.

Örnek:
time(int s=0, int d=0) varsayılan fonksiyona "time( )" denktir.

Kopyalayıcı Yapıcı Fonksiyonlar (Copy Constructor)

Yeni oluşturulan nesnenin, varolan bir nesnenin özelliklerine sahip olmasının istendiği durumlarda kopyalayıcı yapıcılar kullanılabilir. Eğer programcı herhangi bir kopyalayıcı yapıcı oluşturmaz ise, gereken durumlarda derleyici bunu otomatik olarak yapar. Derleyici eşitlik operatörü "=" (assignment operator) kullanıldığı durumlarda kopyalayıcı yapıcıyı oluşturur.

Örnek 6:
```
void main( )
{
Time sabah(10,15);
Time aksam = sabah; //eşitlik operatörü kopyalayıcı yapıcının çalışmasını sağlayıp, aksam
                    nesnesine de sabah nesnesinin saat ve dakika değerlerini kopyalar.
}
```

Programcı tarafından oluşturulan "Kopyalayıcı" yapıcılar

Yazım Kuralı:
*Sınıf_adi(**const** sınıf_adi & nesne);*

Yukarıdaki ifadede "nesne" varolan bir nesneyi temsil eder. Referans operatörü kullanılarak yeni oluşturulan nesneye varolan nesnenin kopyalanması sağlanır. "const" anahtar sözcüğünün kullanılma sebebi ise özelliklerin kopyalanırken herhangi bir değişikliğe uğramalarını önlemektir.

Örnek 7:
```
class time{
private:
        int dakika, saat;
public:
        //kapyalayıcı yapıcı fonksiyon
        time(const time & nesne)
        {
                saat=nesne.saat;
                dakika=nesne.dakika;
        }
```

Yıkıcı Fonksiyon (Destructor)

Sınıfların nesnelerin hafızadan silmek için kullanılan fonksiyonlara yıkıcı fonksiyonlar denir.

- Yıkıcı fonksiyonun ismi tilde "~" simgesi ile başlar ve sınıf ismi ile sonlanır.

- Yıkıcılar, yapıcılar gibi herhangi bir değer döndürmezler, bu yüzden “return” gibi bir geri dönüş komutu içermezler. Void (boş) bile olamazlar.
- Program sonlandığı zaman derleyici tarafından her nesne için bir kere çalıştırılırlar.
- Parametre almazlar.
- Parametre alamayacağı için her sınıfın sadece bir tane yıkıcı fonksiyonu olabilir.
- Herhangi bir yapıcı fonksiyon tanımlanmaz ise, derleyici otomatik olarak varsayılan bir yıkıcı oluşturup nesneleri hafızadan siler.
- Yıkıcıların kullanımı özellikle dinamik nesne yaratıldığında çok büyük önem taşır.

Örnek 8:

```
class time{
private:
        int dakika,saat;
public:
        time()
        {
                dakika=0;
                saat=0;
        }
        ~time()
        {
                cout<<”Nesne başarı ile silinmiştir.”<<endl;
        }
```