BÖLÜM 12

NESNEYE YÖNELIK PROGRAMLAMA

ANAHTAR KAVRAMLAR

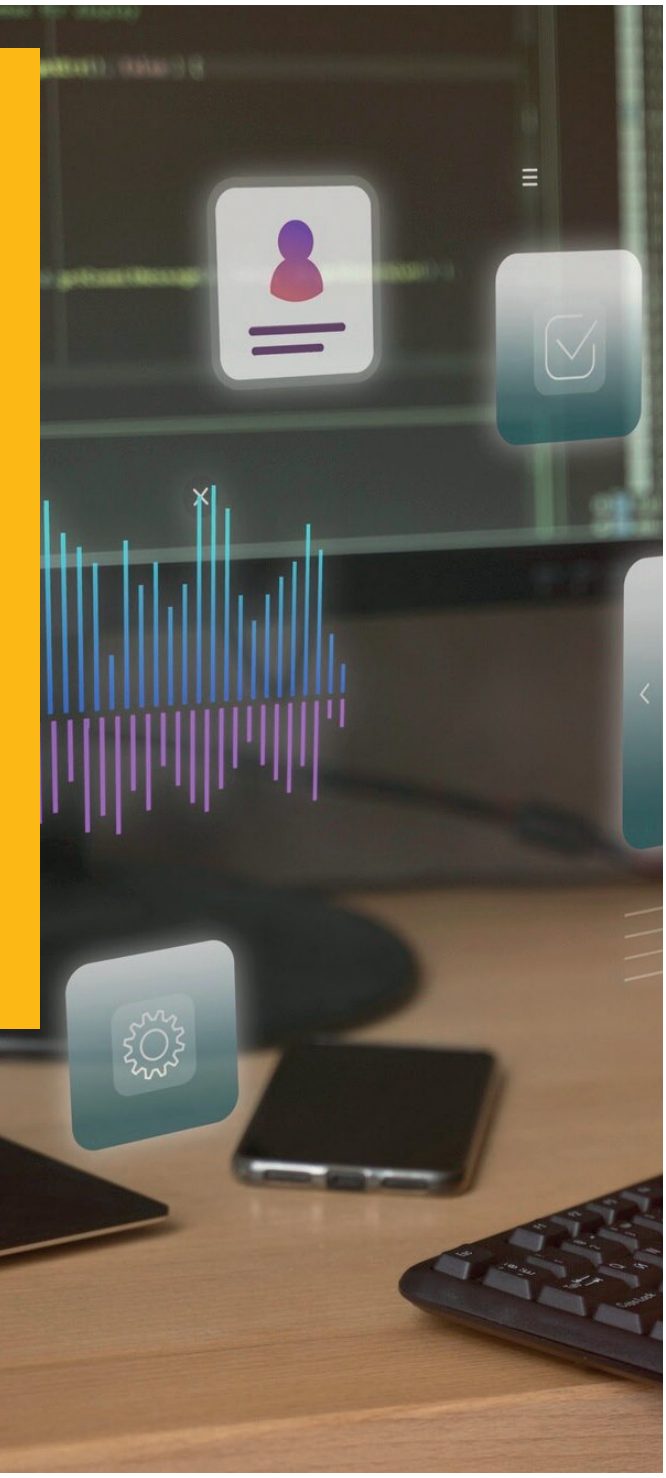
Sınıf, Üye Fonksiyonlar, Veri Üyeleri, Yapıcı Fonksiyon, Yıkıcı Fonksiyon, İşleç Aşırı Yükleme, Bileşim, Katılım, Çok Biçimlilik

ÖĞRENME HEDEFLERİ

1. Üye fonksiyonları ve veri üyeleri ile sınıf tanımı yapabilir.
2. Sınıf kullanarak kullanıcı tanımlı veri tipi oluşturabilir.
3. Oluşturulan yeni veri tipinden nesneler tanımlayabilir.
4. İşleç aşırı yükleme gerçekleştirerek, işleçleri sınıflardan tanımlanan nesneler erinde kullanabilir.
5. Bileşim işlemi ile bir sınıfta, baska bir sınıfn nesnesini veri üyesi olarak kullanabilir.
6. Kalıtm kullanarak temel sınıflardan türetilmiş sınıflar oluşturabilir.
7. Çok biçimlilik kullanarak türetilmiş sınıf nesnelerine temel sınıf işaretçileri veya referansları ile ulaşabilir.

ONİKİNCİ BÖLÜM

GİRİŞ



Bu bölümde önce sınıf tanımlamasının nasıl gerçekleştirileceği anlatılacaktır. Sınıf kullanarak kullanıcı tanımlı yeni veri tipleri oluşturulabilir. Bu şekilde C++ programlama dilinde var olan yerleşik veri tiplerine yeni veri tipleri eklenebilir ve dilin kapasitesi genişletilebilir. Sınıf tanımlamalarında gerekli üye fonksiyonlar ve veri üyeleri ve bu üyelere erişim seviyeleri detaylı olarak tartışılacaktır. Yapıcı ve yıkıcı fonksiyonlar ve kullanım amaçları sınıflar açısından önemlidir. Nesneler tanımlandığında otomatik olarak yapıcı fonksiyonlar, yok edildiğinde otomatik olarak yıkıcı fonksiyonlar çağrılır. En son oluşturulan nesne, ilgili kapsam alanı tamamlandığında ilk yok edilecek nesne olacaktır. İşleç aşırı yükleme gerçekleştirilerek yerleşik veri tiplerinde kullanılan işleçler, sınıflardan tanımlanabilecek nesneler üzerinde kullanılabilir. Bu bölümde toplama ve ekrana yazdırma işleçlerinin aşırı yükleme ile Rasyonel sınıfına ait nesneler üzerinde kullanılması örnek olarak gösterilecektir. Bir sınıfın nesnesinin bir başka sınıf içerisinde veri üyesi olarak tanımlanması bileşim olarak isimlendirilir. Bir Tarih sınıfı nesnesinin Personel sınıfında veri üyesi olarak tanımlanması bu bölümdeki örneklerden birisidir. Bazı sınıflar var olan tanımlanmış sınıflar kullanılarak tanımlanabilir. Var olan sınıf temel sınıf olarak tanımlanır ve üyeleri yeni tanımlanan türemiş sınıfın de üyeleri olur. Türemiş sınıfın kendine ait üye fonksiyonları ve veri üyeleri olabilir. Tüm kalıtım hiyerarşisindeki sınıf nesnelerine temel sınıf işaretçileri veya referansları kullanarak erişmek mümkündür. Bu şekilde tüm nesneler farklı türemiş sınıflara da ait olsalar tek bir listede veya dizide tutulabilir. Sanal fonksiyonlar yardımı ile dinamik olarak doğru nesnenin ilgili fonksiyonu çağrılarak tek bir liste üzerinden tüm sınıfların nesnelerine erişmek mümkün olmaktadır. Bu yaklaşım çok biçimlilik olarak tanımlanır.

1. SINIFLAR VE NESNELER

C++ programlama dilinde geliştiricilerin kullanabilmesi için tanımlanmış yerleşik veri tipleri vardır. Bu veri tipleri int, float, double ve char başta olmak üzere daha önceki bölümlerde anlatılmıştır. Bu veri tipleri sayesinde tam sayı, kesirli sayı ve karakter gibi nesneleri tanımlamak ve programlarda uygun durumlarda kullanmak mümkündür. Bu veri türleri üzerinde gerçekleştirilebilecek işlemler de tanımlıdır. Örnek olarak; tam sayılar üzerinde, toplama, çıkartma, çarpma, bölme vb. işlemler tanımlıdır. Bu işlemler çoğunlukla ikili işlemlerdir ve iki adet işlenen gereklidir. Ayrıca tekli işleçler de tam sayılar üzerinde kullanılabilir. Örneğin (++) ve (--) işleçleri sadece bir tam sayı işlenen üzerinde çalışırlar.

Bir C++ programında kullanabileceğimiz yerleşik veri tipleri tüm ihtiyaçları karşılama konusunda bazı durumlarda yetersiz kalabilmektedir. Birçok problemin çözümünde tam sayı, kesirli sayı, karakter vb. veri tipleri ile oluşturabileceğimiz nesnelerin dışında farklı nesnelere gereksinim söz konusudur. Karmaşık sayılar ve rasyonel sayılar gibi matematiksel nesneleri tanımlamak için yeni veri tipleri kullanabilme yeteneği programlama diline, problemlerin çözümünde büyük kolaylıklar getirecektir. Ayrıca yeni tanımlanabilecek nesneler sadece matematiksel sayılar ile sınırlı olmayacaktır. Günlük hayatta karşımıza çıkabilecek her türlü nesne yeni bir veri tipi olarak tanımlanabilme imkanına sahip olabilecektir. Örneğin, bir öğrenci nesnesi, bir çalışan nesnesi, koordinat sisteminde bir nokta nesnesi vb. nesneler rahatlıkla tanımlanabilecek ve kullanılabileceklerdir. Bu tür durumlardan dolayı yerleşik veri tiplerine ek olarak kullanıcı tanımlı yeni veri tipleri söz konusu olmuştur. Kullanıcı tanımlı yeni veri tipleri C++programlama dilinde sınıf yapıları kullanılarak gerçekleştirilir. Bu veri tipleri için, ilgili nesneyi tanımlayan veri üyeleri ve bu nesne üzerinde gerçekleştirilebilecek işlemleri gösteren üye fonksiyonlar sınıf yapısı içinde tanımlanır.

Program12.1.cpp rasyonel sayılar için yazılmış bir sınıf tanımı içermektedir.

```
//Program12.1.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
    public:
        Rasyonel(){
            pay=1;
            payda=2;
        }
        void setRasyonel(int a, int b)
        {
            pay=a;
            payda=b;
        }
        void yazdirRasyonel()
        {
            cout<<pay<<"/"<<payda<<endl;
        }
```

```
        private:
            int pay;
            int payda;
};
int main()
{
        Rasyonel r1;
        r1.yazdirRasyonel();
        r1.setRasyonel(3,5);
        r1.yazdirRasyonel();
        return 0;
}
```

Program12.1.cpp Çıktısı

```
1/2
3/5
```

Burada verilen Rasyonel sınıfını kullanarak bazı temel kavramları açıklayalım. Bir sınıf tanımlamak için class anahtar kelimesi kullanılır. class kelimesinden sonra sınıfın adı yazılır. Daha sonra kıvrımlı parantezler içerisinde sınıf tanımı gerçekleştirilir. public ve private olmak üzere temel iki farklı erişim seviyesi mevcuttur. Üye fonksiyonlar genellikle public olarak tanımlanırlar. public tanımlanan bir fonksiyon veya değişken sınıf tanımının dışında da ulaşılabilir. private erişim seviyesinde yer alan üyeler ise sadece sınıfın kapsam alanında erişilebilirler. Varsayılan erişim seviyesi private olarak belirlenmiştir. Dolayısıyla herhangi bir erişim seviyesi belirtilmemişse ilgili üyeler private olur.

Sınıfın ismi ile aynı isme sahip özel bir üye fonksiyon vardır. Bu fonksiyon yapıcı fonksiyon (cons tructor) olarak isimlendirilir. Her hangi bir sınıf için bir değişken tanımlandığında yapıcı fonksiyon çağrılır. Program12.1.cpp'de main fonksiyonda r1 isminde bir Rasyonel nesnesi oluşturulur. Bu nesne oluşturma işlemi Rasyonel sınıfının yapıcı fonksiyonunu çağırır. Yapıcı fonksiyonun içinde pay ve payda veri üyelerinin değerleri sırasıyla 1 ve 2 olarak atanmıştır. Burada da görüldüğü gibi private olarak tanımlanan üyeler sınıf fonksiyonları içinde doğrudan kullanılabilirler. setRasyonel üye fonksiyonu iki tam sayı parametre alır. Parametreler fonksiyon tanımı içerisinde veri üyelerine atanır. Fonksiyon her hangi bir değer çevirmediği için dönüş tipi void olarak belirlenmiştir.

yazdirRasyonel fonksiyonu parametre almaz. Fonksiyonu çağıran nesnenin veri üyeleri ekrana rasyonel sayı formatında yazdırılır. Bu fonksiyon da her hangi bir değer çevirmez ve bu nedenle dönüş tipi void olarak yazılmıştır.

Bir üye fonksiyonu çağırmak için ilgili sınıf nesnesi '.' işleci ve fonksiyon ismi ile beraber kullanılır.

```
r1.yazdirRasyonel();
```

ifadesinde yazdirRasyonel fonksiyonu r1 nesnesi ile çağrılır. r1 nesnesi ilk oluşturulduğunda yapıcı fonksiyonlar çağrılmış; pay ve payda değerleri 1 ve 2 olarak atanmıştı. Bir sınıftan oluşturulan tüm nesnelerin kendilerine ait veri üyeleri vardır. Dolayısı ile bu örnekte de r1 nesnesinin kendine ait pay ve payda değerleri bulunmaktadır. Bir nesnenin bellekte kapladığı alan bir anlamda sahip olduğu veri üyelerinin boyutu kadardır. yazdirRasyonel fonksiyonu çalıştığında Rasyonel sınıfının r1 nesnesine ait pay ve payda değerleri ekrana yazdırılır.

r1.setRasyonel(3,5);

ifadesi ile setRasyonel fonksiyonu r1 nesnesi ile çağırılır. Yapıcı fonksiyon ile başlangıç değeri 1 ve 2 olarak atanan r1 nesnesinin pay ve payda veri üyelerine parametre olarak gönderilen 3 ve 5 değerleri atanır. yazdirRasyonel fonksiyonu tekrar çağrıldığı zaman ekrana r1 nesnesinin veri üyelerinin yeni değerleri yazdırılır.

r1 nesnesine ait veri üyeleri private erişim seviyesi ile tanımlandığından doğrudan main fonksiyon içinde ulaşılamazlar. Ancak üye fonksiyonlar public olarak tanımlandığından main fonksiyon içinde doğrudan erişilebilirler. private olarak tanımlanan erişime sahip üyeler sınıf kapsama alanında, örneğin public tanımlanan üye fonksiyonlar içinde doğrudan erişilebilirler.

DİKKAT EDELİM

Yapıcı fonksiyonlar bir nesne oluşturulduğunda otomatik olarak çağrılırlar. Bu fonksiyonlar sınıfın ismi ile aynı isme sahiptirler.

```
//Program12.2.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
    public:
        Rasyonel(){
            pay=1;
            payda=2;
        }
        Rasyonel(int i, int j){
            pay=i;
            payda=j;
        }
        void setRasyonel(int a, int b){
            pay=a;
            payda=b;
        }
        void setPay(int a){
            pay=a;
        }
        void setPayda(int a){
            payda=a;
        }
```

```
        int getPay(){
                return pay;
        }
        int getPayda(){
                return payda;
        }
        void yazdirRasyonel()
        {
                cout<<pay<<"/"<<payda<<endl;
        }
private:
        int pay;
        int payda;
};

int main()
{
    Rasyonel r1;
    Rasyonel r2(4,7);
    r1.yazdirRasyonel();
    r2.yazdirRasyonel();
    r1.setPay(5);
    r1.setPayda(9);
    r1.yazdirRasyonel();
    r1=r2;
    r1.yazdirRasyonel();
    cout<<r1.getPay()<<"/"<<r1.getPayda()<<endl;
    return 0;
}
```

Program12.2.cpp Çıktısı

```
1/2
4/7
5/9
4/7
4/7
```

Program12.2.cpp'de sınıf tanımı içerisinde bireysel olarak üye verilere değer atayan ve değerleri çeviren üye fonksiyonlar tanımlanmıştır. Bu fonksiyonlar set ve get fonksiyonları olarak isimlendirilirler. private olarak tanımlanan üye verileri doğrudan sadece sınıf kapsama alanında erişilebilirler. Bu nedenle sınıf kapsama alanı dışında örneğin main fonksiyonda bu verilere erişebilmek için bu verileri bireysel olarak döndüren fonksiyonlara ihtiyaç vardır. Örnek programımızda pay ve payda üye verilerini çeviren getPay ve getPayda fonksiyonları verilmiştir. Aynı zamanda bu verilere bireysel olarak değer atayan setPay ve setPayda üye fonksiyonlarının kullanımı da gösterilmiştir.

```
//Program12.3.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
	public:
		Rasyonel(){
			pay=1;
			payda=2;
			cout<<"Yapici (constructor) fonksiyon cagrildi."<<" Pay:"<<pay<<"
			Payda:"<<payda<<endl;
		}
		Rasyonel(int i, int j){
			ppay=i;
			ppayda=j;
			pcout<<"Yapici (constructor) fonksiyon cagrildi."<<"
			Pay:"<<pay<<" Payda:"<<payda<<endl;
		}
		~Rasyonel()
		{
			cout<<"Yikici (destructor) fonksiyon cagrildi."<<" Pay:"<<pay<<"
			Payda:"<<payda<<endl;
		}
		void yazdirRasyonel()
		{
			cout<<pay<<"/"<<payda<<endl;
		}
	private:
		int pay;
		int payda;
};
int main()
{
	Rasyonel r1;
	Rasyonel r2(4,7);
	Rasyonel *r1Ptr, *r2Ptr;
	r1Ptr=&r1;
	r2Ptr=&r2;
	r1Ptr->yazdirRasyonel();
	r2Ptr->yazdirRasyonel();
	return 0;
}
```

Program12.3.cpp Çıktısı

```
Yapici (constructor) fonksiyon cagrildi. Pay:1 Payda:2
Yapici (constructor) fonksiyon cagrildi. Pay:4 Payda:7
1/2
4/7
Yikici (destructor) fonksiyon cagrildi. Pay:4 Payda:7
Yikici (destructor) fonksiyon cagrildi. Pay:1 Payda:2
```

Program12.3.cpp'de yıkıcı fonksiyon (destructor) kullanımı gösterilmiştir. Bir nesne tanımlandığı kapsama alanı tamamlanınca bellekten silinir. Yıkıcı fonksiyonlar tam bu noktada otomatik olarak çağrılırlar. Yapıcı fonksiyonlar gibi özel fonksiyonlardır ve sınıfın ismi ile aynı isimde olmak zorundadırlar. İsim olarak yapıcı fonksiyondan farkları '~' karakteri ile başlamalarıdır. Nesne ile ilgili gerçekleştirilmesi gereken bellek yönetimi işlemlerinin tamamlanmasını sağlayan ifadeler bu fonksiyonlar içinde yer alır. Örneğin bir üye verisi dinamik bellek alanı kullanıyorsa bu alanın geriye verilme işlemi bu fonksiyonlar içine yazılabilir.

Yapıcı fonksiyonlar bir nesne yaratıldığında, yıkıcı fonksiyonlar ise bir nesne silindiğinde çağrılırlar. Program.12.3.cpp'de yapıcı ve yıkıcı fonksiyonlar içinde ekrana, ilgili fonksiyonların çağrıldığına yönelik ifadeler yazdırılmıştır. Bu şekilde yapıcı ve yıkıcı fonksiyonların çağrılma sıraları takip edilebilir. Bu örnekte de görüldüğü gibi bir kapsam alanında en son oluşturulan nesne ilk yok edilen nesne olmaktadır. Önce r1 nesnesi daha sonra r2 nesnesi oluşturulmuş ancak önce r2 nesnesi daha sonra r1 nesnesi silinmiştir.

Ayrıca Program12.3.cpp'de bir sınıf tipinden tanımlanan işaretçilerin kullanımı gösterilmiştir. Diğer yerleşik veri tiplerinde olduğu gibi kullanıcı tanımlı veri tipleri için de işaretçiler tanımlanabilir. Bu örnekte r1Ptr ve r2Ptr işaretçilerine sırasıyla r1 ve r2 nesnelerinin adresleri atanmıştır. Bir işaretçi kullanarak üye fonksiyonu çağırmak için '->' işleci kullanılır. Bu örnekte yazdirRasyonel fonksiyonu r1Ptr ve r2Ptr işaretçileri ile çağrılmıştır.

```
//Program12.4.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
    public:
        Rasyonel(){
            pay=1;
            payda=2;
        }
        Rasyonel(int i, int j){
            pay=i;
            payda=j;
        }
        void setRasyonel(int a, int b){
```

```
                pay=a;
                payda=b;
        }
        Rasyonel toplaRasyonel(Rasyonel r)
        {
                Rasyonel temp;
                temp.pay=payda*r.pay+r.payda*pay;
                temp.payda=payda*r.payda;
                return temp;
        }
        void yazdirRasyonel()
        {
                cout<<pay<<"/"<<payda<<endl;
            }
        private:
            int pay;
            int payda;
};
int main()
{
        Rasyonel r1;
        Rasyonel r2(4,7);
        Rasyonel r3;
r1.yazdirRasyonel();
        r2.yazdirRasyonel();
        r3=r1.toplaRasyonel(r2);
        r3.yazdirRasyonel();
        return 0;
}
```

Program12.4.cpp Çıktısı

```
1/2
4/7
15/14
```

Program12.4.cpp'de toplaRasyonel fonksiyonunun kullanımı gösterilmiştir. Bu fonksiyon ile iki rasyonel sayının toplanması gerçekleştirilmiştir. toplaRasyonel fonksiyonu bir sınıf nesnesinin fonksiyonlarda dönüş tipi ve parametre olarak kullanılabilmesini göstermesi açısından iyi bir örnektir. toplaRasyonel fonksiyonunun dönüş tipi Rasyonel sınıfının nesnesi olarak belirlenmiştir. Aynı zamanda fonksiyon bir Rasyonel sınıfı nesnesini parametre olarak almaktadır.

r3=r1.toplaRasyonel(r2);

ifadesinde toplaRasyonel fonksiyonu r1 nesnesi tarafından çağrılmıştır. Dolayısı ile bu fonksiyonun içinde doğrudan ulaşılan pay ve payda veri üyeleri r1 nesnesinin üyeleridir. toplaRasyonel fonksiyonu-

nun aldığı parametre r2 nesnesi (parametre listesinde r olarak listelenmiş, değer ile çağırma yaklaşımı kullanılmıştır), r1 nesnesi ile toplanmak istenen rasyonel sayıdır. Bu nesnenin pay ve paydasına fonksiyon içinde erişmek için r.pay ve r.payda ifadeleri kullanmak gerekir. Fonksiyonun içinde temp isminde bir Rasyonel nesnesi oluşturulmuş ve bu nesnenin pay ve paydası iki nesnenin toplanması ile bulunmuştur. Elde edilen nesne toplaRasyonel fonksiyonunun sonucu olarak çevrilmiştir. main fonksiyonda çevrilen bu değer r3 nesnesine atanmıştır ve yazdirRasyonel fonksiyonu ile ekrana yazdırılmıştır.

ARAŞTIRALIM

Sınıf üye fonksiyonlarının sınıf tanımlaması dışında geliştirilmesi konusunu araştıralım.

2. İŞLEÇLERİN AŞIRI YÜKLENMESİ

Program14.4.cpp'de toplaRasyonel fonksiyonu kullanılarak iki rasyonel sayı toplanmıştır. Ancak iki tam sayıyı toplamak için '+' işleci kullanılabilirken, iki rasyonel sayıyı toplayabilmek için sayılardan biri ile bir üye fonksiyon çağırmak gerekmektedir. Yerleşik veri tipleri ile kullanılabilen işleçleri kullanıcı tanımlı veri tipleri ile kullanabilmek için işleçlerin aşırı yüklenmesi yaklaşımı geliştirilmiştir. Temel amaç; r3=r1+r2; ifadesi ile iki rasyonel sayıyı toplamaktır.

Bir tam sayıyı ekrana yazdırmak için '<<' işleci kullanılmaktadır. Ancak bir rasyonel sayıyı ekrana yazdırmak için Program14.4.cpp'de yazdirRasyonel fonksiyonu kullanılmıştır. Yerleşik veri tiplerini yazdırmak için kullandığımız '<<' işlecini kullanıcı tanımlı tipler için de kullanılabilmesi beklenmektedir. '+' işlecinde olduğu gibi '<<' işleci de, işleçlerin aşırı yüklemesi yaklaşımı ile kullanıcı tanımlı tipler için de tanımlanabilir. Buradaki temel amaç yazdirRasyonel fonksiyonunu çağırmak yerine cout<<r;ifadesi ile bir rasyonel sayıyı ekrana yazdırmaktır.

İşleç aşırı yükleme için tanımlanacak fonksiyonun ismi operator anahtar kelimesi ve işleç karakterinden oluşur. Prograram12.5.cpp'de operator<< ve operator+ fonksiyonları işleç aşırı yükleme için tanımlanmışlardır. operator+ fonksiyonunun tanımı toplaRasyonel fonksiyonunun tanımına benzerdir. Bu fonksiyon '+' işlecinin sol tarafındaki rasyonel sayı tarafından çağrılır. İşlecin sağ tarafındaki rasyonel sayı ise operator+ fonksiyonuna parametre olarak gönderilir.

operator<< fonksiyonu friend olarak tanımlanmış bir işleç aşırı yükleme fonksiyonudur. friend fonksiyonlar bir sınıfın üye fonksiyonu olmamakla beraber, sınıfın public üyelerinin yanında private üyelerine de erişme yetkisine sahip fonksiyonlardır. cout<<r ifadesinde '<<' işlecin sol tarafında bir ostream sınıf nesnesi olduğu için bu fonksiyon doğrudan üye fonksiyon olarak tanımlanamaz. İşleç aşırı yükleme için tanımlanan fonksiyonlar işlecin sol tarafındaki nesne ile çağrıldığı için, cout<<r ifadesinin üye fonksiyon olması mümkün değildir. Bu fonksiyon iki parametre alır. Birinci parametre işlecin sol tarafındaki ostream nesnesi, ikinci parametre işlecin sağ tarafındaki rasyonel sayı nesnesidir. Aynı zaman da bu fonksiyonun doğrudan sınıfın private üyelerine de erişmesi gerekir. Bu nedenle ilgili işleç aşırı yükleme fonksiyonu friend olarak tanımlanmıştır. Bu fonksiyonun dönüş tipi ostream nesnesidir. Bu şekilde birden fazla nesne aynı cout ifadesi içinde yer alabilir. Örneğin cout<<r1<<" "<<r2<<endl; ifadesi ile birden fazla nesne ekrana yazdırılabilir.

```
//Program12.5.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
	friend ostream& operator<<(ostream& o, Rasyonel r);
	public:
		Rasyonel(){
			pay=1;
			payda=2;
		}
		Rasyonel(int i, int j){
			pay=i;
			payda=j;
		}
		void setRasyonel(int a, int b){
			pay=a;
			payda=b;
		}
		Rasyonel operator+(Rasyonel r)
		{
			Rasyonel temp;

			temp.pay=payda*r.pay+r.payda*pay;
			temp.payda=payda*r.payda;

			return temp;
		}
private:
		int pay;
		int payda;
};
ostream& operator<<(ostream& o, Rasyonel r)
{
	o<<r.pay<<"/"<<r.payda;

	return o;
}
int main()
{
	Rasyonel r1;
	Rasyonel r2(4,7);
```

```
        Rasyonel r3;
        r3=r1+r2;
        cout<<r1<<endl;
        cout<<r2<<endl;
        cout<<r3<<endl;
        return 0;
}
```

Program12.5.cpp Çıktısı

```
1/2
4/7
15/14
```

3. BİLEŞİM (COMPOSİTİON)

Bileşim (composition) bir sınıfın, kullanıcı tanımlı bir başka sınıfın nesnesini veri üyesi olarak içermesi durumudur. Program12.6.cpp'da iki farklı sınıf tanımlanmıştır. Tarih sınıfı gun, ay ve yil veri üyelerine sahip, yapıcı ve yazdırTarih üye fonksiyonları olan bir sınıftır. Personel sınıfı ad, soyad ve doğum-Tarihi veri üyelerine sahip bir sınıf olarak tanımlanmıştır. ad ve soyad üyeleri string sınıfını tipinden tanımlanmışken, dogumTarihi veri üyesi daha önce oluşturulmuş Tarih sınıfının bir nesnesidir. Bu nesneye başlangıç değeri atamak için üye değer atama listesi kullanılır. Bu yapıcı fonksiyonun başlığından hemen sonra ':' karakteri ile başlayan listedir. Üye değer atama listesi aynı zamanda yerleşik veri tiplerinden oluşan veri üyelerine de değer atamak için kullanılabilir. Veri üyesi ismi ve daha sonra parantez içinde ilgili üyenin değeri yazılarak bu işlem gerçekleştirilir.

```
//Program12.6.cpp
#include <iostream>
#include <string>
using namespace std;

class Tarih
{
        public:
                Tarih(int a, int b, int c)
                {
                        gun=a;
                        ay=b;
                        yil=c;
                }
                void yazdirTarih()
                {
                        cout<<gun<<"-"<<ay<<"-"<<yil;
                }
        private:
                int gun;
```

```
        int ay;
        int yil;
};
class Personel
{
    public:
        Personel(string personelAdi, string personelSoyadi, int g, int a, int y):dogumTarihi(g,a,y)
    {
        ad=personelAdi;
        soyad=personelSoyadi;
    }
        void yazdirPersonel()
    {
        cout<<ad<<" "<<soyad<<endl;
        cout<<"Dogum Tarihi: ";
        dogumTarihi.yazdirTarih();
    }
    private:
        string ad;
        string soyad;
        Tarih dogumTarihi;
};

int main()
{
    Personel a1("Ahmet","Kartal",10,2,2000);
    a1.yazdirPersonel();
    cout<<endl;
    return 0;
}
```

Program12.6.cpp Çıktısı

```
Ahmet Kartal
Dogum Tarihi: 10-2-2000
```

4. KALITIM (INHERİTANCE)

Kalıtım kullanarak var olan sınıflardan yeni sınıflar oluşturulabilir. Kalıtım kullanmanın amacı geliştirilmiş olan kodların yeniden kullanılabilmesidir. Var olan sınıfa temel sınıf, temel sınıftan miras yolu ile oluşturulmuş yeni sınıfa türetilmiş sınıf denir. Temel sınıfın tüm üye fonksiyonları ve veri üyeleri aynı zamanda türetilmiş sınıfın da üye fonksiyonu ve veri üyeleri olarak kabul edilir. Türetilmiş sınıfta, temel sınıfın bazı üye fonksiyonları yeniden yazılabilir. Temel sınıfta yer almayan yeni veri üyeleri veya yeni üye fonksiyonları türetilmiş sınıfa eklenebilir. public, private ve protected olmak üzere üç farklı kalıtım yöntemi vardır. private ve protected kalıtım çok kullanılan yöntemler değildir, bu ne-

denle bu bölümde temel olarak public kalıtım anlatılacaktır. Bu kalıtım şeklinde türetilmiş sınıfın nesneleri aynı zamanda temel sınıfın nesneleri olarak değerlendirilir. Ancak temel sınıfın nesneleri aynı zamanda türetilmiş sınıfın nesneleri olarak düşünülemez. Örneğin Araç sınıfından Otomobil sınıfını kalıtım ile oluşturduğumuzu düşünelim. Tüm otomobiller araçtır ancak tüm araçlar otomobil değildir. Bu kalıtım türünde temel sınıfın private olmayan üyelerine türetilmiş sınıfta doğrudan ulaşılabilir. Ancak private üyelere, private olmayan üye fonksiyonları kullanarak ulaşılabilir. Bu durum için ayrı bir çözüm de geliştirilmiştir. Türetilmiş sınıflarda temel sınıfın veri üyelerine ulaşabilmek ve bu üyelerin aynı zamanda sınıfların kapsama alanları dışında ulaşılabilmesini engellemek amacıyla protected erişim seviyesi geliştirilmiştir. protected seviye public ve private seviyeler arasında erişim düzeyi sağlar. Sınıfların dışında protected erişim seviyesindeki üyelere ulaşmak mümkün olmazken türetilmiş sınıflarda temel sınıfın protected verisine ulaşılabilir. Program12.7.cpp'de Personel temel sınıfından AkademikPersonel ve IdariPersonel sınıfları public kalıtım ile oluşturulmuştur. Bu işlem için türetilmiş sınıf tanımlanırken sınıfın isminden sonra ':' karakteri ve public anahtar kelimesi ile temel sınıfın ismi yazılır. AkademikPersonel türetilmiş sınıfı Personel temel sınıfının ad, soyad ve maas veri üyelerine sahiptir. Ek olarak dersSaati veri üyesi AkademikPersonel sınıfında tanımlanmıştır. AkademikPersonel sınıfında hesaplaToplamMaas temel sınıf fonksiyonu yeniden yazılmış toplam maaş hesabına dersSaati veri üyesi kullanılarak ek ders ücreti ilave edilmiştir. yazdirPersonel fonksiyonunda herhangi bir değişiklik yapılmadığı için AkademikPersonel tipinde tanımlanmış nesneler, temel sınıf versiyonunu doğrudan kullanabilirler. IdariPersonel türetilmiş sınıfında ise temel sınıf üye verilerine ek olarak gorev veri üyesi tanımlanmıştır. IdariPersonel sınıfında yazdirPersonel üye fonksiyonu tekrar yazılmış ancak hesaplaToplamMaas fonksiyonu üzerinde bir değişiklik yapılmamıştır. Türetilmiş AkademikPersonel ve IdariPersonel sınıflarında temel sınıfın veri üyelerine doğrudan erişebilmek amacıyla, bu üyelerin erişim düzeyi protected olarak belirlenmiştir. Bu şekilde bu veriler türetilmiş sınıflarda erişilebilirken, sınıfların kapsama alanları dışında erişilemezler.

```
//Program12.7.cpp
#include <iostream>
#include <string>
using namespace std;
class Personel
{
        public:
                Personel(string personelAdi, string personelSoyadi, double m)
        {
                ad=personelAdi;
                soyad=personelSoyadi;
                maas=m;
        }
        void yazdirPersonel()
        {
                cout<<ad<<" "<<soyad<<endl;
        }
        double hesaplaToplamMaas()
```

```
        {
                return maas;
        }
protected:
        string ad;
        string soyad;
        double maas;
};
class AkademikPersonel: public Personel
{
        public:
                AkademikPersonel(string pAdi, string pSoyadi, double m, int dSaati):Personel(pAdi,p-
                Soyadi,m)
        {
                dersSaati=dSaati;
        }
        double hesaplaToplamMaas()
        {
                return maas + dersSaati*40;
        }
private:
        int dersSaati;
};
class IdariPersonel: public Personel
{
        public:
                IdariPersonel(string pAdi, string pSoyadi, double m, string g):Personel(pAdi,pSoyadi,m)
                {
                        gorev=g;
                }
                void yazdirPersonel()
                {
                        cout<<ad<<" "<<soyad<<" "<<gorev<<endl;
                }
        private:
                string gorev;
};
int main()
{
        AkademikPersonel a1("Ahmet","Kartal",10000,75);
        IdariPersonel i1("Ali","Kanarya",8000,"Ogrenci Isleri");
```

```
        a1.yazdirPersonel();
        cout<<"Toplam Maas:"<<a1.hesaplaToplamMaas()<<endl;

        i1.yazdirPersonel();
        cout<<"Toplam Maas:"<<i1.hesaplaToplamMaas()<<endl;

        return 0;
}
```

Program12.7.cpp Çıktısı

```
Ahmet Kartal
Toplam Maas:13000
Ali Kanarya Ogrenci Isleri
Toplam Maas:8000
```

DİKKAT EDELİM

Türetilmiş sınıflarda temel sınıfın private üyelerine doğrudan erişim söz konusu değildir.

5. ÇOK BİÇİMLİLİK (POLYMORPHİSM)

Kalıtım hiyerarşisinde yer olan tüm sınıfların nesnelerine temel sınıf nesnesi gibi davranarak çok biçimlilik oluşturulur. Hiyerarşide yer alan her bir nesne kendi sınıfının görevini gerçekleştirir. Örneğin, Animal temel sınıfını düşünelim. Bu temel sınıftan türetilmiş her bir sınıfın move fonksiyonu vardır. Hiyerarşide yer alan farklı Animal nesneleri bir Animal temel sınıfı işaretçi listesi kullanılarak erişilebilir. Her bir türetilmiş sınıf nesnesinin kendine özgü move fonksiyonunu kullanması beklenecektir. Ancak normal şartlarda bir temel sınıf işaretçisi veya referansı ile türetilmiş sınıf nesnesini gösterdiğimizde, herhangi bir üye fonksiyon çağrıldığında temel sınıf fonksiyonu kullanılır. Ancak burada temel amacımız her bir türetilmiş sınıf nesnesini aynı listede tutarak (işaretçi veya referans kullanarak), nesnelerin kendilerine özgü fonksiyonu kullanabilmelerini sağlamaktır. Bu durum için sanal (virtual) fonksiyonlar geliştirilmiştir. Temel sınıf işaretçisi veya referansı kullanarak sanal fonksiyonlar üzerinden türetilmiş sınıf fonksiyonlarına ulaşarak çok biçimlilik gerçekleştirilir. Program çalıştığında, dinamik olarak nesnenin doğru fonksiyonuna erişim sağlanacaktır. Temel sınıf işaretçisi (veya referansı) ile türetilmiş sınıfın nesnesini gösterdiğimizde, bu işaretçi ile sanal fonksiyon çağrıldığında temel sınıfın değil türetilmiş sınıfın fonksiyonu kullanılır.

Temel sınıftaki sanal fonksiyonlardan herhangi biri saf sanal (pure virtual) olarak tanımlanmışsa bu sınıfa soyut temel sınıf denir. Bir fonksiyonun saf sanal olarak tanımlanması, fonksiyon başlığına atama işleci ile 0 atanması sayesinde gerçekleştirilir. Soyut temel sınıflardan herhangi bir nesne tanımlanması söz konusu değildir. Bu sınıflar diğer sınıfların türetilebilmesi için bir temel olarak kullanılırlar. Program12.8. cpp'de yer alan Sekil sınıfına ait hesaplaAlan fonksiyonu saf sanal olarak tanımlanmış ve bu şekilde Sekil sınıfı soyut temel sınıf olmuştur. Bu programda Daire ve Dikdortgen sınıfları Sekil sınıfından public kalıtım ile türetilmiştir. Sekil sınıfında saf sanal olarak belirlenmiş hesaplaAlan fonk-

siyonu türetilmiş sınıflarda tanımlanmıştır. Daire sınıfını yaricap, Diktortgen sınıfının kisaKenar ve uzunKenar isminde veri üyeleri vardır. Her iki sınıfta da veri üyelerine değer ataması, üye değer atama listesi ile gerçekleştirilmiştir. main fonksiyonda Daire sınıfından bir nesne, Dikdortgen sınıfından dört nesne oluşturulmuştur. Bu nesnelerin adresleri temel sınıf işaretçisi dizisine atanmıştır. Dolayısı ile farklı türetilmiş sınıf nesneleri aynı liste içinde tutulmuştur. Aynı liste üzerinde bir for döngüsü ile temel sınıf işaretçileri kullanarak hesaplaAlan fonksiyonu çağrılmıştır. Bu fonksiyon sanal olarak tanımlandığından, dinamik olarak doğru sınıfın hesaplaAlan fonksiyonu çağrılmış ve kullanılmıştır.

```
//Program12.8.cpp
#include <iostream>
#include <string>
using namespace std;

class Sekil
{
	public:
		virtual double hesaplaAlan() =0;
};
class Daire: public Sekil
{
	public:
		Daire(double r):yaricap(r){}
		double hesaplaAlan(){return 3.14*yaricap*yaricap;}
	private:
		double yaricap;
};
class Dikdortgen: public Sekil
{
	public:
	Dikdortgen(double a=1.0, double
b=2.0):kisaKenar(a),uzunKenar(b){}
	double hesaplaAlan(){return kisa-
Kenar*uzunKenar;}
	private:
		double kisaKenar;
		double uzunKenar;
};
int main()
{
	Daire c1(20);
	Dikdortgen r1(2,7);
	Dikdortgen r2, r3(4);
	Dikdortgen r4(r1);
```

```
        Sekil *arr[]={&c1,&r1,&r2,&r3,&r4};
        for(int i=0; i<5; i++)
                cout<<arr[i]->hesaplaA-
lan()<<endl;
        return 0;
}
```

Program12.8.cpp Çıktısı

```
1256
14
2
8
14
```

DİKKAT EDELİM

Soyut temel sınıftan bir nesne tanımlamak mümkün değildir. Eğer soyut sınıftan nesne tanımlanırsa, derleyici hatası oluşur.

BÖLÜM ÖZETİ

Sınıf kullanarak kullanıcı tanımlı yeni veri tipleri oluşturulabilir. Bir sınıfı oluşturmak için üye fonksiyonlar ve veri üyeleri tanımlanır. Yapıcı ve yıkıcı fonksiyonlar sınıf ismi kullanılarak tanımlanan özel fonksiyonlardır. Yapıcı fonksiyon bir sınıf nesnesi oluşturulduğunda, otomatik olarak çağrılır. Bu fonksiyon içinde veri üyelerine ilk değerleri atanabilir. Bir nesnenin tanımlandığı alan çalışmasını tamamladığında nesne silinir ve otomatik olarak yıkıcı fonksiyon çağrılır. Bir sınıf içerisinde kullanılabilecek public, protected ve private olmak üzere üç farklı erişim seviyesi vardır. private üyeler sadece sınıfın kapsam alanında erişilip kullanılabilirler. public üyeler sınıfın kapsam alanının yanında main fonksiyon ve diğer fonksiyonlar içerisinden ulaşılabilirler. protected olarak tanımlanmış üyeler kalıtım hiyerarşisinde türetilmiş sınıflar içerisinde erişilip kullanılabilirler. Bir sınıfın nesnesinin bir başka sınıf içerisinde veri üyesi olarak kullanılması bileşim olarak isimlendirilir. İşleç aşırı yükleme ile yerleşik veri tipleri üzerinde kullanılan işleçlerin kullanıcı tanımlı veri tipleri üzerinde kullanılabilmesi sağlanabilir. Temel sınıflardan yeni türetilmiş sınıflar oluşturulmasına kalıtım denir. Türetilmiş sınıf temel sınıfın üye fonksiyonlarına ve veri üyelerine sahip olur. Gerektiğinde ek veri üyeleri tanımlayabilir veya var olan temel sınıf fonksiyonlarında değişiklik gerçekleştirebilir. Temel sınıf işaretçileri veya referansları ile türetilmiş sınıf nesnelerini göstererek ve sanal fonksiyonlar üzerinden türetilmiş sınıf fonksiyonlarına dinamik olarak erişerek çok biçimlilik gerçekleştirilir.

GÖZDEN GEÇİRELİM

1. **Aşağıdakilerden hangisi Rasyonel sınıfından r1 isimli nesneyi doğru olarak tanımlar?**
   a. class Rasyonel r1;
   b. Rasyonel r1;
   c. public Rasyonel r1;
   d. R r1;
   e. Rasyonel class r1;

2.
```
class Rasyonel {
        public:
        void yazdirRasyonel();
};
```
Rasyonel sınıfındaki yazdirRasyonel() fonksiyonunun sınıf dışında geliştirilmesi hangi şıkta doğru verilmiştir?
   a. void Rasyonel->yazdirRasyonel(){}
   b. void Rasyonel.yazdirRasyonel(){}
   c. Rasyonel::yazdirRasyonel(){}
   d. void yazdirRasyonel(){}
   e. void Rasyonel::yazdirRasyonel(){}

3. **Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?**

```
#include <iostream>
using namespace std;
class Dikdortgen
{
  public:
      int x;
      int y;
};
int main()
{
      D ikdortgen d1;
      D ikdortgen d2;
      d1.x=5;
      d1.y=10;
      d2.x=1;
      d2.y=2;
      d2=d1;
      d1.x=3;
      d1.y=4;
      cout <<d1.x<<" "<<d2.y<<endl;
      return 0;
}
```
   a. 3 10
   b. 5 10
   c. 1 4
   d. 1 2
   e. 3 4

4. **Aşağıdakilerden hangisinde Personel sınıfı doğru olarak tanımlanmıştır?**
   a. class Personel{string ad;};
   b. public class Personel{string ad;};
   c. private class Personel{string ad;};
   d. protected class Personel{string ad;};
   e. class Personel(string ad;){ };

5. **Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?**

```
#include <iostream>
using namespace std;
class Test {
        public:
            int x;
            Test(){x=10;}
};
int main()
{
        Test t1,t2,t3;
        t1.x=7;t3.x=8;t1.x=5;
        Test dizi[] ={t1,t2,t3};
        cout<<dizi[1].x;
        return 0;
}
```

a. 5
b. 7
c. 8
d. 10
e. Derleme hatası üretir.

6.

```
int main(){
    Personel p("Ali","Ahmet");
}
```

Yukarıda verilen kodun çalışabilmesi için gerekli yapıcı fonksiyon aşağıdakilerden hangisidir?

a. Personel (string a, string s){
ad = a;
soyad = s;}
b. constructor Personel (string a, string s){
ad = a;
soyad = s;}
c. Personel.Personel (string a, string s){
ad = a;
soyad = s;}
d. Personel (string a, string s) public {
ad = a;
soyad = s;}
e. public Personel (string a, string s){
ad = a;
soyad = s;}

7. **ArastirmaGorevlisi sınıfını Personel ve Ogrenci sınıflarından miras alarak oluştu-racak C++kodu aşağıdakilerden hangisidir?**

a. class ArastirmaGorevlisi ->public Personel, public Ogrenci{ };
b. class Personel, class Ogrenci : class ArastirmaGorevlisi { };
c. class Personel, class Ogrenci: class ArastirmaGorevlisi (){ };
d. class ArastirmaGorevlisi: public Personel, public Ogrenci{ };
e. class ArastirmaGorevlisi: : public Personel, public Ogrenci(){ };

8. **Aşağıda verilen ifadelerden hangisi ++ işlecini aşırı yüklemek için kullanılan fonksiyonun ismidir?**

a. operator++
b. ++
c. ++operator
d. overload++
e. ++overload

9. **Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?**

```
#include <iostream>
using namespace std;
class Dikdortgen{
    public:
        int x;
        int y;
        virtual void alan()
{cout<<x*y<<endl;}
    };
class Kare:public Dikdortgen{
    public:
    void alan(){cout<<x*x<<endl;}
};
int main()
{
    Kare k;
Dikdortgen *d=&k;
    d->x =2; d->y =3;
d->alan();
return 0;
}
```

a. Derleyici hata üretir
b. 5
c. 6
d. 9
e. 4

10. **Aşağıdakilerden hangisi hesapla fonksiyonunu saf sanal (pure virtual) olarak tanımlar?**

a. pure virtual void hesapla();
b. virtual void hesapla()=0;
c. void virtual hesapla();
d. void hesapla()=0;
e. virtual void hesapla();

Yanıt Anahtarı: 1-B, 2-E, 3-A, 4-A, 5-D, 6-A, 7-D, 8-A, 9-E, 10-B

Kaynakça

Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.

Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Prog-ramlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.

Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.

Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.

Kaleli, C., Bilge, A., Ergin, S., & Şora Günal, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.