

Sınıflar ve Nesneler

Dr. Metin Özkan

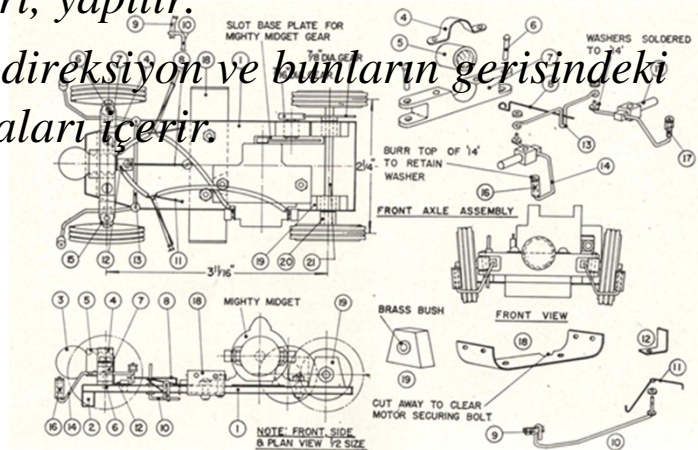
Nesne Tabanlı Programlama Kavramı

- Nesne tabanlı programlama yaklaşımı ile, programlama probleminin çözümü nesnelere paylaştırılmaktadır.
- Böylece, program tasarımı fonksiyonlar ve veri yapıları bağlamında değil, nesneler bağlamında ele alınacaktır.
- Gerçek dünyadaki nesneler ile programdaki nesneler arasında yakın bir eşleşme söz konusudur.
- Her nesnenin özellikleri (member data) ve davranışları (member function) bulunmaktadır. Ortak özellik ve davranış tipleri bulunan nesneler, bir sınıf oluşturur.
- Aşağıdaki resimde iki farklı nesne görülmektedir. Ancak, bunlar aynı sınıfa (araba sınıfı) aittir.



Nesne Tabanlı Programlama Kavramı

- Basit bir analogi yapmak üzere, nesne olarak otomobili ele alalım.
 - *Bir otomobili kullanmak istiyorsunuz.*
 - *Gaz pedalına basarak hızını arttıracaksınız.*
 - *Fren pedalına basarak , hızını azaltacaksınız.*
 - *Direksiyon ile yönlendirme yapacaksınız.*
 - *Siz otomobili kullanmadan önce birilerinin onu tasarlaması gerekir.*
 - *Mühendislik çizimleri, yapılar.*
 - *Çizimler, pedallar , direksiyon ve buntların gerisindeki karmaşık mekanizmaları içerir.*



Nesne Tabanlı Programlama Kavramı

- Pedallar ve direksiyon, harekete geçirdiği karmaşık mekanizmaları otomobil kullanıcısından saklar.
- Otomobil kullanıcısı, motorlar, frenleme ve döndürme gibi karmaşık mekanizmalar hakkında az ya da hiç bilgi sahibi olmaksızın, pedallar ve direksiyon vasıtasıyla kolayca arabayı kullanabilir.
- Elbette otomobil kendisi hareket etmez. Birisinin, hızlandırmak için pedala basması ya da döndürmek için direksiyonu çevirmesi gerekir.
- **Özellikler ve üye veriler (attributes and member data):** Bir otomobil, görevleri yerine getirecek yeteneklerinin olması yanında, rengi, kapı sayısı, deposundaki gaz miktarı, mevcut hızı, katettiği toplam mesafe gibi özelliklere (attributes) sahiptir.

Nesne Tabanlı Programlama Kavramı

- **Üye fonksiyonlar (member functions):** Otomobilde gaz pedalına basılarak bir dizi karmaşık mekanizmanın işlemesi sağlanır ve araç hızlanır. Gaz pedalı, bu karmaşık mekanizmayı saklar; mekanizmayı kolayca işletmek üzere gaz pedalı sağlanır.
 - *Gaz pedalı, üye fonksiyondur. Arkasındaki mekanizma ise fonksiyon kodlamasını oluşturur. Gaz pedalı, otomobilin mevcut hızı ve katettiği toplam mesafe özelliklerinin değerlerinin değişmesini sağlar.*
- **Sınıflar (Classes):** Hız pedalı, fren pedalı, direksiyon ve mekanizmaların mühendislik çizimleri bir sınıf (class) olarak düşünülebilir.
- **Örnekleme, nesne yaratma (Instantiation):** Bir otomobil kullanılmadan önce mühendislik çizimleri kullanılarak üretilmesi gerekir. Bunu, bir sınıftan nesne yaratmak olarak da ifade edebiliriz.

Nesne Tabanlı Programlama Kavramı

- **Örnekleme, nesne yaratma (Instantiation):** Bir otomobil kullanılmadan önce mühendislik çizimleri kullanılarak üretilmesi gerekir. Bunu, bir sınıftan nesne yaratmak olarak da ifade edebiliriz. Bu süreç, **örnekleme (instantiation)** olarak belirtilir. Bir **nesne (object)** ise, sınıfın bir **örneği (instance)** olarak ifade edilir.
- **Mesaj ve üye fonksiyon çağrısı (Message and member function calls):** Bir sürücü otomobili kullanırken, gaz pedalına basarak otomobile gerekli mekanizmayı işletmesi için **mesaj** göndermiş olur. Her **mesaj**, bir **üye fonksiyon çağrısı (member function call)** olarak gerçekleştirilir.
- **Yeniden kullanma (Reuse):** Otomobil mühendislik çizimleri, tekrar tekrar yeni otomobiller üretmek için kullanılabilir. Daha güvenilir ve etkin sistem tasarlamak için de mevcut çizimler kullanılabilir. Sonuçta, bu çizimler bir çok test aşamasını ve performans iyileştirmelerini kapsamaktadır.

Nesne Tabanlı Programlama Kavramı

- **Kapsülleme (Encapsulation):** Sınıflar, özellikleri ve üye fonksiyonları bir nesne içerisinde tutar (**encapsulation**). Bir **nesne (sürücü)**, bir **nesneye (otomobil)** mesaj göndererek haberleşir (**pedala basmak**). Nesneden (sürücü) mesajı alan nesnenin (otomobil) **gerçekleme detayı (mekanizmanın işleyişi)** saklanır. Bu **bilgi saklama (information hiding)**, iyi **yazılım mühendisliği** için çok önemlidir.
- **Ref:** H.M. Deitel and P.J.Deitel, C++ How to Program, Pearson, 9th Edition, 2014.

Sınıflar (Classes)

- Böylece, nesne tabanlı tasarımda (object oriented design), uygulama ortaya çıkmadan önce bileşenlerin (nesneler, objects) belirlenmesi gerekir.
- Bir nesnenin, sahip olacağı veriler ve veriler üzerinde işlem yürütecek operasyonlar, sınıf (class) adı verilen tek bir birim içinde kodlanır.

Sınıflar (Classes)

- Bir sınıf (class), sabit sayıda bileşen topluluğudur.
- Sınıf bileşenleri, sınıfın üyeleri (members) olarak belirtilir.
- Bir sınıfın tanımlanmasında kullanılan genel sentaks şöyledir:

```
class classIdentifier  
{  
    classMemberList;  
};
```

- *Değişken beyanları*
(Variable declarations)
- *Fonksiyonlar*
(Functions)

- C++'da, sınıf (class) ayrılmış bir kelimedir. Bir veri tipi tanımlar. Bir sınıf beyanı (declaration) yapar. Bellekte alan ayrılmaz.

Sınıflar (Classes)

- Nesnel olmayan ve olan iki programın karşılaştırması

```
struct Point{
    int x;
    int y;
};

void print(Point p){
    cout<<p.x<<" "<<p.y;
}

int main(){
    Point p1;
    print(p1);
    return 0;
}
```

```
class Point{
    int x;
    int y;
public:
    void print(){
        cout<<x<<" "<<y;
    }
};

int main(){
    Point p1;
    p1.print();
    return 0;
}
```

Sınıflar (Classes)

- Bir sınıfın (**class**) üyeleri 3 kategoride bulunabilir:

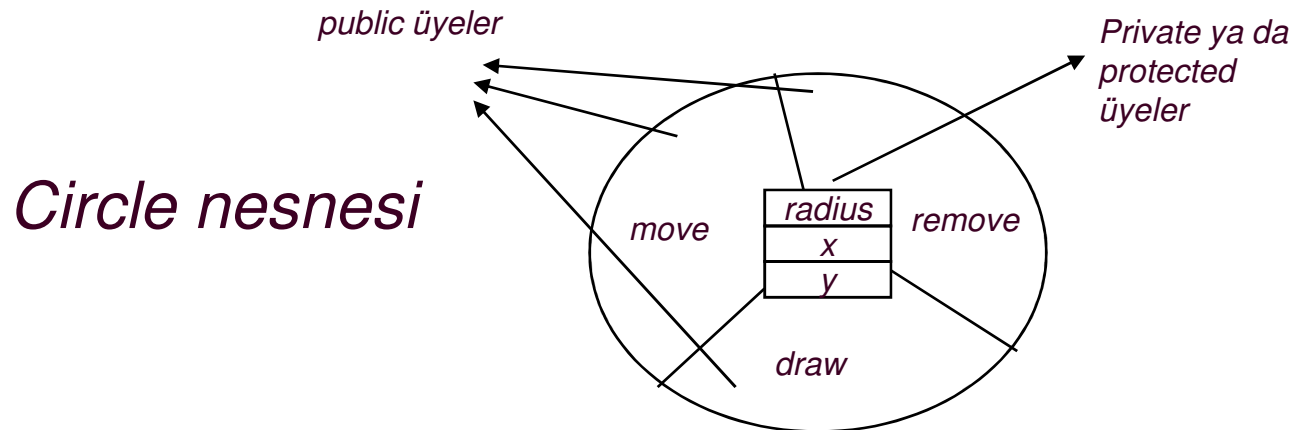
- *Private*

- Varsayılan olarak, bütün sınıf üyeleri private kategorisinde bulunur.
- Sınıf dışından, private olan üyelere erişime izin verilmez.

- *Public*

- public sınıf üyeleri, sınıf dışından erişilebilen üyelere dir.
- Bir sınıf üyesinin public olması için, üyenin beyanı (declaration) öncesinde üye erişim belirteci olarak public kelimesini izleyen : ile kullanılmalıdır.

- *Protected* (İleriki aşamalarda ele alınacaktır. Şu aşamada, private üyeler ile aynı erişim izinlerine sahip olduğu düşünülebilir.)



Örnek Bir Sınıf: Saat (Clock)

- Bir program içinde, gün içi zamanla ilgili işlemleri yürütmek üzere bir sınıf tanımlamak isteniyor. Bu sınıf, `Clock` olarak adlandırılacaktır.
- Gün içi zaman bilgisini bellekte tutmak için 3 değişken ihtiyacı vardır. Bu değişkenler, saat (hour, hr), dakika (minutes, min) ve saniye (seconds, sec) şeklindedir :

```
int hr;           int min;           int sec;
```

- Zaman üzerinde, aşağıdaki işlemlerin yapılabilmesi isteniyor.

<i>Zamanı atama işlemi (Set the time)</i>	<code>setTime</code>
<i>Zamana erişim işlemi (Retrieve the time)</i>	<code>getTime</code>
<i>Zamanı gösterme işlemi (Print the time)</i>	<code>printTime</code>
<i>Saniyeyi bir arttırma işlemi (Increment the time by one second)</i>	<code>incrementSeconds</code>
<i>Dakikayı bir arttırma işlemi (Increment the time by one minute)</i>	<code>incrementMinutes</code>
<i>Saati bir arttırma işlemi (Increment the time by one hour)</i>	<code>incrementHours</code>
<i>İki zamanın eşitlik durumunu karşılaştırma işlemi (Compare the two times for equality)</i>	<code>equalTime</code>

Örnek Bir Sınıf: Saat (Clock)

- Clock sınıfının bazı üyeleri `private`; bazıları ise `public` olacaktır. Karar, üyenin doğasına bağlıdır.
- Aşağıdaki kodlar, Clock sınıfını tanımlamaktadır.

```
class Clock                                     //Declaration of class Clock
{
public:
    void setTime(int, int, int);
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
    bool equalTime (const Clock&) const;

private:
    int hr;
    int min;
    int sec;
};
```

- equaltime (`const Clock&`) `const`

Formal parametre, gerçek parametreyi değiştiremez.

Üye fonksiyon, üye değişkenleri değiştiremez.

Örnek Bir Sınıf: Saat (Clock) - UML

- Unified Modeling Language (UML) Sınıf Diyagramı
 - *Bir sınıf ve üyeleri, grafiksel olarak UML notasyonu olarak tanımlanan bir notasyon ile tanımlanabilir.*
- Diyagram tanımı:
 - *En üst kutucuk: Sınıf adı*
 - *Orta kutucuk: üye değişkenler*
 - *En alt kutucuk: üye fonksiyonlar*
 - *+ (plus): public üyeler*
 - *- (minus): private üyeler*
 - *# (sharp): protected üyeler*

Clock
<pre>-hr: int -min: int -sec: int</pre>
<pre>+setTime(int, int, int): void +getTime(int&, int&, int&) const: void +printTime() const: void +incrementSeconds(): void +incrementMinutes(): void +incrementHours(): void +equalTime(const Clock&) const: bool</pre>

Örnek Bir Sınıf: Saat (Clock) – Nesne Beyanı

- Nesne Beyanı (Object Declaration)

- C++ terminolojisinde, bir sınıf değişkeni ***bir sınıf nesnesi (class object)*** ya da ***bir sınıf örneği (class instance)*** olarak adlandırılır.
- Aşağıdaki kodlar, Clock tipinde iki nesnenin beyanını yapar :

```
Clock myClock;           //an object of type Clock  
Clock yourClock;        //an object of type Clock
```

- Her bir nesne, 10 üyeye sahiptir: 7 üye fonksiyon ve 3 üye değişkendir.
- Her nesne için, değişken değerlerini (hr, min, sec) tutmak üzere ayrı bellek alanı tahsis edilir.
- Bellek alanı sadece üye değişkenler için tahsis edilir. C++ derleyici, üye fonksiyonların aynı kopyasını üretir.
- Her nesne, üye fonksiyonlarının aynı kopyasını çalıştırır.

Örnek Bir Sınıf: Saat (Clock) – Sınıf Üyelerine Erişim

- Sınıf Üyelerine Erişim

➤ *Nokta (.), üyelere erişim için kullanılan bir operatördür (the member access operator)*

```
Clock myClock;           //an object of type Clock
Clock yourClock;         //an object of type Clock

myClock.setTime(5, 2, 30);
myClock.printTime();
yourClock.getTime(x, y, z); //assume x, y, and z are variables of
                             //type int
if (myClock.equalTime(yourClock))
    .
    .
    .
```

```
myClock.hr=0;             //Derleme hatası oluşur.
myClock.min=yourClock.min; //Derleme hatası oluşur.
```

Dahili Operatörler ve Sınıflar

- C++ diline dahili operatörlerin çoğu, sınıflara uygulanmaz.
 - Örneğin, aritmetik operatörler (+, -, *, v.b.) nesneler üzerinde aritmetik işlemler yaptırmak için kullanılmaz.
 - Ancak, iki dahili operatör kullanılabilir:
 - Üye erişimi, Member access (.)
 - Atama, Assignment (=)
 - Aşağıdaki kod ile

```
myClock=yourClock;           //assignment
```

yourClock nesnesinin üye değişkenlerinin değerleri, myClock nesnesinin üye değişkenlerine kopyalanır.

Örnek Bir Sınıf: Saat (Clock) – Üye fonksiyonların icrası (Implementation Member Functions)

- Clock sınıfı tanımlanırken, üye fonksiyonlar fonksiyon prototipleri olarak verilmektedir.
- Bu üye fonksiyonların icrası sırasında, kapsam operatörü (the scope resolution operator ::) ile birlikte fonksiyonun ait olduğu sınıfın belirtilmesi gerekmektedir.

```
void Clock::setTime(int hours, int minutes, int seconds)
{
    if (0 <= hours && hours < 24)
        hr = hours;
    else
        hr = 0;

    if (0 <= minutes && minutes < 60)
        min = minutes;
    else
        min = 0;

    if (0 <= seconds && seconds < 60)
        sec = seconds;
    else
        sec = 0;
}
```

Örnek Bir Sınıf: Saat (Clock) – Üye fonksiyonların icrası (Implementation Member Functions)

```
void Clock::getTime(int& hours, int& minutes, int& seconds) const
{
    hours = hr;
    minutes = min;
    seconds = sec;
}

void Clock::incrementHours()
{
    hr++;
    if (hr > 23)
        hr=0;
}

void Clock::incrementMinutes()
{
    min++;
    if (min > 59) {
        min=0;
        incrementHours(); //increment hours
    }
}
```

Örnek Bir Sınıf: Saat (Clock) – Üye fonksiyonların icrası (Implementation Member Functions)

```
void Clock::incrementSeconds()
{
    sec++;
    if (sec > 59) {
        sec=0;
        incrementMinutes(); //increment minutes
    }
}

bool Clock::equalTime(const CClock& otherClock) const
{
    return (hr == otherClock.hr
            && min == otherClock.min
            && sec == otherClock.sec);
}

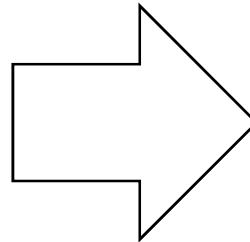
void Clock::printTime() const
{
    if (hr < 10)
        cout << "0";
    cout << hr << ":";
    ...
}
```

Yeniden Kullanılabilirlik için Bir Sınıfı Ayrı Bir Dosyaya Yerleştirme

- Point sınıfının farklı uygulamalarda kullanılabilmesi için, bu sınıfın tanımını .h uzantılı başlık (Header) dosyası olarak kaydetmek gerekir.

pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    void print() {
        cout<<x<<" "<<y;
    }
};
int main(){
    Point p1;
    p1.print();
    return 0;
}
```



Point.h

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    void print() {
        cout<<x<<" "<<y;
    }
};
```

pointApp.cpp

```
#include "Point.h"
int main(){
    Point p1;
    p1.print();
    return 0;
}
```

Arayüzü İcradan Ayırma (Seperating Interface from Implementation)

- Sınıf tanımlamasını, uygulama kodundan ayırmak yeniden kullanılabilirliği sağlamıştır.
- Yazılım Mühendisliği Sorunu
- Ancak, sınıf tanımı ve üye fonksiyon icralarının aynı başlık dosyasında bulunması, yazılım mühendisliği açısından sorun oluşturmaktadır.
- Sınıfın müşterisi (örneğin, main fonksiyonu),
 - *Hangi üye fonksiyonları çağırabileceğini, her bir üye fonksiyon için sağlanması gereken argümanları ve her bir üye fonksiyondan beklenen dönüş değeri tipini bilmeye ihtiyaç duyarken,*
 - *Bu üye fonksiyonların nasıl icra edildiğini (kodlandığını) bilmeye ihtiyaç duymaz.*

Arayüzü İcradan Ayırma

(Seperating Interface from Implementation)

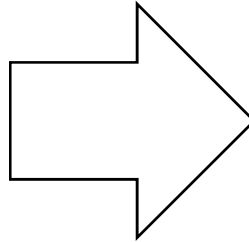
- Bu durumda, sınıfın müşterisinin kullanacağı başlık dosyasında sadece sınıf tanımlamalarının bulunması, üye fonksiyon icralarının bulunmaması beklenir.
- İyi Yazılım Mühendisliği Temel Prensiplerinden Birisi
 - *Arayüzü, icradan ayırma (Seperating Interface from Implementation)*
- Arayüzler, insanlar ve sistemler gibi şeylerin birbirleri ile etkileşim yollarının tanımlanmasını ve standart oluşturulmasını sağlar.
 - *Örneğin, radyo üzerindeki düğmeler kullanıcıya sınırlı işlemler yürütmesini sağlarken, işlemlerin nasıl yürütüldüğünü göstermez. Bu düğmeler birer arayüz oluşturur.*
- Benzer olarak, **bir sınıf arayüzü (interface of a class)**, sınıfın müşterilerinin kullanabileceği servislerin neler olduğunu ve servislerin nasıl isteneceğini tanımlar; ancak, sınıfın servisleri nasıl gerçekleştirdiğini tanımlamaz.

Arayüzü İcradan Ayırma (Seperating Interface from Implementation)

- Bu durumda, sınıf tanımlamaları .h uzantılı başlık dosyasında (header) ve üye fonksiyonların icraları .cpp uzantılı kaynak kod (source-code) dosyasında bulunacaktır.

pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    void print() {
        cout<<x<<" "<<y;
    }
};
int main(){
    Point p1;
    p1.print();
    return 0;
}
```



Point.h

```
class Point{
    int x;
    int y;
public:
    void print();
};
```

Point.cpp

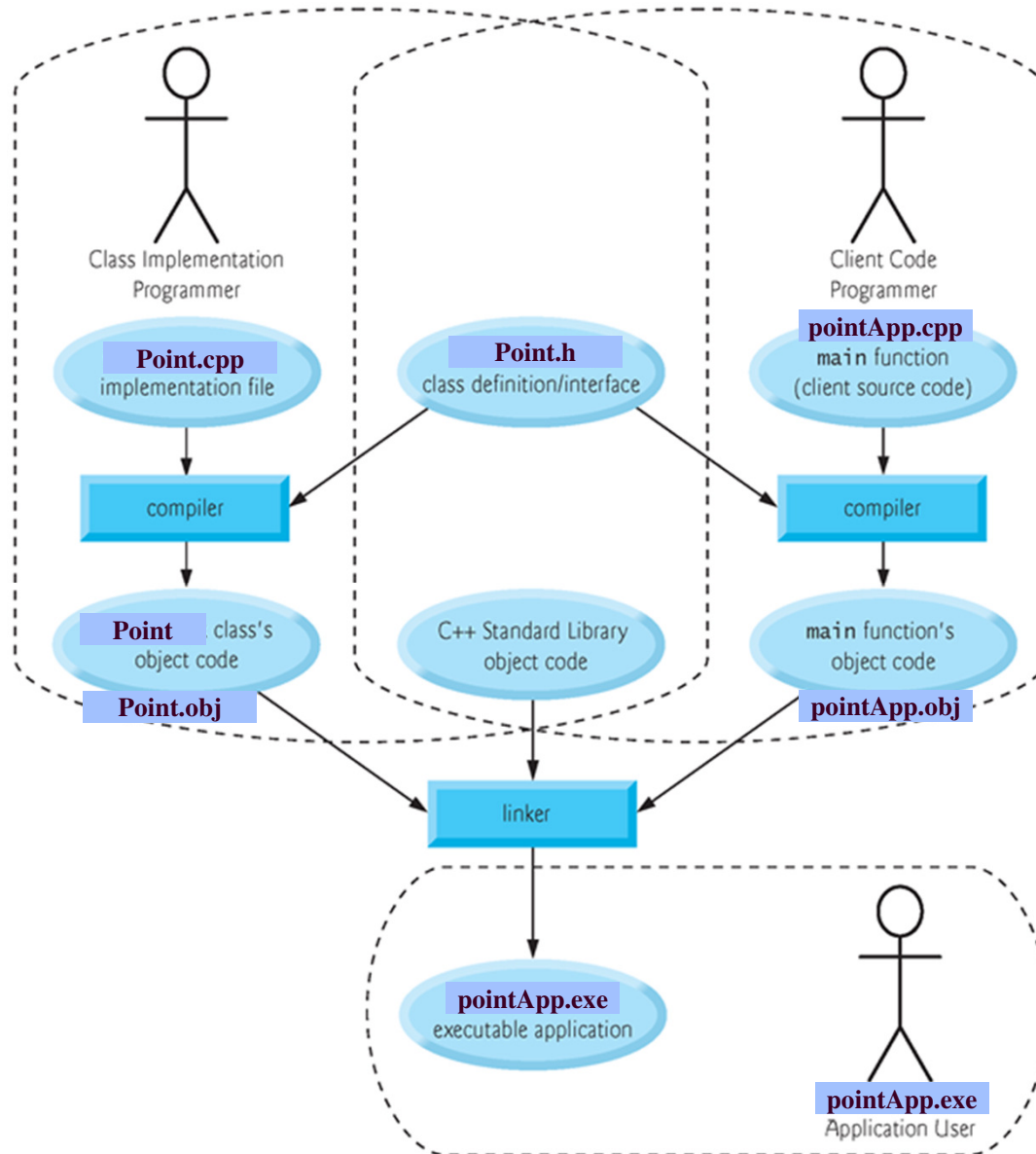
```
#include "Point.h"
#include <iostream>
using namespace std;

void Point::print() {
    cout<<x<<" "<<y;
}
```

pointApp.cpp

```
#include "Point.h"
int main(){
    Point p1;
    p1.print();
    return 0;
}
```

Arayüzü İcradan Ayırma (Seperating Interface from Implementation)



- Bir çalıştırılabilir dosya yaratabilmek için derleme ve link sürecinin gösterimi
- Kaynak: Deitel and Deitel, C++ How to Program, 9th edition.

Erişimci ve Değişimci Üye Fonksiyonlar (Accessor and Mutator Functions)

- **Erişimci Fonksiyon (Accessor function):** Sınıfa ait üye değişkenlere sadece erişim söz konusu olan (değişim gerçekleştirmeyen) üye fonksiyonlar böyle adlandırılır.
 - Örneğin, *equalTime*, *getTime* ve *printTime* fonksiyonları bu tür fonksiyonlardır.
 - Bu tür fonksiyonların başlığı sonunda, *const* kelimesinin yazılması iyi yazılım mühendisliği için gereklidir.
- `void printTime() const;` şeklinde fonksiyon yazıldığında, bu fonksiyon icrası sırasında sınıfa ait değişkenlerin değeri değiştirilmek istenirse derleme hatası oluşur.
- **Değişimci Fonksiyon (Mutator function):** Sınıfa ait üye değişkenlerin değeri değiştirilen üye fonksiyonlardır.
 - *setTime*, *incrementHours*, *incrementMinutes*, ve *incrementSeconds* fonksiyonları bu tür fonksiyonlardır.

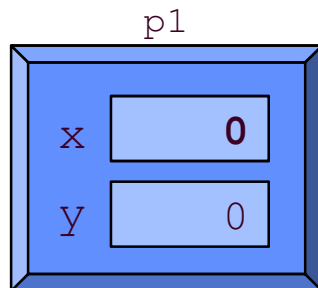
Satırıçi Üye Fonksiyonlar (Inline Function)

- Üye fonksiyonların icrası, sınıf tanımını içerisinde verilebilir.
- Ancak, uzun kodlamalar içeren üye fonksiyonlar için bu tercih edilmemelidir.

```
class Clock{                                     //Declaration of class Cclock
public:
    void setTime(int hours, int minutes, int seconds)
        {hr=hours;
         min=minutes;
         sec=seconds}                             //inline function
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
    bool equaltime (const Cclock&) const;
private:
    int hr;
    int min;
    int sec;
};
```

Yapıcı Fonksiyonlar (Constructors)

- Eğer, kullanıcı bir nesnenin üye değişkenlerine başlangıç değeri atamayı unutursa, program hatalı sonuçlar üretebilir.
- Bir sınıfın üye değişkenlerinin başlangıç değeri almasını garanti etmek için, yapıcı (constructor) fonksiyonlar kullanılabilir.
- Nesne yaratılırken, bu fonksiyona otomatik olarak çağrı gerçekleşir.



pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    Point() {
        x=0;
        y=0;
    }
    void print() {
        cout<<x<<" "<<y;
    }
};

int main() {
    Point p1;
    p1.print();
    return 0;
}
```

Yapıcı Fonksiyonlar (Constructors)

- Aşağıdaki özelliklere sahiptir:
 - *Fonksiyon ismi, sınıf ismi ile aynıdır.*
 - `Point()`
 - *Bir değer döndürmez ve dönüş tipi tanımlanmaz.*
 - `__X__ Point()`
 - *Bir sınıf, birden çok yapıcı fonksiyon içerebilir. Hepsi aynı isimde olmalıdır.*
 - *Birden çok yapıcı fonksiyon varsa, her biri farklı sayıda ya da tipte parametre içermelidir*
 - `Point();`
 - `Point(int _x, int _y);`

pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    Point() {
        x=0;
        y=0;
    }
    Point(int _x,int _y){
        x=_x;
        y=_y;
    }
    void print() {
        cout<<x<<" "<<y;
    }
};
int main() {
    Point p1;
    p1.print();
    return 0;
}
```


Yapıcı Fonksiyonlar (Constructors)

- Aşağıdaki özelliklere sahiptir:
 - *Bir sınıftan nesne yaratılırken, bu fonksiyon otomatik olarak çalışır.*
 - *Hangi yapıcı (constructor) fonksiyona çağrı gerçekleşeceği nesne yaratılırken, verilen parametrelere bağlıdır.*
 - `Point p1;` -> `Point();`
 - `Point p2(1,2);` -> `Point(int _x,int _y);`
 - *İki tip vardır: parametrelili ve parametresiz (with and without parameters)*
 - `Point()` : parametresiz
 - `Point(int _x,int _y)`: parametrelili (`_x` ve `_y`)

pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    Point() {
        x=0;
        y=0;
    }
    Point(int _x,int _y){
        x=_x;
        y=_y;
    }
    void print() {
        cout<<x<<" "<<y;
    }
};
int main() {
    Point p1;
    Point p2(1,2);
    p1.print();
    return 0;
}
```

Yapıcı Fonksiyonlar (Constructors)

- Aşağıdaki özelliklere sahiptir:
 - *Parametre gerektirmeyen yapıcı fonksiyon, varsayılan yapıcı fonksiyon (the default constructor) olarak adlandırılır.*
 - **Point(): varsayılan**
 - *Varsayılan (default) yapıcı fonksiyon tek bir tane olmalıdır.*

pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    Point() {
        x=0;
        y=0;
    }
    Point(int _x,int _y){
        x=_x;
        y=_y;
    }
    void print() {
        cout<<x<<" "<<y;
    }
};
int main(){
    Point p1;
    Point p2(1,2);
    p1.print();
    return 0;
}
```

Yapıcı Fonksiyonlar (Constructors) – Clock Sınıfı

- Clock sınıfının iki yapıcı fonksiyon içeren tanımı

```
class CClock                                     //Declaration of class CClock
{
public:
    void setTime(int, int, int);
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
    bool equaltime (const CClock&) const
    Clock(int, int, int);                          //constructor with parameters
    Clock();                                         //default constructor
private:
    int hr;
    int min;
    int sec;
};
```

Yapıcı Fonksiyonlar (Constructors) – Clock Sınıfı

- Yapıcı fonksiyonların icrası (implementation)

```
Clock::Clock(int hours, int minutes, int seconds)
{
    if (0 <= hours && hours < 24)
        hr = hours;
    else
        hr = 0;
    if (0 <= minutes && minutes < 60)
        min = minutes;
    else
        min = 0;
    if (0 <= seconds && seconds < 60)
        sec = seconds;
    else
        sec = 0;
}

OR

Clock::Clock(int hours, int minutes, int seconds)
{
    setTime(hours, minutes, seconds);
}

Clock::Clock() //default constructor
{
    hr=0;
    min=0;
    sec=0;
}
```

Yapıcı Fonksiyonlar (Constructors) – Clock Sınıfı

- Bir yapıcı fonksiyon çağrısı
 - *Nesne yaratılırken otomatik olarak çağrılır.*
- Bir varsayılan yapıcı fonksiyon çağrısı (The Default Constructor)

```
className classObjectName;
```

➤ *Örneğin*

```
Clock yourClock;  
Clock yourClock(); //Geçersiz beyan(declaration)
```

- Bir yapıcı fonksiyonun parametre ile çağrısı

```
className classObjectName(argument1, argument2, . . .);
```

➤ *Örneğin*

```
Clock myClock(5, 12, 40);
```

Yapıcı Fonksiyonlar (Constructors) – Clock Sınıfı

- Yapıcı Fonksiyon (Constructor) ve varsayılan parametreler (Default Parameters)
 - *Bir yapıcı fonksiyon, varsayılan parametreye sahip olabilir.*
 - *Parametresiz ya da tüm parametreler için varsayılan değere sahip yapıcı fonksiyon, varsayılan yapıcı fonksiyondur.*

```
class Clock                                //Declaration of class Cclock
{
public:
    Clock(int =0, int =0, int =0);         //constructor

private:
    int hr;
    int min;
    int sec;
};

Clock clock1;                             // initialization: hr=0, min=0, sec=0
Clock clock2(5);                           // initialization: hr=5, min=0, sec=0
Clock clock3(12, 30); // initialization: hr=12, min=30, sec=0
Clock clock4(7, 34, 18); // initialization: hr=7, min=34, sec=18
```

Yapıcı Fonksiyonlar (Constructors) – Clock Sınıfı

- Bir önlem

- *Eğer bir sınıfta, yapıcı fonksiyon tanımlanmadı ise, C++ otomatik olarak bir varsayılan yapıcı fonksiyon sağlar.*
- *Bir sınıf düşünün: Parametrelili yapıcı fonksiyon (lar) bulunuyor; ancak, varsayılan bir yapıcı fonksiyon bulunmuyor.*
 - *Bu durumda, nesne yaratılırken gereken parametrelerin verilmesi zorunludur.*
 - *Örneğin;*

Clock sınıfı içerisinde sadece

```
Clock(int, int, int);
```

şeklinde tek bir yapıcı fonksiyon tanımlanmış olsun.

```
Clock clock1;           // Beyanı derleme hatası verir.
```

```
Clock clock1(1,2,3); // Beyanı, hata vermez.
```


Yıkıcı Fonksiyonlar (Destructors)

- Yıkıcı fonksiyonların özellikleri
 - Yapıcı fonksiyonlar gibi, bunlarda fonksiyondur.
 - Bu fonksiyon, sınıf ismi ile aynıdır, ancak ismin başında Tilda karakteri (~) bulunur.
 - `~Point();`
 - Fonksiyon değer döndürmez ve parametre almaz.
 - `__X__ ~Point();`

pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    ~Point() {

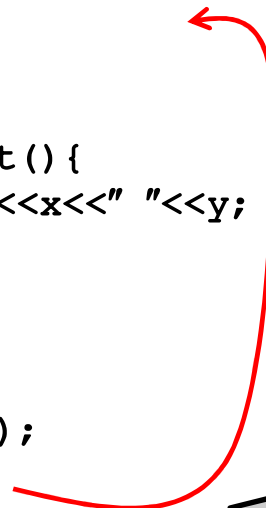
    }
    void print() {
        cout<<x<<" "<<y;
    }
};
int main() {
    Point p1;
    p1.print();
    return 0;
}
```

Yıkıcı Fonksiyonlar (Destructors)

- Yıkıcı fonksiyonların özellikleri
 - *Bir sınıf, sadece bir yıkıcı fonksiyona sahiptir.*
 - *Yıkıcı fonksiyon, nesne yok olurken otomatik olarak çağrılır. Ya da başka bir ifade ile, nesnenin kapsamı (scope) dışına çıkarken çağrılır.*

pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
    int x;
    int y;
public:
    ~Point() {
    }
    void print() {
        cout<<x<<" "<<y;
    }
};
int main() {
    Point p1;
    p1.print();
    return 0;
}
```



Kaynaklar

- T.C. Lethbridge and R. Laganier, Object-Oriented Software Engineering - Practical software development using UML and Java, McGraw Hill, Second Edition, 2005.
- H.M.Deitel and P.J.Deitel, C++ How To Program, 9E, Pearson Press, 2014.
- B. Stroustrup, The C++ Programming Language, 3rd Edition, Special Edition, Addison Wesley, 2000.
- Dr. Feza Buzluca, Ders Notları.
- Ç. Turhan ve F.C. Serçe, C++ Dersi: Nesne Tabanlı Programlama, 2nci Baskı, 2014.