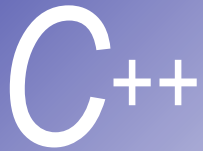


C++

Ders 2

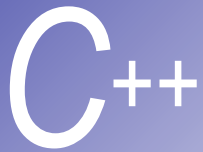
Constructor, Destructor' lar ve
Nesne Yönelimli Programlama
Öğeleri ile Nesne Erişim
Kontrolü



Constructor - Destructor

Fonksiyonların Çağırılması

- **Constructor ve Destructor**
Fonksiyonlar nesne oluşturulurken ve yok edilirken otomatik olarak çağırılır. Bu sebeple çağırıldıkları zamanı ömürleri (scope) belirler.
- **‘Global Scope’** : tüm fonksiyonlardan (main dahil) önce constructor, main fonksiyonundan çıkarken destructor çağırılır.



Constructor - Destructor Fonksiyonların Çağırılması

- **‘Automatic Local Scope’** : Tanımlandıkları yerde constructor, tanımlı oldukları blok sona erince destructor çağırılır.
- **‘Static Local Scope’** : Tanımlandıkları yerde constructor, **main** fonksiyonundan çıkarken destructor çağırılır.
- Programdan **abort** ile çıkılırsa her üç tür için de destructor çağırılmaz. Programdan **exit** ile çıkılırsa sadece ‘automatic’ tipte destructor çağırılmaz.

C++

Örnek (Constructor)

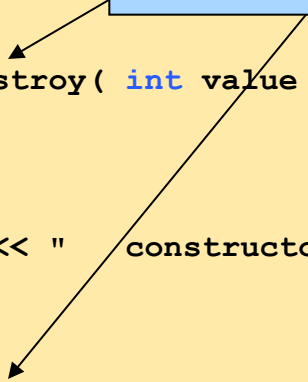
```
1 // Fig. 6.9: create.h
2 // Definition of class CreateAndDestroy.
3 // Member functions defined in create.cpp.
4 #ifndef CREATE_H
5 #define CREATE_H
6
7 class CreateAndDestroy {
8 public:
9     CreateAndDestroy( int ); // constructor
10    ~CreateAndDestroy();      // destructor
11 private:
12    int data;
13 };
14
15 #endif
```

C++

Örnek (Constructor)

```
16 // Fig. 6.9: create.cpp
17 // Member function definitions for class CreateAndDestroy
18 #include <iostream>
19
20 using std::cout;
21 using std::endl;
22
23 #include "create.h"
24
25 CreateAndDestroy::CreateAndDestroy( int value )
26 {
27     data = value;
28     cout << "Object " << data << " constructor";
29 }
30
31 CreateAndDestroy::~~CreateAndDestroy()
32 { cout << "Object " << data << " destructor " << endl; }
```

Constructor ve Destructor fonksiyonlar çağırıldıklarında kendi numaralarını yazacaklar.



C++

Örnek (Constructor)

```
33 // Fig. 6.9: fig06 09.cpp
34 // Demonstrating the order in which constructors and
35 // destructors are called.
36 #include <iostream>
37
38 using std::cout;
39 using std::endl;
40
41 #include "create.h"
42
43 void create( void );    // prototype
44
45 CreateAndDestroy first( 1 ); // global object
46
47 int main()
48 {
49     cout << "    (global created before main)" << endl;
50
51     CreateAndDestroy second( 2 );    // local object
52     cout << "    (local automatic in main)" << endl;
```

C++

Örnek (Constructor)

```
54     static CreateAndDestroy third( 3 ); // local object
55     cout << "    (local static in main)" << endl;
56
57     create(); // call function to create objects
58
59     CreateAndDestroy fourth( 4 ); // local object
60     cout << "    (local automatic in main)" << endl;
61     return 0;
62 }
63
64 // Function to create objects
65 void create( void )
66 {
67     CreateAndDestroy fifth( 5 );
68     cout << "    (local automatic in create)" << endl;
69
70     static CreateAndDestroy sixth( 6 );
71     cout << "    (local static in create)" << endl;
```

C++

Örnek (Constructor)

```

72
73   CreateAndDestroy seventh( 7 );
74   cout << "    (local automatic in create)" << endl;
75 }

```

OUTPUT

```

Object 1    constructor    (global created before main)
Object 2    constructor    (local automatic in main)
Object 3    constructor    (local static in main)
Object 5    constructor    (local automatic in create)
Object 6    constructor    (local static in create)
Object 7    constructor    (local automatic in create)
Object 7    destructor
Object 5    destructor
Object 4    constructor    (local automatic in main)
Object 4    destructor
Object 2    destructor
Object 6    destructor
Object 3    destructor
Object 1    destructor

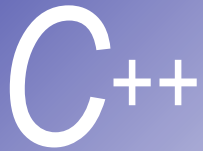
```

Tanımlandıkları tipe göre
(automatic, global and **static**)
çağırılma zamanının nasıl
değiştiğine dikkat edin!

C++

Private Veri' ye Fonksiyonla Erişim

- Sınıf' ı kullanan herkes, okuma ve yazma işleri için public fonksiyonları kullanır.
- Örneğin banka hesabını temsil eden bir nesnemiz olsun. Balance (hesap tutarı) bilgisi (Private tipte bir veri) bir fonksiyon tarafından okunur ve başka bir fonksiyon tarafından değiştirilir.
- Bu fonksiyon verilen sayının geçerli olup olmadığını kontrol ettikten sonra yeni değeri ayarlar.
- Private veriye erişmemize rağmen veri herkese açık değildir ve güvendedir.



Private Veri' ye Referansla Erişim

- Nesneye referansla erişilebilir.
- Referansla erişim sırasında nesneye (aynen orijinalinde yapıldığı gibi) atamalar yapılabilir ve referansla atanan değer orijinalini değiştirir.
- Public fonksiyonlar private verileri değiştirebilme hakkına sahip olduğundan, referansla erişilen public fonksiyon private veriyi değiştirebilir.
- Bu 'encapsulation' zırhını delen tehlikeli bir durumdur.

C++

Örnek (Referansla Erişim)

```
5 // preprocessor directives that
6 // prevent multiple inclusions of header file
7 #ifndef TIME4_H
8 #define TIME4_H
9
10 class Time {
11 public:
12     Time( int = 0, int = 0, int = 0 );
13     void setTime( int, int, int );
14     int getHour();
15     int &badSetHour( int ); // DANGEROUS reference return
16 private:
17     int hour;
18     int minute;
19     int second;
20 };
21
22 #endif
```

badSetHour fonksiyonunun bir referans (adres) döndürdüğüne dikkat edin!

Dönüş tipi **int &** dir.

C++

Örnek (devam)

```
23 // Fig. 6.11: time4.cpp
24 // Member function definitions for Time class.
25 #include "time4.h"
26
27 // Constructor function to initialize private data.
28 // Calls member function setTime to set variables.
29 // Default values are 0 (see class definition).
30 Time::Time( int hr, int min, int sec )
31     { setTime( hr, min, sec ); }
32
33 // Set the values of hour, minute, and second.
34 void Time::setTime( int h, int m, int s )
35 {
36     hour    = ( h >= 0 && h < 24 ) ? h : 0;
37     minute  = ( m >= 0 && m < 60 ) ? m : 0;
38     second  = ( s >= 0 && s < 60 ) ? s : 0;
39 }
```

C++

Örnek (devam)

```
40
41 // Get the hour value
42 int Time::getHour() { return hour; }
43
44 // POOR PROGRAMMING PRACTICE:
45 // Returning a reference to a private data member.
46 int &Time::badSetHour( int hh )
47 {
48     hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
49
50     return hour; // DANGEROUS reference return
51 }
```

badSetHour geri dönüş değeri olarak **private** bir değişken olan **hour** 'un adresini döndürüyor.

Bu değeri kullanarak daha sonra kolaylıkla dışarıdan **hour** 'un değeri değiştirilecek.

C++

Örnek (devam)

```

63 int main()
64 {
65     Time t;
66     int &hourRef = t.badSetHour( 20 );
67
68     cout << "Hour before modification: " << hourRef;
69     hourRef = 30; // modification with invalid value
70     cout << "\nHour after modification: " << t.getHour();
71
72     // Dangerous: Function call that returns
73     // a reference can be used as an lvalue!
74     t.badSetHour(12) = 74;
75     cout << "\n\n*****\n"
76         << "POOR PROGRAMMING PRACTICE!!!!!!\n"
77         << "badSetHour as an lvalue, Hour: "
78         << t.getHour()
79         << "\n*****" << endl;
80
81     return 0;
82 }

```

Time tipindeki **t** nesnesinde **t.badSetHour(20)** fonksiyonunun geridöndürdüğü adresi **hourRef** değişkeni aracılığı ile kullanacağız.

Referans sayesinde saçma değerler (30 ve 74) atayabildik!

C++

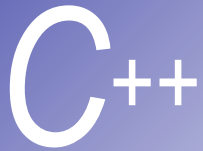
Örnek (Ekran Çıktısı)

```
Hour before modification: 20
Hour after modification: 30

*****
POOR PROGRAMMING PRACTICE!!!!!!!
badSetHour as an lvalue, Hour: 74
*****
```

HourRef sayesinde **hour** değişkenine saçma değerler atanabildi.

Normalde **setbadSetHour** fonksiyonu verilen değerleri kontrol ederek böyle saçma değerlerin girilmesine engel oluyordu!



Nesnelerde Atama

- Aynı tipte (sınıf yapısında) iki nesne söz konusu olduğunda, bir nesne diğer nesneye '=' operatörü ile atanabilir.
- Bu atama ile tüm üyelerin değerleri bir nesneden diğerine kopyalanmış olur.

C++

Örnek (Atama)

```
9 // Simple Date class
10 class Date {
11 public:
12     Date( int = 1, int = 1, int = 1990 ); // default constructor
13     void print();
14 private:
15     int month;
16     int day;
17     int year;
18 };
19
20 // Simple Date constructor with no range checking
21 Date::Date( int m, int d, int y )
22 {
23     month = m;
24     day = d;
25     year = y;
26 }
27
28 // Print the Date in the form mm-dd-yyyy
29 void Date::print()
30 { cout << month << '-' << day << '-' << year; }
```

C++

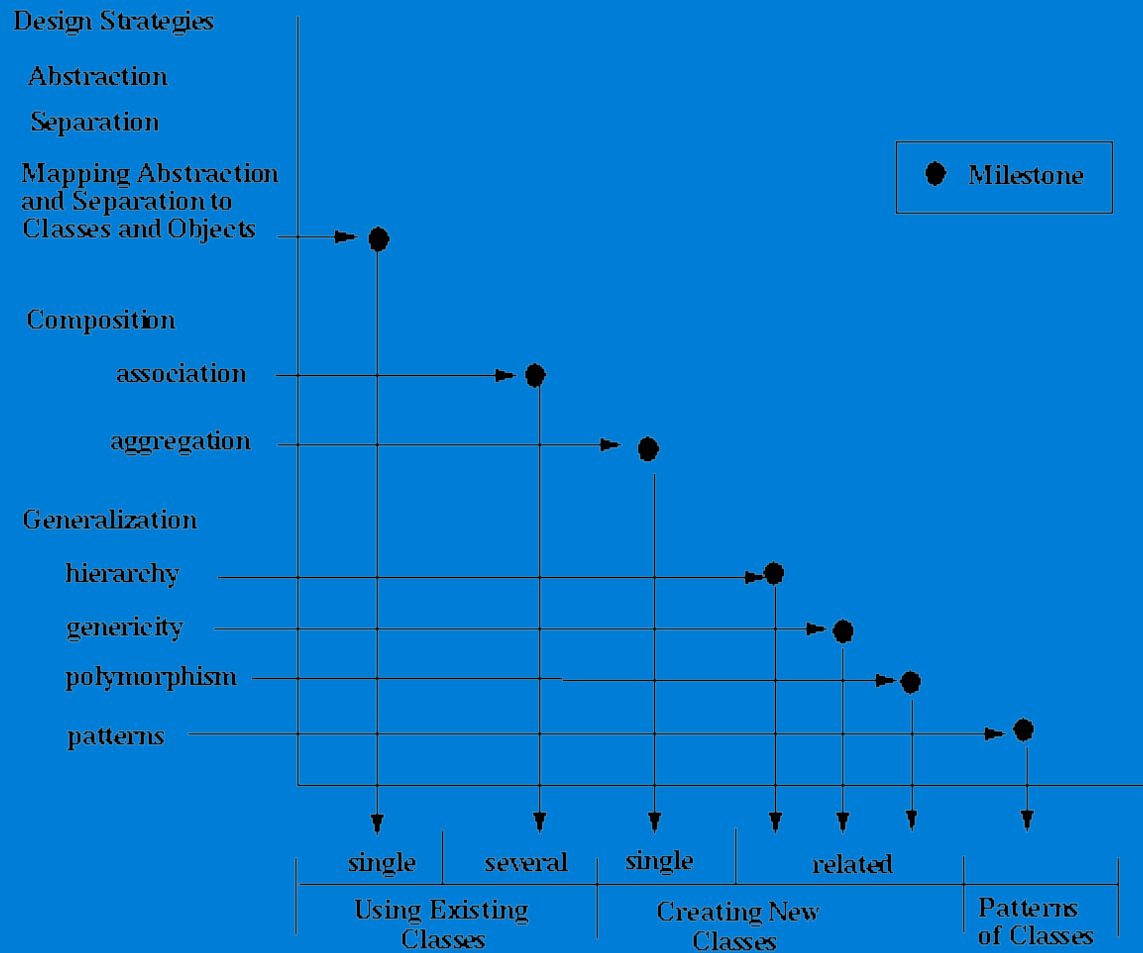
Örnek (devam)

```
32 int main()
33 {
34     Date date1( 7, 4, 1993 ), date2; // d2 defaults to 1/1/90
35
36     cout << "date1 = ";
37     date1.print();
38     cout << "\ndate2 = ";
39     date2.print();
40
41     date2 = date1; // assignment by default memberwise copy
42     cout << "\n\nAfter default memberwise copy, date2 = ";
43     date2.print();
44     cout << endl;
45
46     return 0;
47 }
```

```
date1 = 7-4-1993
date2 = 1-1-1990
```

```
After default memberwise copy, date2 = 7-4-1993
```

Rollerin Gelişimi



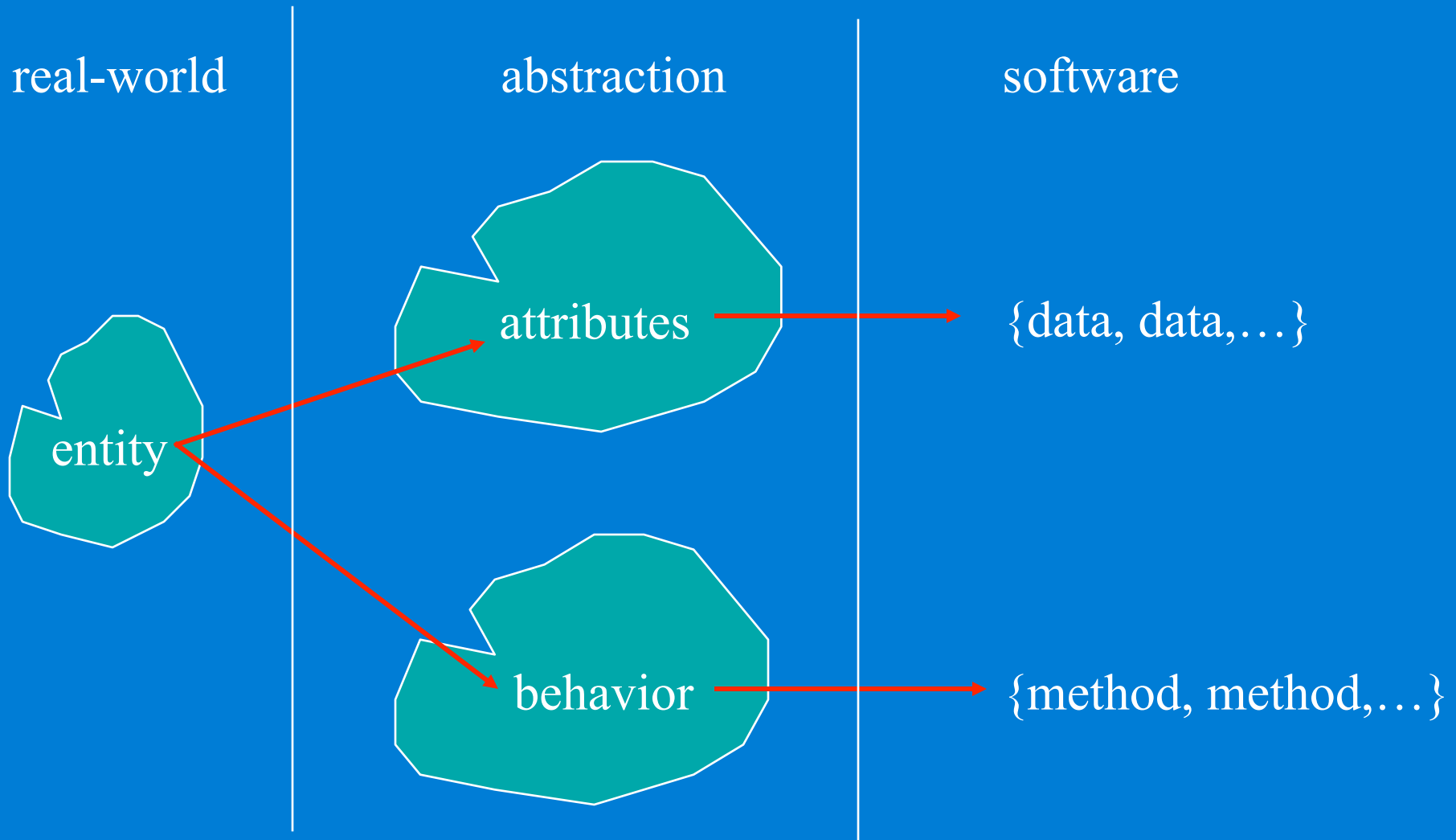
Soyutlama (Abstraction)

Belirli bir amaç için bir varlığın modellenmesi ile ilişkili verilerin ve fonksiyonların toplamına verilen isimdir

İyi bir Soyutlama (Abstraction) nın Özellikleri

- İyi İsimlendirilmiş
- Geçirgen
- Doğru
- minimal
- bütünleşik

Abstraction Kavramının Yazılıma aktarılması

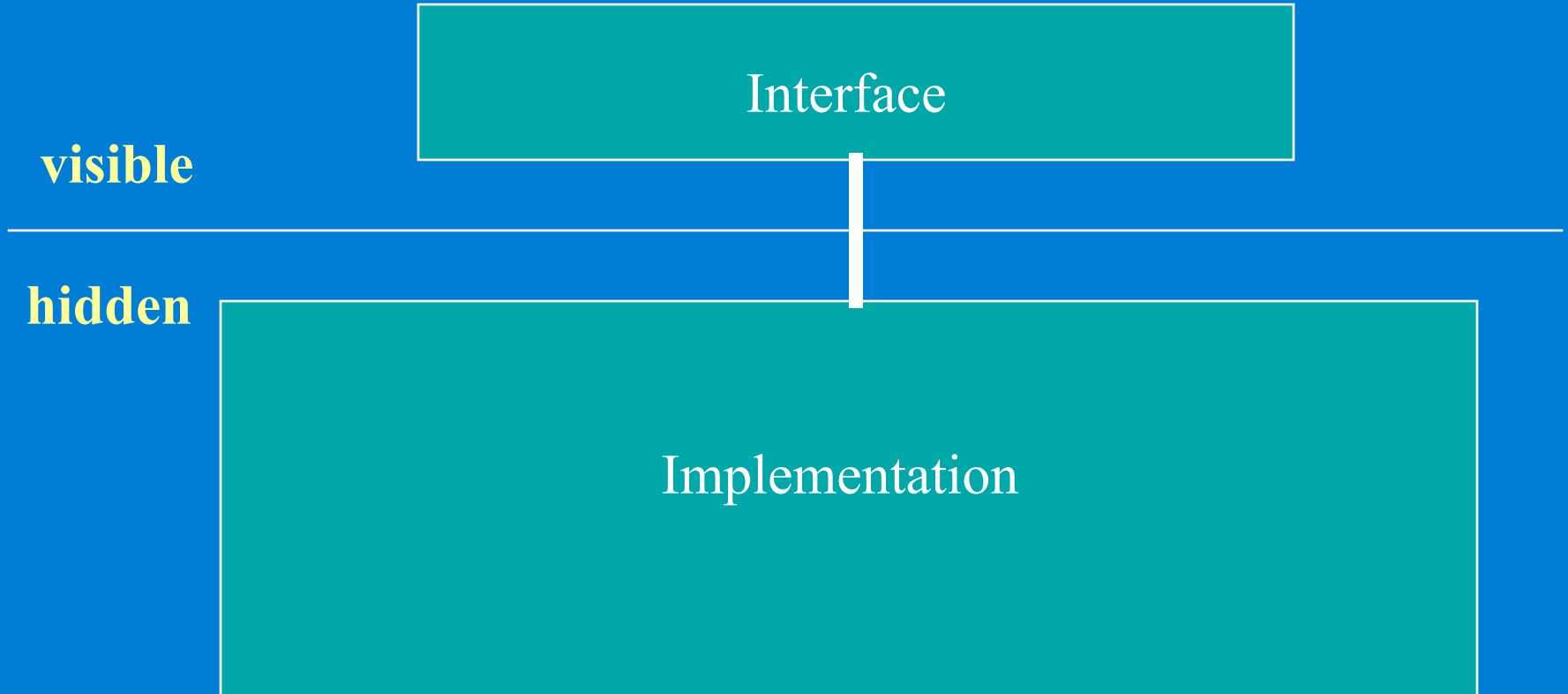


-
-
-

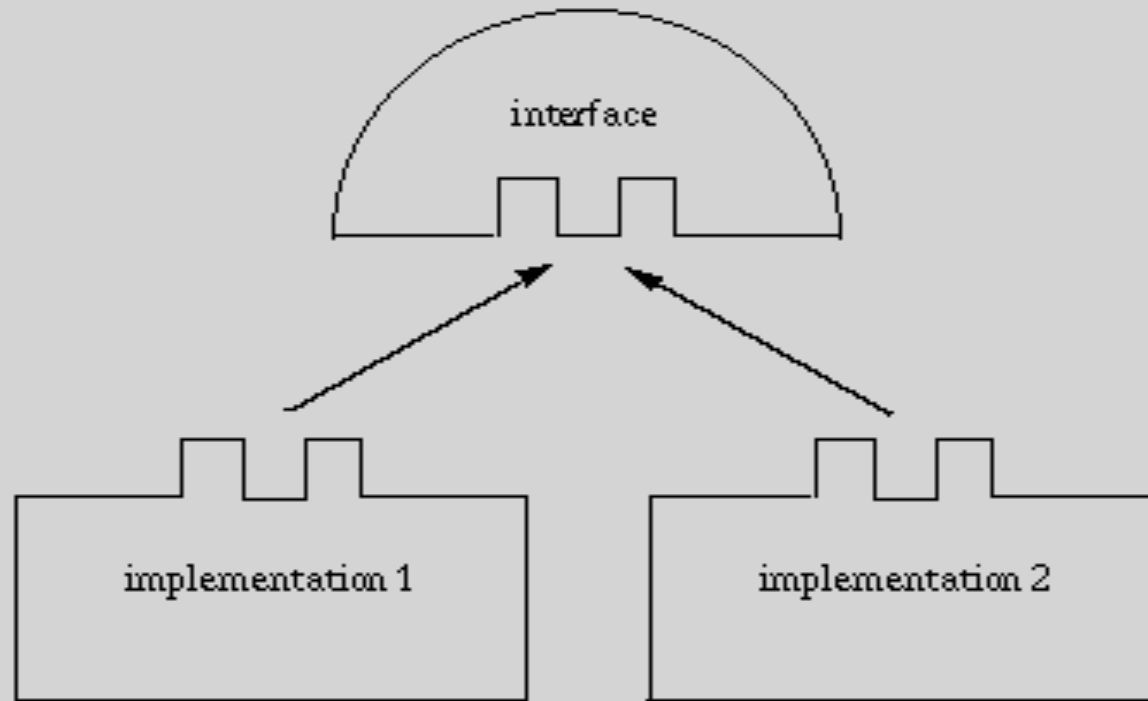
Ayrıştırma-Separation

Nesne Yönelimli Programlamada,
arayüzün bağımsız tanımlanması ve bir
veya birden fazla bu arayüzün
uygulanmasıdır.

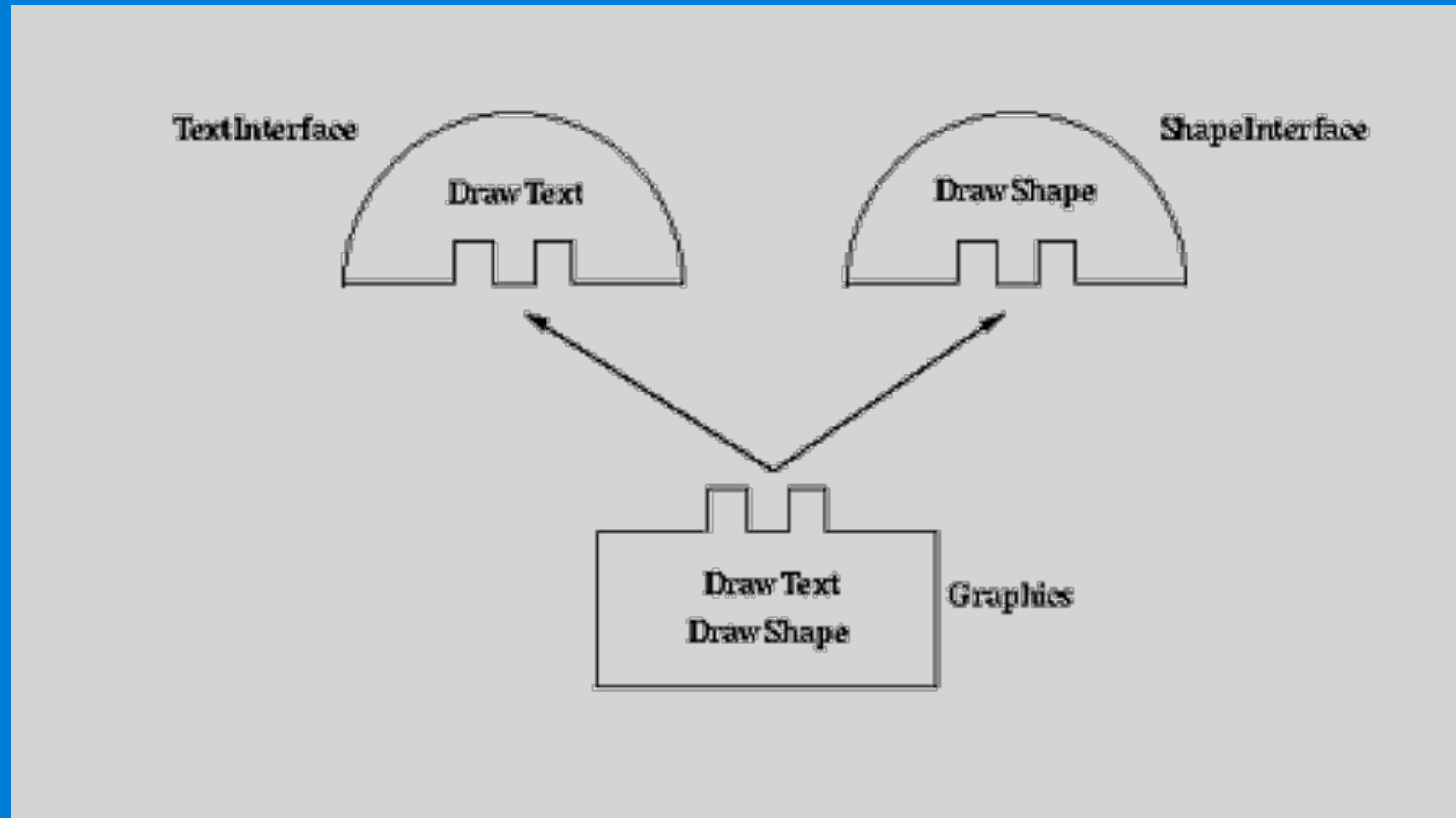
Ayrıştırma-Separation



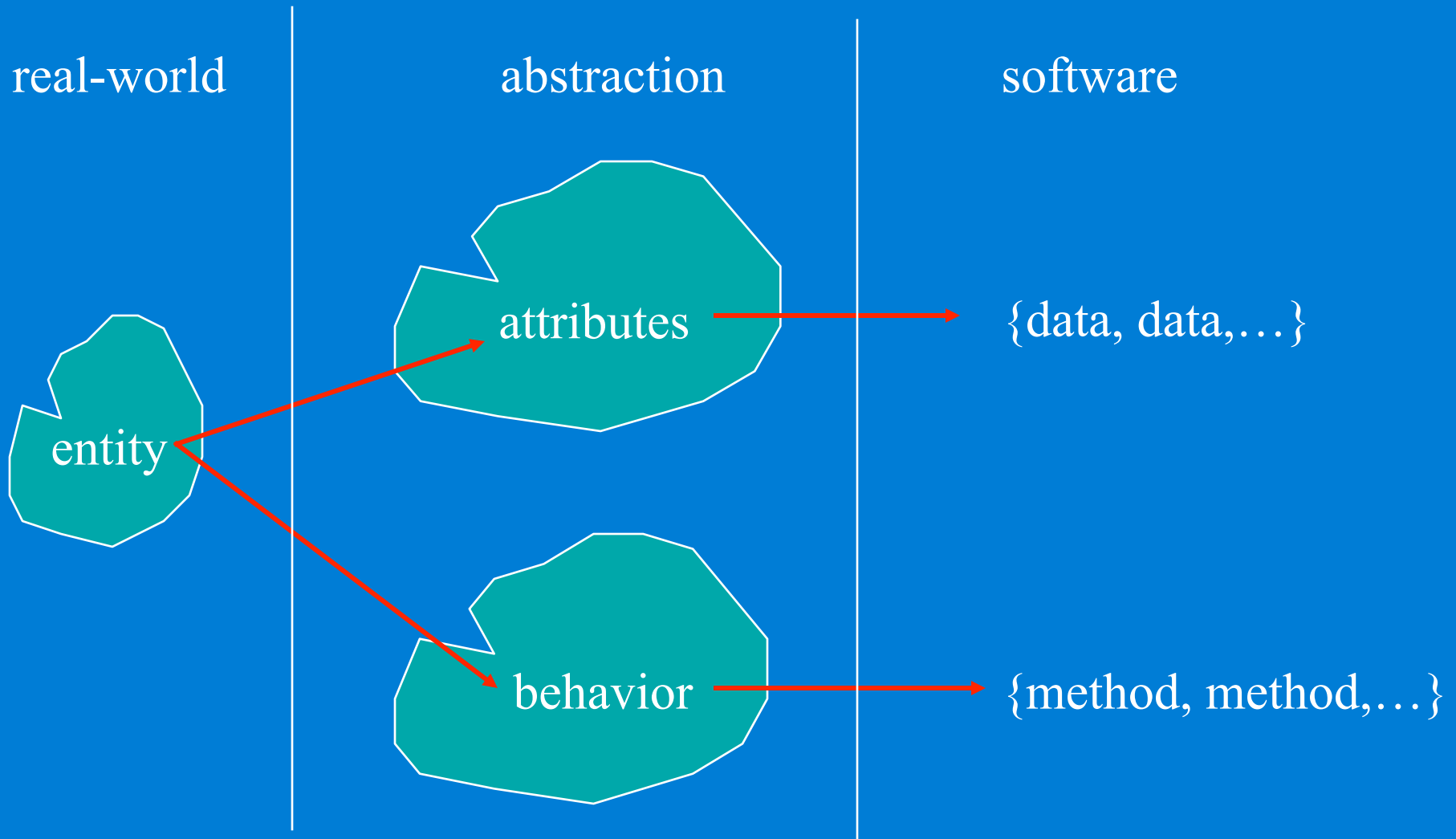
Uygulamadan Arayüzün Ayırıştırılması



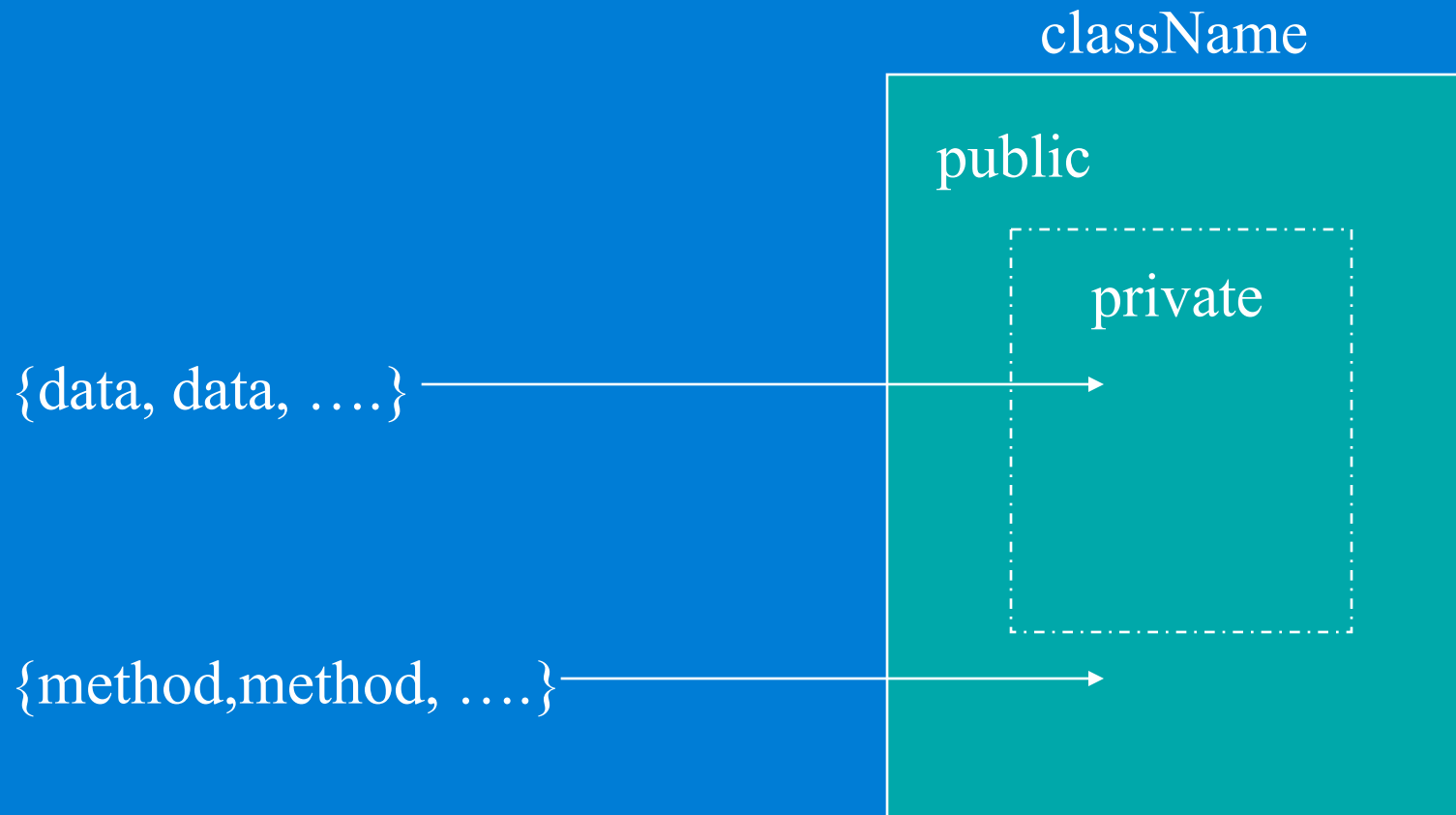
Uygulamaların Yerdeğiştirilebilirliği



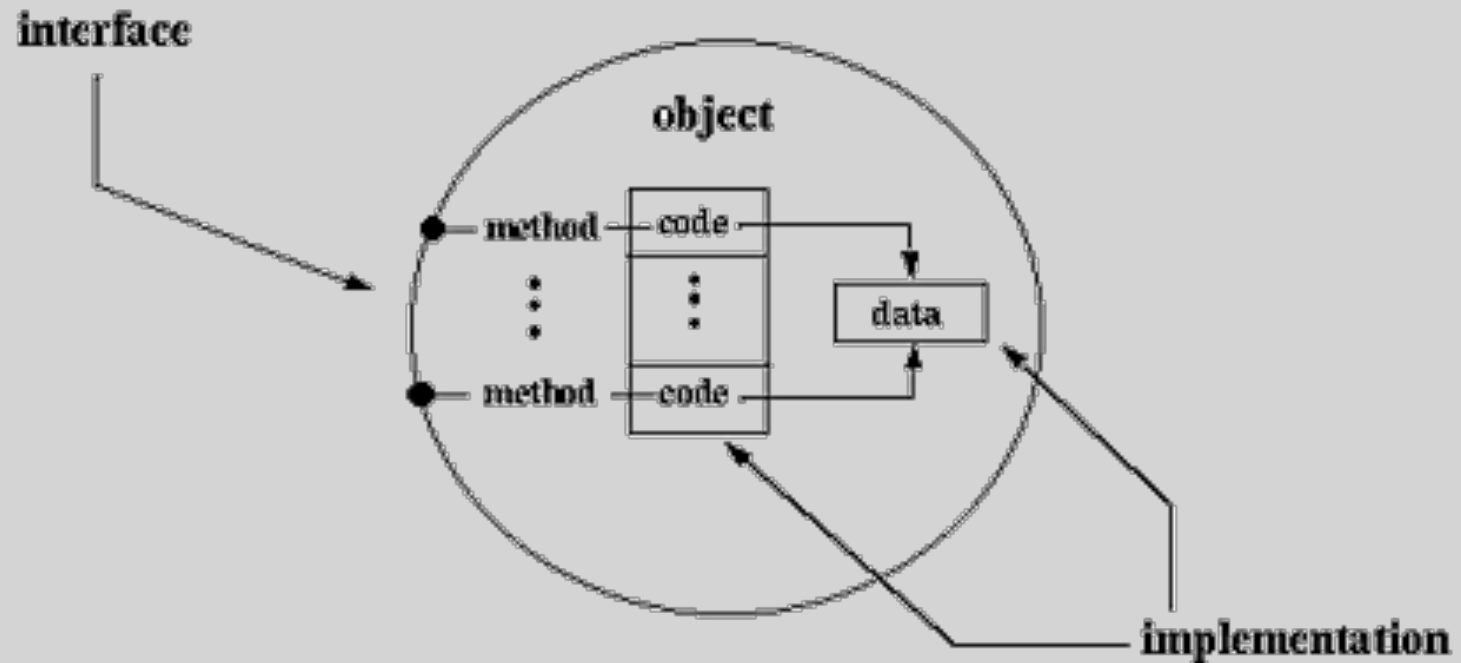
Abstraction kavramının Yazılıma Aktarılması



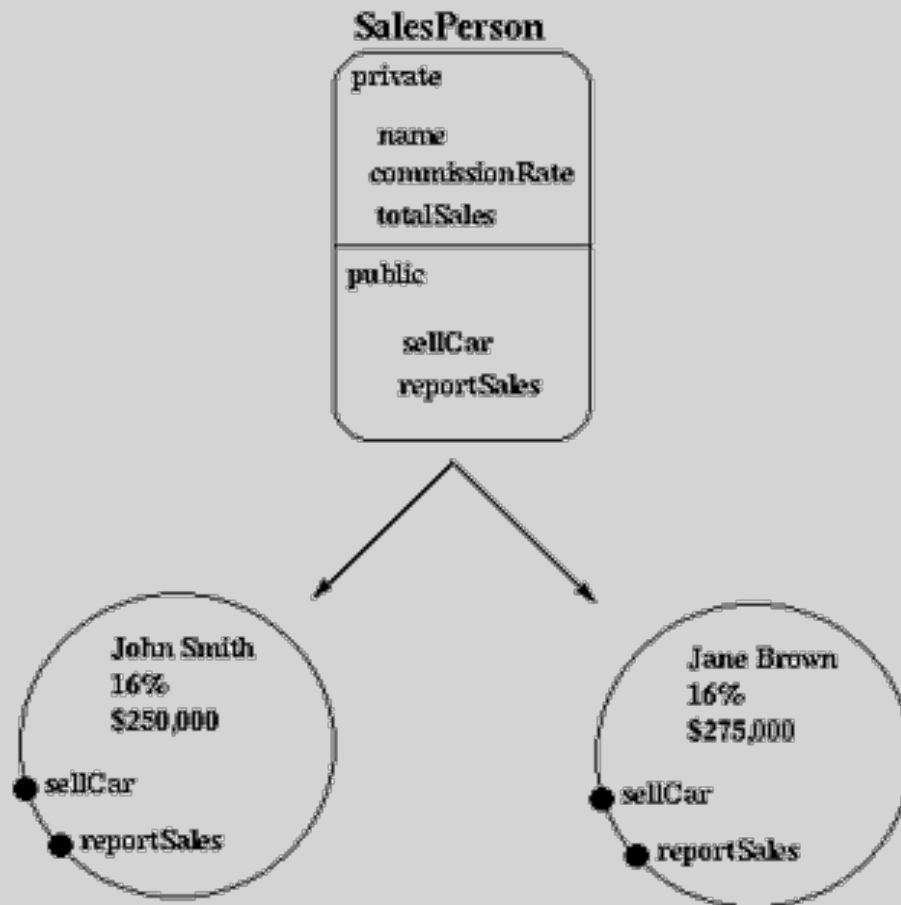
Bir sınıfın genel yapısı



Bir nesne' nin genel yapısı

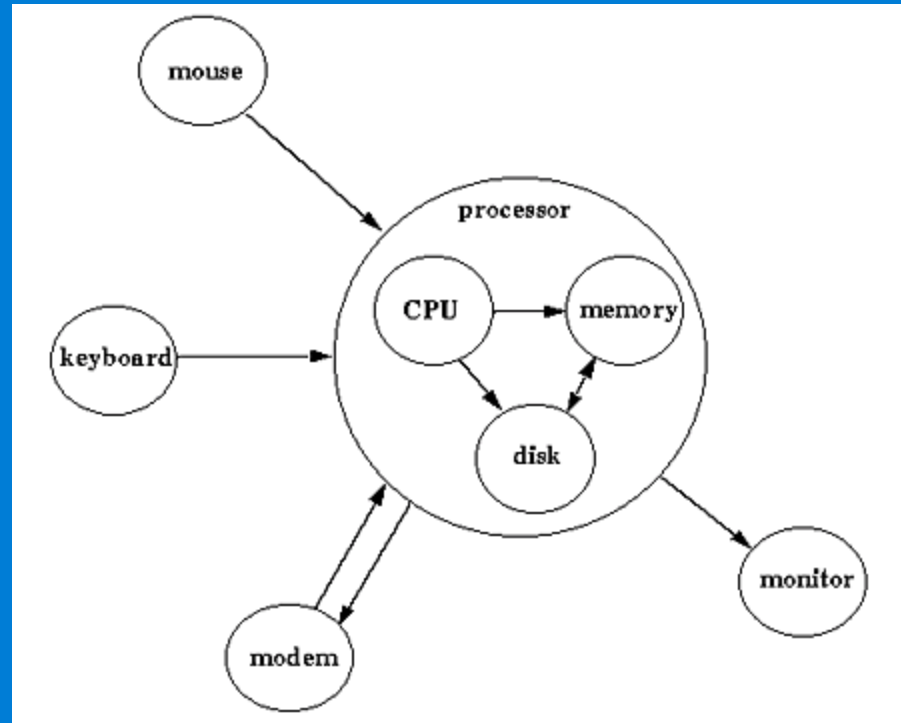


Bir sınıfın çoklu türetilmesi



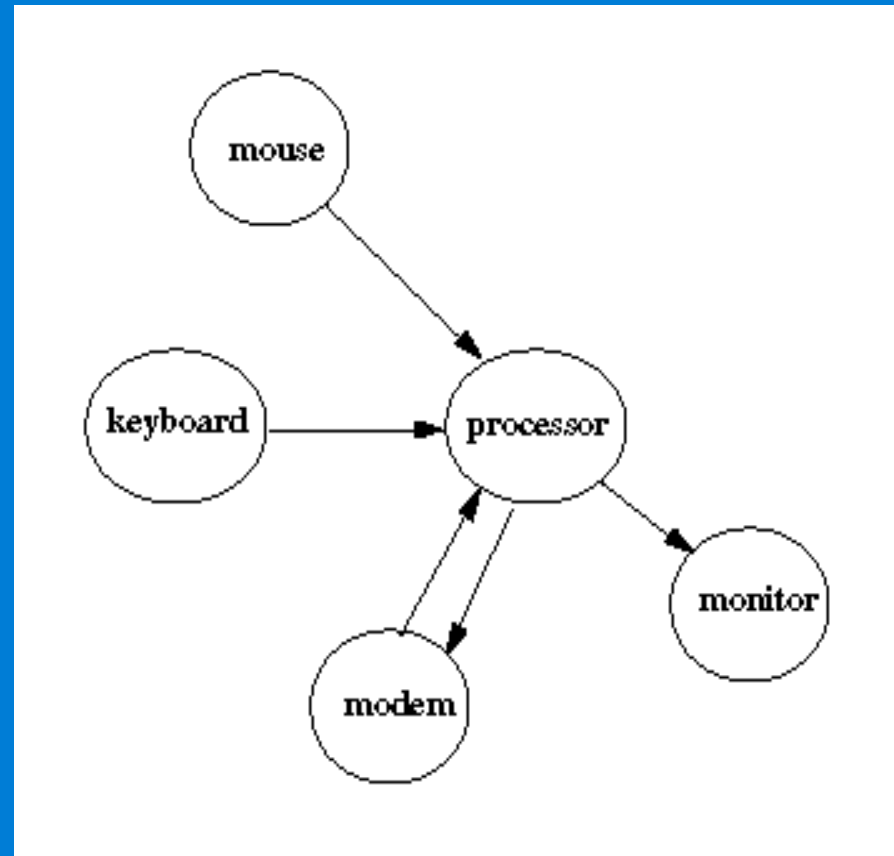
Composition

Farklı kısımların
birleştirilmesi ile
bir sınıfın
oluşturulması



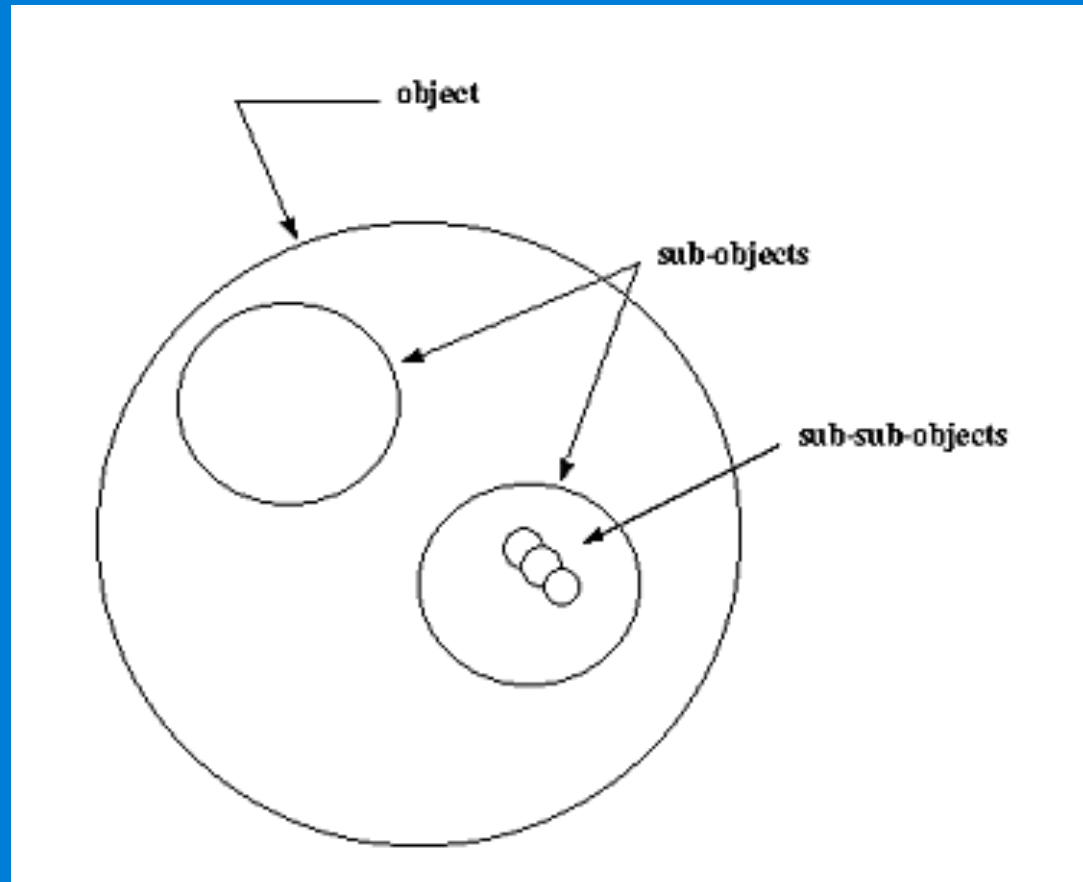
Association

Bağımsız olarak
oluşturulan ve
dışarıdan
görülebilen
kısımların bir
bileşimi



Aggregation

Composition neticesinde oluşan kısımları gizleyen (encapsulate) bir iç gruplama şeklidir



Genelleştirme (Generalization)

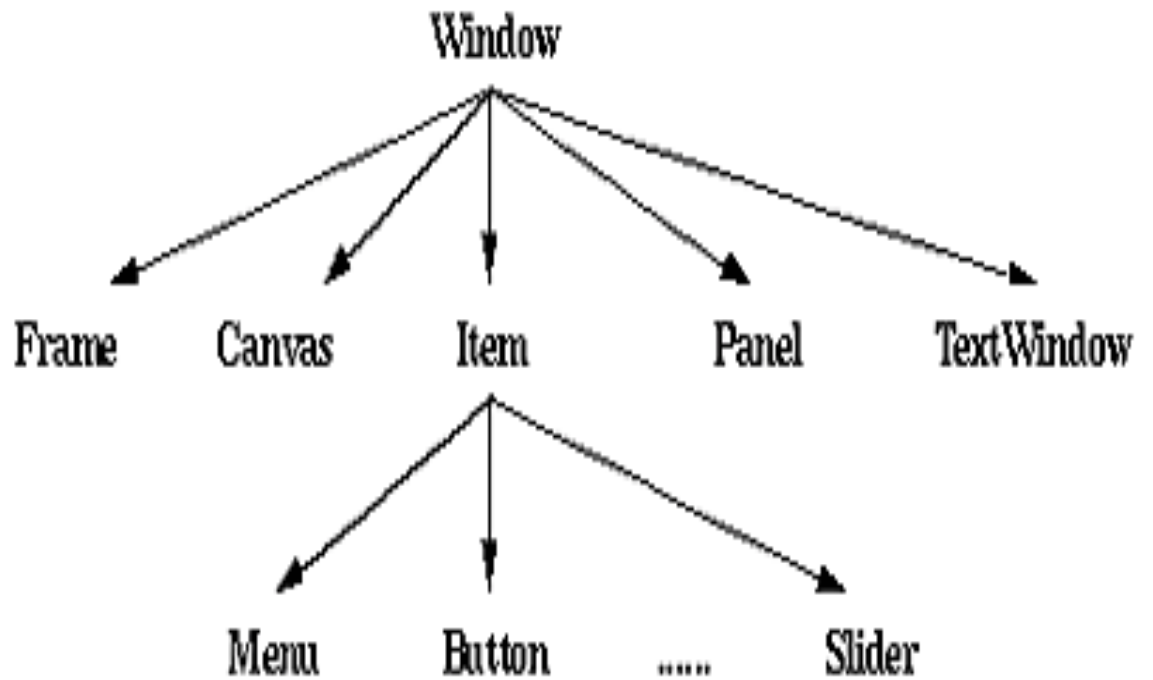
Soyutlama' nın (abstraction) ortak özelliklerinin tanımlanması ve düzenlenmesi.

- hierarchy
- genericity
- design patterns

-
-
-

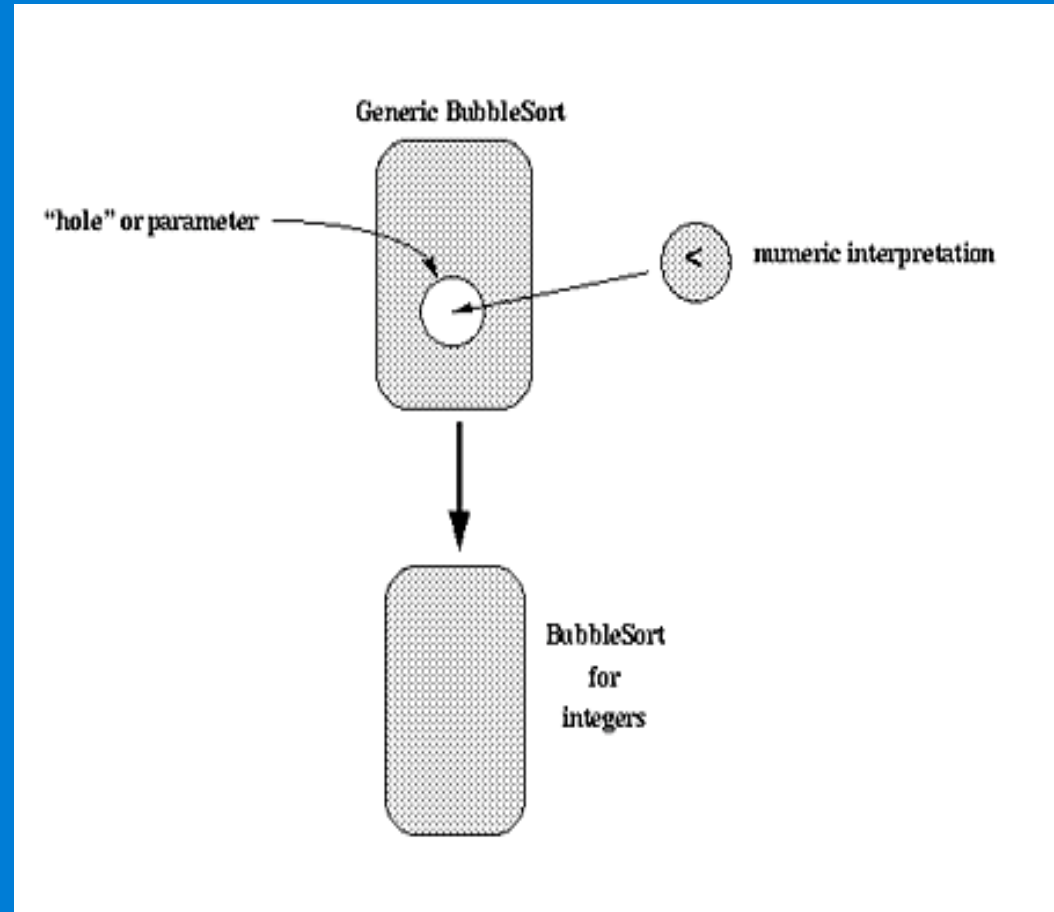
Hierarchy

A generalization
based on an “is-a”
relation



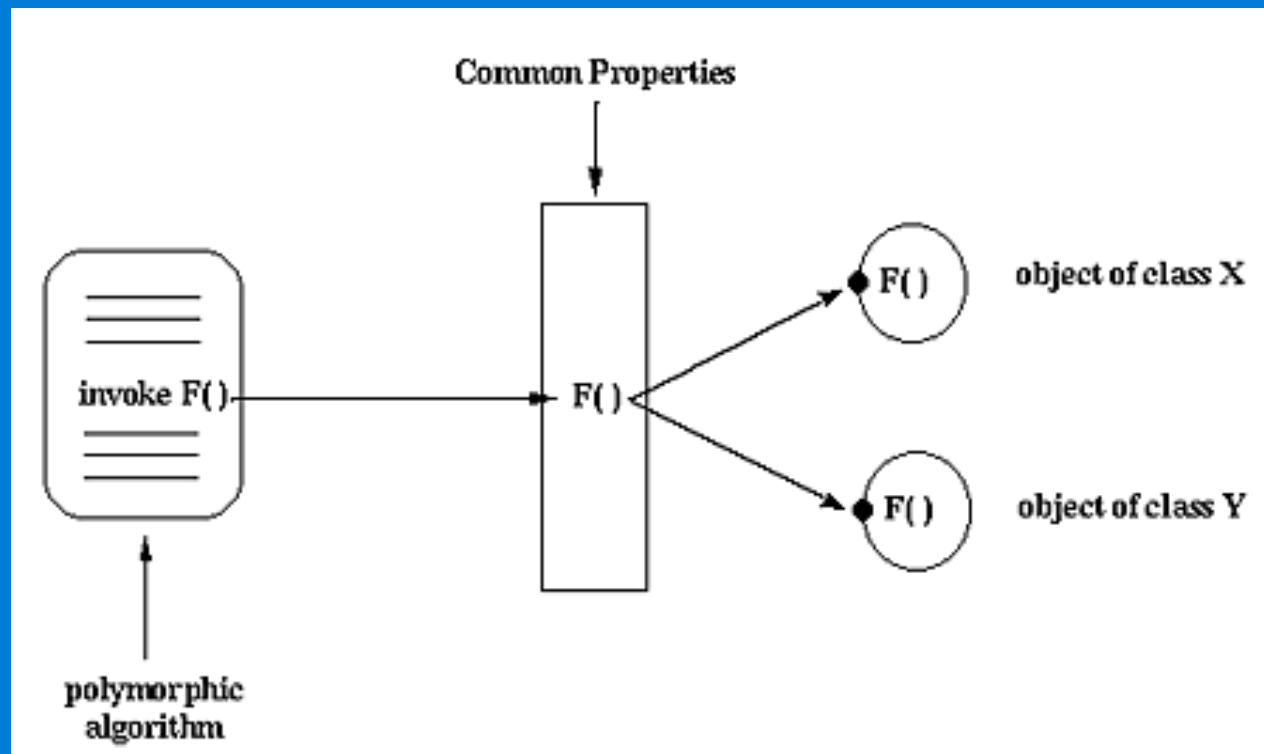
Genericity

Parametre olarak verilen, diğer özellikleri belirlenmemiş, soyutlamalara göre tanımlanmış bir genelleştirme dir.



Polymorphism

Kesin tiplerini bilmeksizin – sadece yaygın özellikleri ile nesneleri yönlendirme işlemidir.

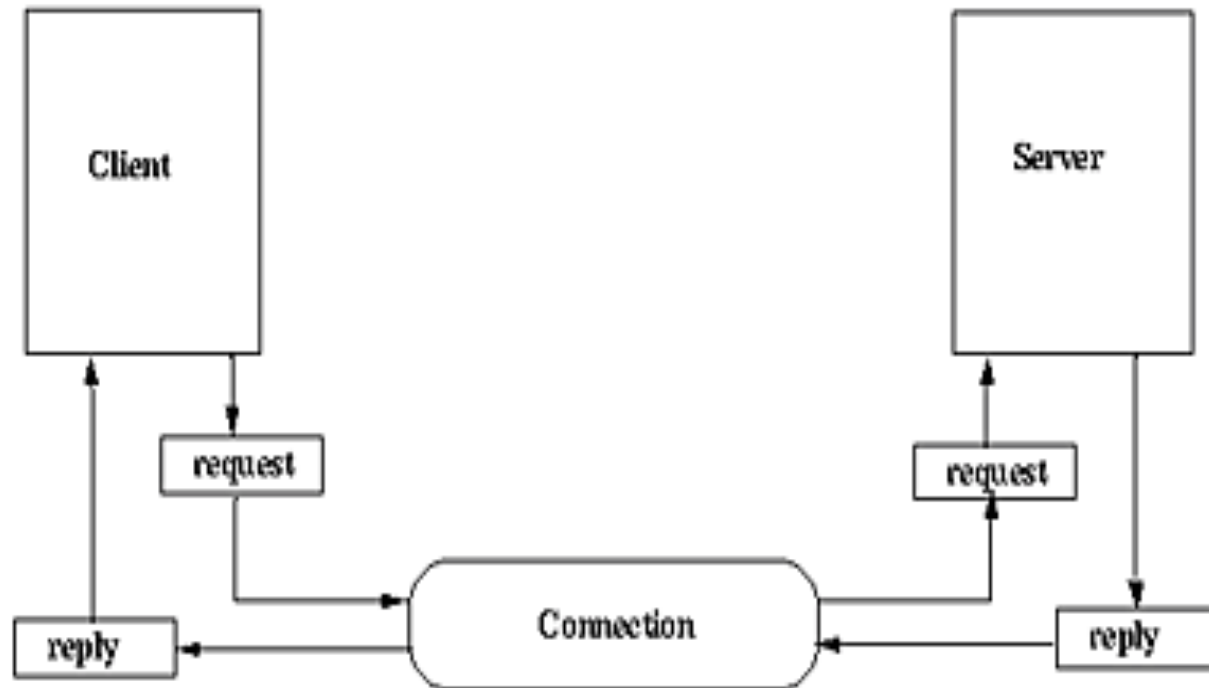


Design Patterns

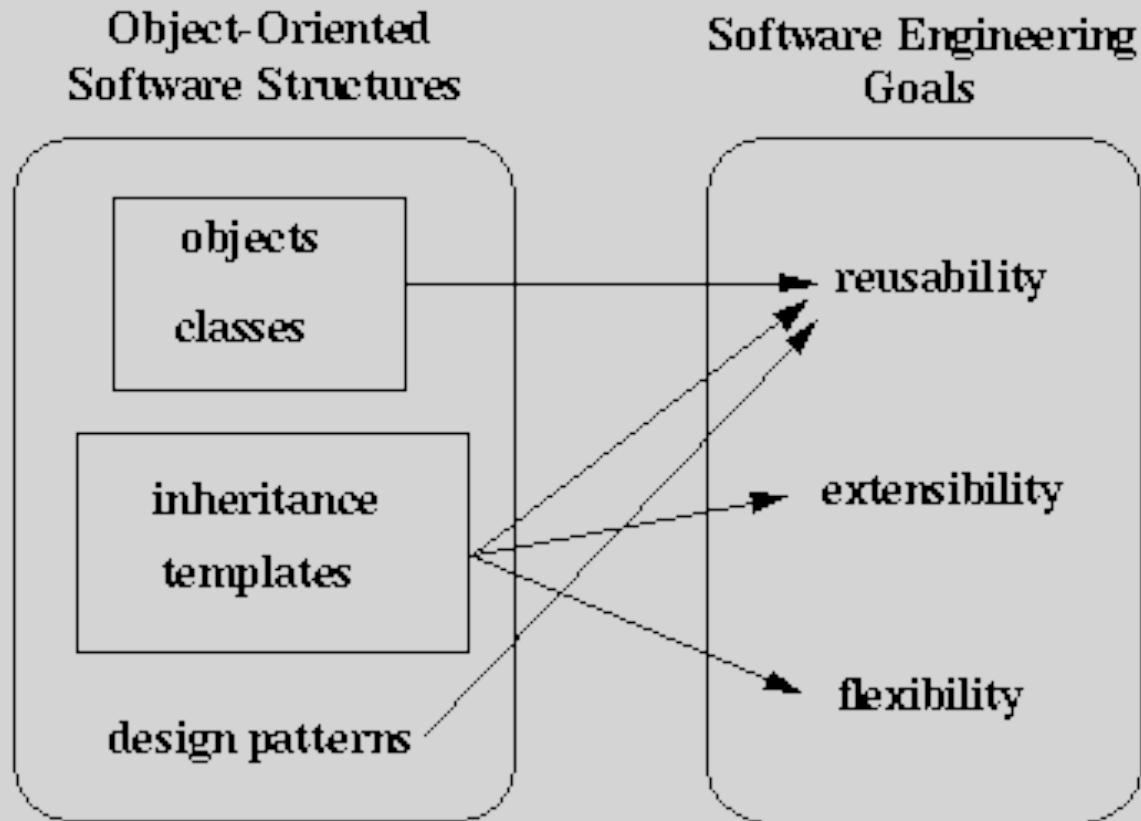
Ortak, erişilebilir bir biçimde tasarım bilgisini ve deneyimini yakalamak.

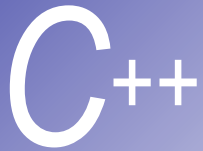
- Deneyimsizlerin uzmanlardan öğrenmesine izin verir
- Daha iyi tasarım oluşturmaya olanak tanır
- Bir “tasarım dağarcığı” oluşturur

Design Patterns



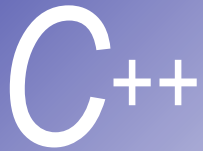
Yazılım Mühendisliği Hedefleri





'const' Nesneler

- **En düşük izin prensibi:** Nesnelere sadece gerekli olduğu kadar erişim izni verin.
- **const** anahtar kelimesi kullanılarak deklare edilen nesne örnekleri **değiştirilemezler**, aksi takdirde derleyici "syntax" hatası verir.
- Örnek : **const Time noon (12, 0, 0);**
- Burada **Time** sınıfından **const** olarak deklare edilen **noon** nesnesi 12, 0, 0 başlangıç değeri ile oluşturuluyor.



‘const’ Fonksiyonlar

- `const` nesneler, `const` fonksiyonlar gerektirir.
- Bildirimi `const` olarak yapılan fonksiyonlar, kendi nesnelerindeki değişkenleri değiştiremez. `const` ifadesi hem prototipte, hem de deklarasyonda yer almalıdır.
- Örnek: `int A::getValue() const { ... };`
- **Constructor** ve **Destructor** fonksiyonlar, değişkenlerin başlangıç değerlerini atadıklarından `const` olamazlar.

C++

Örnek (const nesneler)

```
1 // Fig. 7.1: time5.h
2 // Declaration of the class Time.
3 // Member functions defined in time5.cpp
4 #ifndef TIME5_H
5 #define TIME5_H
6
7 class Time {
8 public:
9     Time( int = 0, int = 0, int = 0 ); // default constructor
10
11     // set functions
12     void setTime( int, int, int ); // set time
13     void setHour( int );           // set hour
14     void setMinute( int );         // set minute
15     void setSecond( int );         // set second
16
```

Const olmayan
fonksiyonlar.

C++

Örnek (devam)

```
17 // get functions (normally declared const)
18 int getHour() const; // return hour
19 int getMinute() const; // return minute
20 int getSecond() const; // return second
21
22 // print functions (normally declared const)
23 void printMilitary() const; // print military time
24 void printStandard(); // print standard time
25 private:
26 int hour; // 0 - 23
27 int minute; // 0 - 59
28 int second; // 0 - 59
29 };
30
31 #endif
```

Const fonksiyonlar

Const olmayan bir fonksiyon.

C++

Örnek (devam)

```
32 // Fig. 7.1: time5.cpp
33 // Member function definitions for Time class
34 #include <iostream>
35
36 using std::cout;
37
38 #include "time5.h"
39
40 // Constructor function to initialize private data.
41 // Default values are 0 (see class definition).
42 Time::Time( int hr, int min, int sec )
43     { setTime( hr, min, sec ); }
44
45 // Set the values of hour, minute, and second.
46 void Time::setTime( int h, int m, int s )
47 {
48     setHour( h );
49     setMinute( m );
50     setSecond( s );
51 }
```

Constructor fonksiyonlar **const** olamamasına rağmen, **const** nesnelerde normal constructor fonksiyonlar kullanılabilir.

C++

Örnek (devam)

```
53 // Set the hour value
54 void Time::setHour( int h )
55     { hour = ( h >= 0 && h < 24 ) ? h : 0; }
56
57 // Set the minute value
58 void Time::setMinute( int m )
59     { minute = ( m >= 0 && m < 60 ) ? m : 0; }
60
61 // Set the second value
62 void Time::setSecond( int s )
63     { second = ( s >= 0 && s < 60 ) ? s : 0; }
64
65 // Get the hour value
66 int Time::getHour() const { return hour; }
67
68 // Get the minute value
69 int Time::getMinute() const { return minute; }
```

const anahtar sözcüğü
fonksiyon prototipinde
de, tanımlanmasında da
bulunmalıdır.

C++

Örnek (devam)

```
71 // Get the second value
72 int Time::getSecond() const { return second; }
73
74 // Display military format time: HH:MM
75 void Time::printMilitary() const
76 {
77     cout << ( hour < 10 ? "0" : "" )
78         << ( minute < 10 ? "0" : "" )
79 }
80
81 // Display standard format time: HH:MM:SS AM (OR PM)
82 void Time::printStandard() // should be const
83 {
84     cout << ( ( hour == 12 ) ? 12 : hour % 12 ) << ":"
85         << ( minute < 10 ? "0" : "" ) << minute << ":"
86         << ( second < 10 ? "0" : "" ) << second
87         << ( hour < 12 ? " AM" : " PM" );
88 }
```

const olmayan fonksiyonlar, herhangi bir nesne değerini değiştirmeseler bile (printStandard fonksiyonunda olduğu gibi) const nesne örneklerince çağırılamazlar.

C++

Örnek (devam)

```

89 // Fig. 7.1: fig07_01.cpp
90 // Attempting to access a const object with
91 // non-const member functions.
92 #include "time5.h"
93
94 int main()
95 {
96     Time wakeUp( 6, 45, 0 );           // non-constant object
97     const Time noon( 12, 0, 0 );       // constant object
98
99                                     // MEMBER FUNCTION   OBJECT
100    wakeUp.setHour( 18 );               // non-const        non-const
101
102    noon.setHour( 12 );                 // non-const        const
103
104    wakeUp.getHour();                   // const            non-const
105
106    noon.getMinute();                   // const            const
107    noon.printMilitary();                // const            const
108    noon.printStandard();                // non-const        const
109    return 0;
110 }

```

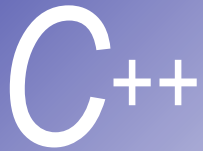
Compiler syntax
hatası verir.

C++

Ekran Çıktısı

```
Compiling...
Fig07_01.cpp
d:fig07_01.cpp(14) : error C2662: 'setHour' : cannot convert 'this'
pointer from 'const class Time' to 'class Time &'
Conversion loses qualifiers
d:\fig07_01.cpp(20) : error C2662: 'printStandard' : cannot convert
'this' pointer from 'const class Time' to 'class Time &'
Conversion loses qualifiers
Time5.cpp
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```



Const nesnelerde üyelere ilk değer verme

- **Increment** sınıfındaki **const increment** değişkenine **ilk değer (i)** verme işi **constructor Increment** fonksiyonunda yapılır:

```
Increment::Increment( int c, int i )  
    : increment( i )  
    { count = c; }
```

- `: increment(i)` ifadesi `increment`'e `i` başlangıç değerini atar. Tüm üyelere bu şekilde başlangıç değeri verilebilir. **const** ve referans tipleri için bu zorunludur.
- Birden çok atama yapılacaksa üyeler virgül ile ayrılarak yazılır.

C++

Örnek (ilk değer verme)

```
1 // Fig. 7.2: fig07_02.cpp
2 // Using a member initializer to initialize a
3 // constant of a built-in data type.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 class Increment {
10 public:
11     Increment( int c = 0, int i = 1 );
12     void addIncrement() { count += increment; }
13     void print() const;
14
15 private:
16     int count;
17     const int increment;    // const data member
18 };
19
```

C++

Örnek(devamı)

```
20 // Constructor for class Increment
21 Increment::Increment( int c, int i )
22     : increment( i )    // initializer for const member
23 { count = c; }
24
25 // Print the data
26 void Increment::print() const
27 {
28     cout << "count = " << count
29         << ", increment = " << increment << endl;
30 }
31
32 int main()
33 {
34     Increment value( 10, 5 );
35
36     cout << "Before incrementing: ";
37     value.print();
38
```

Bu iş **initializer** bölümünde değilde, doğrudan `increment`'e bir atama olarak (`increment = i`) yapılırsa idi compiler hata verecekti.

C++

Örnek(devamı)

```
39     for ( int j = 0; j < 3; j++ ) {  
40         value.addIncrement();  
41         cout << "After increment " << j + 1 << ": ";  
42         value.print();  
43     }  
44  
45     return 0;  
46 }
```

```
Before incrementing: count = 10, increment = 5  
After increment 1: count = 15, increment = 5  
After increment 2: count = 20, increment = 5  
After increment 3: count = 25, increment = 5
```