Sınıflar ve Nesneler II

Dr. Metin Özkan

Sabit (`const`) Nesneler ve Üye Fonksiyonlar

- Bu özelliğin kullanılması, iyi yazılım mühendisliği için önemlidir.
- `const` kelimesi ile yaratılan nesne, değiştirilemez ve değiştirme teşebbüsü olan kodlarda, derleme hatası olur.
  - `p1.setXY(1,2); //derleme hatası //verir`
- C++, sabit nesnelerin üye fonksiyonları sabit tanımlanmadı ise, nesneyi değiştirmiyor olsa da çağrısına izin verilmez.
  - `p1.print(); //derleme hatası //verir`

pointApp.cpp

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

Sabit (`const`) Nesneler ve Üye Fonksiyonlar

- Üye fonksiyon, sabit (const) tanımlandı ise, sabit nesnenin bu üye fonksiyonuna çağrı yapılabilir.
  - `p1.print(); //derleme hatası //vermez`

pointApp.cpp

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

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

```
1. //
2. // Time.h
3. // Time class definition with const member functions.
4. // Member functions defined in Time.cpp.
5. #ifndef TIME_H
6. #define TIME_H
7.
8. class Time
9. {
10. public:
11.    Time( int = 0, int = 0, int = 0 ); // default constructor
12.
13.    // set functions
14.    void setTime( int, int, int ); // set time
15.    void setHour( int ); // set hour
16.    void setMinute( int ); // set minute
17.    void setSecond( int ); // set second
18.
19.    // get functions (normally declared const)
20.    int getHour() const; // return hour
21.    int getMinute() const; // return minute
22.    int getSecond() const; // return second
```

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

```
23.     // print functions (normally declared const)
24.     void printUniversal() const; // print universal time
25.     void printStandard(); // print standard time (should be const)
26. private:
27.     int hour; // 0 - 23 (24-hour clock format)
28.     int minute; // 0 - 59
29.     int second; // 0 - 59
30. }; // end class Time
31.
32. #endif
```

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

```
1. // Time.cpp
2. // Time class member-function definitions.
3. #include <iostream>
4. #include <iomanip>
5. #include "Time.h" // include definition of class Time
6. using namespace std;
7.
8. // constructor function to initialize private data;
9. // calls member function setTime to set variables;
10. // default values are 0 (see class definition)
11. Time::Time( int hour, int minute, int second )
12. {
13.    setTime( hour, minute, second );
14. } // end Time constructor
15.
16. // set hour, minute and second values
17. void Time::setTime( int hour, int minute, int second )
18. {
19.    setHour( hour );
20.    setMinute( minute );
21.    setSecond( second );
22. } // end function setTime
```

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

```
23. // set hour value
24. void Time::setHour( int h )
25. {
26.    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
27. } // end function setHour
28.
29. // set minute value
30. void Time::setMinute( int m )
31. {
32.    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
33. } // end function setMinute
34.
35. // set second value
36. void Time::setSecond( int s )
37. {
38.    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
39. } // end function setSecond
```

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

```
40. // return hour value
41. int Time::getHour() const // get functions should be const
42. {
43.    return hour;
44. } // end function getHour
45.
46. // return minute value
47. int Time::getMinute() const
48. {
49.    return minute;
50. } // end function getMinute
51.
52. // return second value
53. int Time::getSecond() const
54. {
55.    return second;
56. } // end function getSecond
```

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

```
57. // print Time in universal-time format (HH:MM:SS)
58. void Time::printUniversal() const
59. {
60.    cout << setfill( '0' ) << setw( 2 ) << hour << ":"
61.         << setw( 2 ) << minute << ":" << setw( 2 ) << second;
62. } // end function printUniversal
63. // print Time in standard-time format (HH:MM:SS AM or PM)
64. void Time::printStandard() // note lack of const declaration
65. {
66.    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
67.         << ":" << setfill( '0' ) << setw( 2 ) << minute
68.         << ":" << setw( 2 ) << second << ( hour < 12 ? " AM" : " PM" );
69. } // end function printStandard
```

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

```
1. // main.cpp
2. // Attempting to access a const object with non-const member functions.
3. #include "Time.h" // include Time class definition
4.
5. int main()
6. {
7.    Time wakeUp( 6, 45, 0 ); // non-constant object
8.    const Time noon( 12, 0, 0 ); // constant object
9.
10.    // OBJECT MEMBER FUNCTION
11.    wakeUp.setHour( 18 ); // non-const non-const
12.
13.    noon.setHour( 12 ); // const non-const
14.
15.    wakeUp.getHour(); // non-const const
16.
17.    noon.getMinute(); // const const
18.    noon.printUniversal(); // const const
19.
20.    noon.printStandard(); // const non-const
21. } // end main
```

Sabit (`const`) Nesneler ve Üye Fonksiyonlar – Time sınıfı

Microsoft Visual C++ compiler error messages:

```
C:\cpphtp7_examples\ch10\Fig10_01_03\fig10_03.cpp(13) : error C2662:
   'Time::setHour' : cannot convert 'this' pointer from 'const Time' to
   'Time &'
        Conversion loses qualifiers
C:\cpphtp7_examples\ch10\Fig10_01_03\fig10_03.cpp(20) : error C2662:
   'Time::printStandard' : cannot convert 'this' pointer from 'const Time' to
   'Time &'
        Conversion loses qualifiers
```

GNU C++ compiler error messages:

```
fig10_03.cpp:13: error: passing 'const Time' as 'this' argument of
   'void Time::setHour(int)' discards qualifiers
fig10_03.cpp:20: error: passing 'const Time' as 'this' argument of
   'void Time::printStandard()' discards qualifiers
```

Sabit (const) Veri Üyeleri (Data Members)

- Üye başlatıcı sözdizimi (Member initializer syntax)
- Bütün veri üyelerin, üye başlatıcı sözdiziminde başlatılabilir; ancak, sabit (const) veri üyeler mutlaka burada başlatılmak zorundadır.
- Üye başlatıcı sözdizimi, yapıcı fonksiyonda bulunur ve fonksiyon parametrelerinden sonra (:) ile başlar.
  - `Point():` Bu kısım, üye başlatıcı (member initializer) alanı `{}`

pointApp.cpp

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

Sabit (const) Veri Üyeleri (Data Members)

- Bütün veri üyelerin, üye başlatıcı sözdiziminde başlatılabilir; ancak, sabit (const) veri üyeler mutlaka burada başlatılmak zorundadır.

```
Point(int X,int Y,int i):x(X),y(Y),index(i){

}
```

Tercihli

zorunlu

Değer atanan üye değişken

Üye değişkene atanan değer

pointApp.cpp

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

Sabit (const) Veri Üyeleri (Data Members) (Doğru kullanım)

```
1. // Increment.h
2. // Definition of class Increment.
3. #ifndef INCREMENT_H
4. #define INCREMENT_H
5.
6. class Increment
7. {
8. public:
9.    Increment( int c = 0, int i = 1 ); // default constructor
10.
11.    // function addIncrement definition
12.    void addIncrement()
13.    {
14.       count += increment;
15.    } // end function addIncrement
16.
17.    void print() const; // prints count and increment
18. private:
19.    int count;
20.    const int increment; // const data member
21. }; // end class Increment
22.
23. #endif
```

Sabit (const) Veri Üyeleri (Data Members) (Doğru kullanım)

```
1. // Increment.cpp
2. // Member-function definitions for class Increment demonstrate using a
3. // member initializer to initialize a constant of a built-in data type.
4. #include <iostream>
5. #include "Increment.h" // include definition of class Increment
6. using namespace std;
7.
8. // constructor
9. Increment::Increment( int c, int i )
10.    : count( c ), // initializer for non-const member
11.      increment( i ) // required initializer for const member
12. {
13.    // empty body
14. } // end constructor Increment
15.
16. // print count and increment values
17. void Increment::print() const
18. {
19.    cout << "count = " << count << ", increment = " << increment << endl;
20. } // end function print
```

Sabit (const) Veri Üyeleri (Data Members) (Doğru kullanım)

```
1. // main.cpp
2. // Program to test class Increment.
3. #include <iostream>
4. #include "Increment.h" // include definition of class Increment
5. using namespace std;
6.
7. int main()
8. {
9.     Increment value( 10, 5 );
10.
11.    cout << "Before incrementing: ";
12.    value.print();
13.
14.    for ( int j = 1; j <= 3; j++ )
15.    {
16.        value.addIncrement();
17.        cout << "After increment " << j << ": ";
18.        value.print();
19.     } // end for
20. } // end main
```

Sabit (const) Veri Üyeleri (Data Members) (Doğru kullanım)

```
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
```

Sabit (const) Veri Üyeleri (Data Members) (Hatalı kullanım)

```
1.  // Increment.cpp
2.  // Erroneous attempt to initialize a constant of a built-in data
3.  // type by assignment.
4.  #include <iostream>
5.  #include "Increment.h" // include definition of class Increment
6.  using namespace std;
7.
8.  // constructor; constant member 'increment' is not initialized
9.  Increment::Increment( int c, int i )
10. {
11.    count = c; // allowed because count is not constant
12.    increment = i; // ERROR: Cannot modify a const object
13. } // end constructor Increment
14.
15. // print count and increment values
16. void Increment::print() const
17. {
18.    cout << "count = " << count << ", increment = " << increment << endl;
19. } // end function print
```

Sabit (const) Veri Üyeleri (Data Members) (Hatalı kullanım)

```
1. // main.cpp
2. // Program to test class Increment.
3. #include <iostream>
4. #include "Increment.h" // include definition of class Increment
5. using namespace std;
6.
7. int main()
8. {
9.     Increment value( 10, 5 );
10.
11.    cout << "Before incrementing: ";
12.    value.print();
13.
14.    for ( int j = 10; j <= 3; j++ )
15.    {
16.        value.addIncrement();
17.        cout << "After increment " << j << ": ";
18.        value.print();
19.    } // end for
20. } // end main
```

Sabit (const) Veri Üyeleri (Data Members) (Hatalı kullanım)

Microsoft Visual C++ compiler error messages:

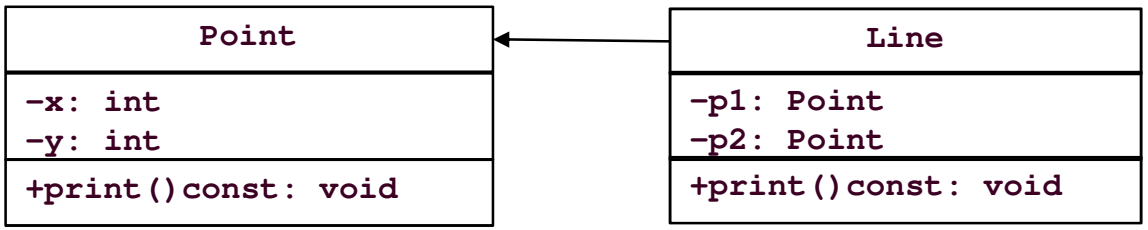
```
C:\cpphtp7_examples\ch10\Fig10_07_09\Increment.cpp(10) : error C2758:
   'Increment::increment' : must be initialized in constructor base/member
   initializer list
       C:\cpphtp7_examples\ch10\Fig10_07_09\increment.h(20) : see
          declaration of 'Increment::increment'
C:\cpphtp7_examples\ch10\Fig10_07_09\Increment.cpp(12) : error C2166:
   l-value specifies const object
```

GNU C++ compiler error messages:

```
Increment.cpp:9: error: uninitialized member 'Increment::increment' with
   'const' type 'const int'
Increment.cpp:12: error: assignment of read-only data-member
   'Increment::increment'
```

Bileşim(Composition): Sınıf Üyesi Nesneler

- Bir sınıf, başka bir sınıfın nesnesine kendi üyesi olarak sahip olabilir. Bir sınıf üyesinin başka bir nesne olması durumuna Bileşim (Composition) denir.
  - *Line sınıfı, Point sınıfının nesnelerine sahiptir.*
- UML Sınıf Diyagram gösterimi aşağıdaki gibidir.




```
#include <iostream>
using namespace std;
class Point{
        int x;
        int y;
public:
        Point(int X=0,int Y=0)
        :x(X),y(Y){}
        void print() const{
             cout<<x<<" "<<y;
        }
};
class Line{
        Point p1;
        Point p2;
public:
        void print() const{
             p1.print();
             cout<<" - ";
             p2.print();
        }
};
int main(){
        Line line;
        line.print();
        return 0;
}
```

Bileşim(Composition): Sınıf Üyesi Nesneler

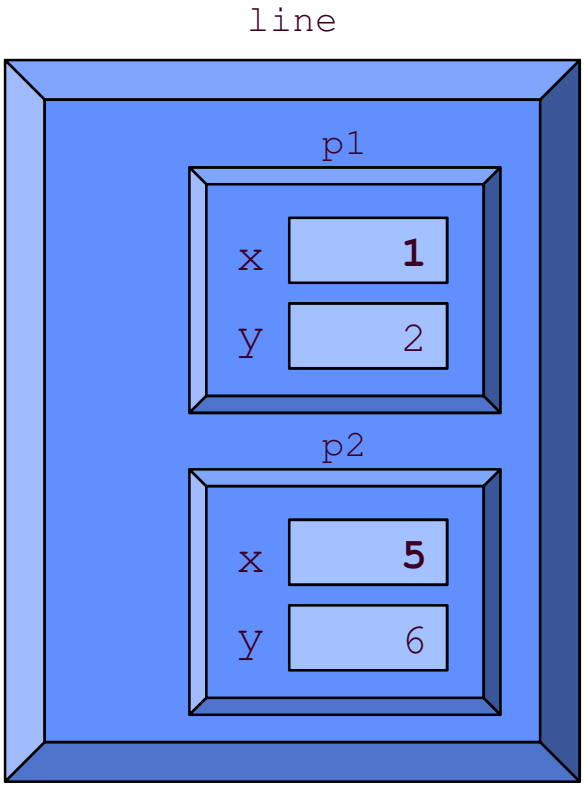
- Bir nesnenin yapıcı fonksiyonu, parametreleri üye nesnesinin yapıcı fonksiyonuna üye başlatıcı (member initializer) alanında aktarır.
- Aşağıdaki komut işletildiğinde
  - `Line line(1,2,5,4);`
- Line üyesi olan iki Point nesnesi (p1 ve p2) yaratılır. Yaratılan bu nesnelere başlangıç değerleri
  - `Line(int x1,int y1,int x2,int y2)`
  - `:p1(x1,y1),p2(x2,y2){}`

şeklinde, p1 ve p2 'nin yapıcı fonksiyonlarına aktarılır.

```
#include <iostream>
using namespace std;
class Point{
          int x;
          int y;
public:
          Point(int X=0,int Y=0)
          :x(X),y(Y){}
          void print() const{
               cout<<x<<" "<<y;
          }
};
class Line{
          Point p1;
          Point p2;
public:
          Line(int x1,int y1,int x2,int y2)
                    :p1(x1,y1),p2(x2,y2){}
          void print() const{
               p1.print();
               cout<<" - ";
               p2.print();
          }
};
int main(){
          Line line(1,2,5,4);
          line.print();
          return 0;
}
```

Bileşim(Composition): Sınıf Üyesi Nesneler

- Böylece, line nesnesi




```
#include <iostream>
using namespace std;
class Point{
          int x;
          int y;
public:
          Point(int X=0,int Y=0)
          :x(X),y(Y){}
          void print() const{
               cout<<x<<" "<<y;
          }
};
class Line{
          Point p1;
          Point p2;
public:
          Line(int x1,int y1,int x2,int y2)
                    :p1(x1,y1),p2(x2,y2){}
          void print() const{
               p1.print();
               cout<<" - ";
               p2.print();
          }
};
int main(){
          Line line(1,2,5,4);
          line.print();
          return 0;
}
```

Bileşim(Composition): Sınıf Üyesi Nesneler- Date Sınıfı

```
// Date.h
// Date class definition; Member functions defined in Date.cpp
#ifndef DATE_H
#define DATE_H

class Date
{
public:
    static const int monthsPerYear = 12; // number of months in a year
    Date( int = 1, int = 1, int = 1900 ); // default constructor
    void print() const; // print date in month/day/year format
    ~Date(); // provided to confirm destruction order
private:
    int month; // 1-12 (January-December)
    int day; // 1-31 based on month
    int year; // any year

    // utility function to check if day is proper for month and year
    int checkDay( int ) const;
}; // end class Date

#endif
```

Bileşim(Composition): Sınıf Üyesi Nesneler- Date Sınıfı

```
// Date.cpp
// Date class member-function definitions.
#include <iostream>
#include "Date.h" // include Date class definition
using namespace std;

// constructor confirms proper value for month; calls
// utility function checkDay to confirm proper value for day
Date::Date( int mn, int dy, int yr )
{
    if ( mn > 0 && mn <= monthsPerYear ) // validate the month
        month = mn;
    else
    {
        month = 1; // invalid month set to 1
        cout << "Invalid month (" << mn << ") set to 1.\n";
    } // end else

    year = yr; // could validate yr
    day = checkDay( dy ); // validate the day

    // output Date object to show when its constructor is called
    cout << "Date object constructor for date ";
```

Bileşim(Composition): Sınıf Üyesi Nesneler- Date Sınıfı

```
   print();
   cout << endl;
} // end Date constructor

// print Date object in form month/day/year
void Date::print() const
{
   cout << month << '/' << day << '/' << year;
} // end function print

// output Date object to show when its destructor is called
Date::~Date()
{
   cout << "Date object destructor for date ";
   print();
   cout << endl;
} // end ~Date destructor

// utility function to confirm proper day value based on
// month and year; handles leap years, too
int Date::checkDay( int testDay ) const
{
   static const int daysPerMonth[ monthsPerYear + 1 ] =
      { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

Bileşim(Composition): Sınıf Üyesi Nesneler- Date Sınıfı

```
        // determine whether testDay is valid for specified month
        if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
            return testDay;
    // February 29 check for leap year
    if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
        ( year % 4 == 0 && year % 100 != 0 ) ) )
            return testDay;

    cout << "Invalid day (" << testDay << ") set to 1.\n";
    return 1; // leave object in consistent state if bad value
} // end function checkDay
```

Bileşim(Composition) - Employee Sınıfı

```
// Employee.h
// Employee class definition showing composition.
// Member functions defined in Employee.cpp.
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
#include "Date.h" // include Date class definition
using namespace std;

class Employee
{
public:
    Employee( const string &, const string &,
    const Date &, const Date & );
    void print() const;
    ~Employee(); // provided to confirm destruction order
private:
    string firstName; // composition: member object
    string lastName; // composition: member object
    const Date birthDate; // composition: member object
    const Date hireDate; // composition: member object
}; // end class Employee

#endif
```

Bileşim(Composition) - Employee Sınıfı

```
// Employee.cpp
// Employee class member-function definitions.
#include <iostream>
#include "Employee.h" // Employee class definition
#include "Date.h" // Date class definition
using namespace std;

// constructor uses member initializer list to pass initializer
// values to constructors of member objects
Employee::Employee( const string &first, const string &last,
    const Date &dateOfBirth, const Date &dateOfHire )
    : firstName( first ), // initialize firstName
      lastName( last ), // initialize lastName
      birthDate( dateOfBirth ), // initialize birthDate
      hireDate( dateOfHire ) // initialize hireDate
{
    // output Employee object to show when constructor is called
    cout << "Employee object constructor: "
         << firstName << ' ' << lastName << endl;
} // end Employee constructor
```

Copy constructor (will be defined)

Bileşim(Composition) - Employee Sınıfı

```
// print Employee object
void Employee::print() const
{
    cout << lastName << ", " << firstName << " Hired: ";
    hireDate.print();
    cout << " Birthday: ";
    birthDate.print();
    cout << endl;
} // end function print

// output Employee object to show when its destructor is called
Employee::~Employee()
{
    cout << "Employee object destructor: "
         << lastName << ", " << firstName << endl;
} // end ~Employee destructor
```

Bileşim(Composition) – Employee Nesnesi

```
// main.cpp
// Demonstrating composition--an object with member objects.
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

int main()
{
     Date birth( 7, 24, 1949 );
     Date hire( 3, 12, 1988 );
     Employee manager( "Bob", "Blue", birth, hire );

     cout << endl;
     manager.print();

     cout << "\nTest Date constructor with invalid values:\n";
     Date lastDayOff( 14, 35, 1994 ); // invalid month and day
     cout << endl;
} // end main
```

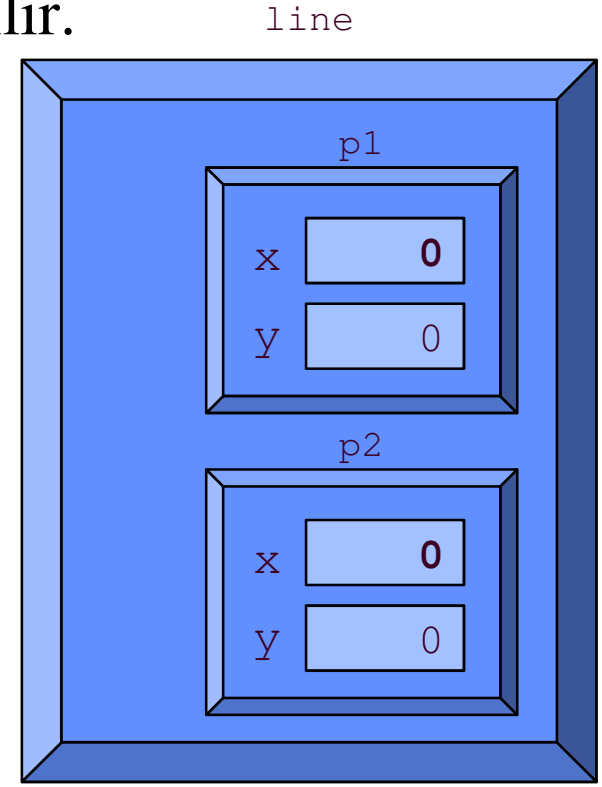
Bileşim(Composition) – Employee Nesnesi

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Blue

Blue, Bob  Hired: 3/12/1988  Birthday: 7/24/1949

Test Date constructor with invalid values:
Invalid month (14) set to 1.
Invalid day (35) set to 1.
Date object constructor for date 1/1/1994

Date object destructor for date 1/1/1994
Employee object destructor: Blue, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
```

There are actually five constructor calls when an **Employee** is constructed—two calls to the **string** class's constructor (lines 12–13 of Fig. 10.13), two calls to the **Date** class's default copy constructor (lines 14–15 of Fig. 10.13) and the call to the **Employee** class's constructor.

Bileşim(Composition): Sınıf Üyesi Nesneler

- Eğer, üye nesne üye başlatıcı (member initializer) alanında başlatılmadı ise, üye nesnenin varsayılan yapıcı fonksiyonu çağrılır.




```
#include <iostream>
using namespace std;
class Point{
          int x;
          int y;
public:
          Point(int X=0,int Y=0)
          :x(X),y(Y){}
          void print() const{
               cout<<x<<" "<<y;
          }
};
class Line{
          Point p1;
          Point p2;
public:
          Line(){
          }
          void print() const{
               p1.print();
               cout<<" - ";
               p2.print();
          }
};
int main(){
          Line line;
          line.print();
          return 0;
}
```

Kopya Yapıcı Fonksiyon (Copy Constructor)

- Bazen, yeni bir nesne, mevcut bir nesnenin kopyası olarak yaratılmak istenir.
- Bu fonksiyon, yapıcı fonksiyonlar ile aynı özelliklere sahiptir. Tek fark, fonksiyon tek bir parametre alır ve parametre kendi sınıfından bir nesneyi referans olur.
  - `Point(const Point &p);`
- Eğer bir sınıf altında kopya yapıcı fonksiyon (copy constructor) tanımlanmadı ise, derleyici otomatik olarak bir tane yaratır.
- Varsayılan kopya yapıcı fonksiyon, sadece değişkenlerdeki değerleri kopyalar.

pointApp.cpp

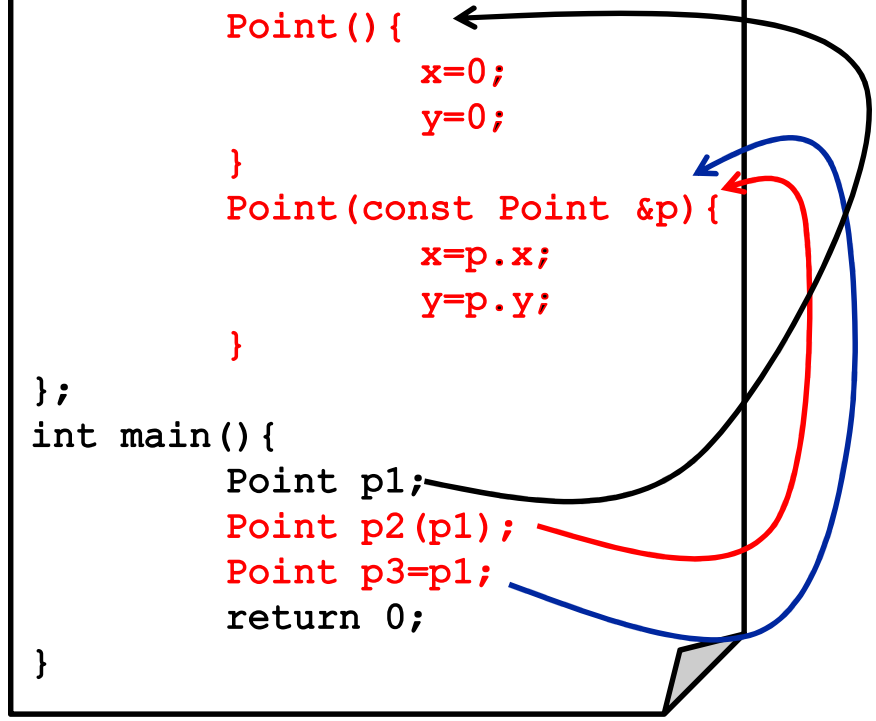
```
#include <iostream>
using namespace std;
class Point{
        int x;
        int y;
public:
        Point(){
                x=0;
                y=0;
        }
        Point(const Point &p){
                x=p.x;
                y=p.y;
        }
};
int main(){
        Point p1;
        Point p2(p1);
        Point p3=p1;
        p1.print();
        return 0;
}
```

Kopya Yapıcı Fonksiyon (Copy Constructor)

- Kopya yapıcı fonksiyon, bir nesne bir başka nesnenin kopyası olarak yaratılıyorsa, çağrılır.
- Aşağıdaki komutlarda, bu fonksiyon çağrılır.
  - `Point p2(p1);`
- Ya da
  - `Point p3=p1;`
- Komutları p2 ve p3 nesnelerinin p1 nesnesinin kopyası olarak yaratılmasını sağlar.

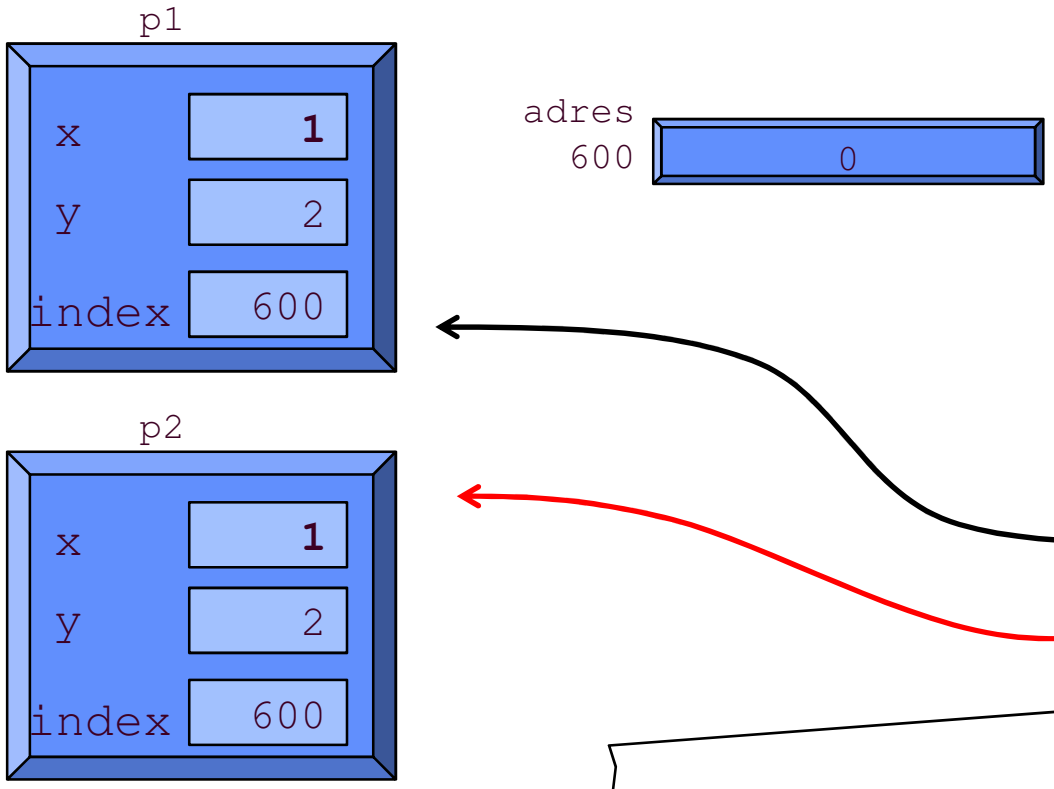
pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
        int x;
        int y;
public:
        Point(){
                x=0;
                y=0;
        }
        Point(const Point &p){
                x=p.x;
                y=p.y;
        }
};
int main(){
        Point p1;
        Point p2(p1);
        Point p3=p1;
        return 0;
}
```



Kopya Yapıcı Fonksiyon (Copy Constructor)

- Dikkat: Eğer, bir sınıf gösterge (pointer) bulunduruyorsa, varsayılan kopya yapıcı fonksiyon hatalı sonuçlar doğurur.
- Eğer varsayılan yapıcı fonksiyon kullanılıyorsa;



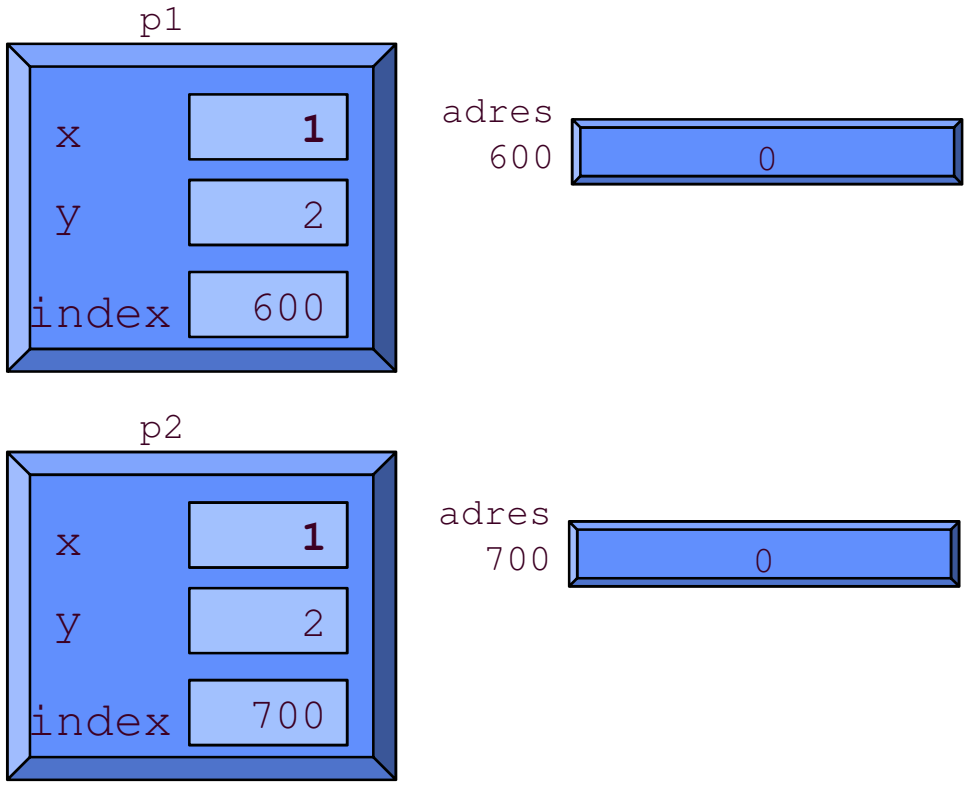
pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
          int x;
          int y;
          int *index;
public:
          Point(int X=0,int Y=0)
                  :x(X),y(Y){
                index=new int(0);
          }
          void setIndex(int i){
                  *index=i;
          }
};
int main(){
          Point p1(1,2);
          Point p2(p1);
          p1.setIndex(1);
          return 0;
}
```

Bu komut ile p1 nesnesinin index değeri değiştirilirse, p2 bundan etkilenir mi?

Kopya Yapıcı Fonksiyon (Copy Constructor)

- Eğer bir yapıcı fonksiyon eklersek;



pointApp.cpp

```
#include <iostream>
using namespace std;
class Point{
        int x;
        int y;
        int *index;
public:
        Point(int X=0,int Y=0)
              :x(X),y(Y){
             index=new int(0);
        }
        Point(const Point &p){
              x=p.x; y=p.y;
              index=new int(p->index);

        }
        void setIndex(int i){
              *index=i;
        }
};
int main(){
        Point p1(1,2);
        Point p2(p1);
        p1.setIndex(1);
        return 0;
}
```

Kopya Yapıcı Fonksiyon (Copy Constructor)

```
class String{
      int size;
      char *content;
public:
      String(const char* str){                                  //constructor
                size=0; int i=0;
                while(str[i]!='\0') { size++; i++;}
                content=new char[size];
                for(int i=0;i<size;i++) content[i]=str[i];
      }
      String(const String &);                                   //copy constructor
      void print(){
                for(int i=0;i<size;i++) cout<<content[i];
                cout<<endl;
      }
      ~String(){delete [] content;}                             //destructor
};
String::String(const String &object){                          //copy constructor
      cout<<"copy constructor"<<endl;
      size=object.size;
      content=new char[size];
      for(int i=0;i<size;i++) content[i]=object.content[i];
}
int main(){
      String str("string1");
      str.print();
      String str1=str;                                  //copy constructor is invoked
      String str2(str);                                 //copy constructor is invoked
}
```

Arkadaş (friend) Sınıflar

- Bir sınıf, bir başka sınıfın arkadaşı (friend) olarak tanımlanabilir.
- Bir sınıfın arkadaşı (friend), o sınıfın tüm üyelerine (private, protected, and public) erişim hakkına sahip olur.
- `friend class Line;` satırı, Point sınıfının Line sınıfının arkadaşı olduğunu belirtir.
  - *Bu tanımlama ile Line sınıfı içinde Point nesnelerinin private ve protected üyelerine erişilebilir.*
  - *Bu tanımlama olmazsa, print() fonksiyonu içinde point nesnelerinin private üyelerine doğrudan erişim olduğu için derleyici hata verirdi.*

```
#include <iostream>
using namespace std;
class Point{
          friend class Line;
          int x;
          int y;
public:
          Point(int X=0,int Y=0)
          :x(X),y(Y){}
          void print() const{
               cout<<x<<" "<<y;
          }
};
class Line{
          Point p1;
          Point p2;
public:
          Line(){
          }
          void print() const{
               cout<<p1.x <<","<<p1.y <<"-"
                    <<p2.x <<","<<p2.y;
          }
};
int main(){
          Line line;
          line.print();
          return 0;
}
```

Arkadaş (friend) Sınıflar

- Bir başka örnek

```
class A{
    friend class B;          //Class B is a friend of class A
private:
    int i;
public:
    void func1();
};
class B{
    int j;
public:
    void func2(A &a){cout<<a.i;} //B can access private members of A
};

int main(){
    A objA;
    B objB;
    objB.func2(objA);
    return 0;
}
```

Arkadaş (friend) Fonksiyonlar

- Bir sınıfın arkadaşı olarak, bir global fonksiyon da tanımlanabilir.
- Bir sınıfın arkadaşı (friend) olan fonksiyon, o sınıfın tüm üyelerine (private, protected, and public) erişim hakkına sahip olur.
- `friend void print(Point);` satırı, Point sınıfının `void print(Point p);` fonksiyonunun arkadaşı olduğunu belirtir.
  - *Bu tanımlama ile* `void print(Point p);` *fonksiyonu içinde Point nesnelerinin private ve protected üyelerine erişilebilir.*
  - *Bu tanımlama olmazsa, print() fonksiyonu içinde point nesnelerinin private üyelerine doğrudan erişim olduğu için derleyici hata verirdi.*

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

void print(Point p){
          cout<<"("<<p.x<<","<<p1.y <<")";
}

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

Arkadaş (friend) Fonksiyonlar – Bir örnek

```
// main.cpp
// Friends can access private members of a class.
#include <iostream>
using namespace std;

// Count class definition
class Count
{
     friend void setX( Count &, int ); // friend declaration
public:
     // constructor
    Count()
    : x( 0 ) // initialize x to 0
    {
        // empty body
    } // end constructor Count

    // output x
    void print() const
    {
        cout << x << endl;
    } // end function print
```

Arkadaş (friend) Fonksiyonlar – Bir örnek

```
private:
    int x; // data member
}; // end class Count

// function setX can modify private data of Count
// because setX is declared as a friend of Count (line 9)
void setX( Count &c, int val )
{
    c.x = val; // allowed because setX is a friend of Count
} // end function setX

int main()
{
    Count counter; // create Count object

    cout << "counter.x after instantiation: ";
    counter.print();

    setX( counter, 8 ); // set x using a friend function
    cout << "counter.x after call to setX friend function: ";
    counter.print();
} // end main
```

`Arkadaş (friend)` Fonksiyonlar – Bir örnek

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```

`Arkadaş (friend)` Fonksiyonlar ve Sınıflar

- Arkadaşlık (friendship) özelliği, bilginin saklanması ve nesne tabanlı tasarım yaklaşımlarının değerini yok etmektedir.
- Bu nedenle, zorunlu kalmadıkça kullanımının tercih edilmemesi tavsiye edilir.

Kaynaklar

- T.C. Lethbridge and R. Laganiere, 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.