BM102
Algoritma ve Programlama II

Bileşim, Arkadaş ve Diğer Araçlar

İçerik

1. Bileşim
2. Arkadaş Fonksiyonlar ve Sınıflar
    - 2.1. Arkadaş Fonksiyonlar
    - 2.2. Arkadaş Sınıflar
3. Sabit Nesneler
4. Sabit Üye Fonksiyonlar
5. Dinamik Bellek Kullanımı
6. thisGöstergesi
7. Statik Sınıf Üyeleri
8. Özel Başlık Dosyası

Ödev Soruları

Hedefler

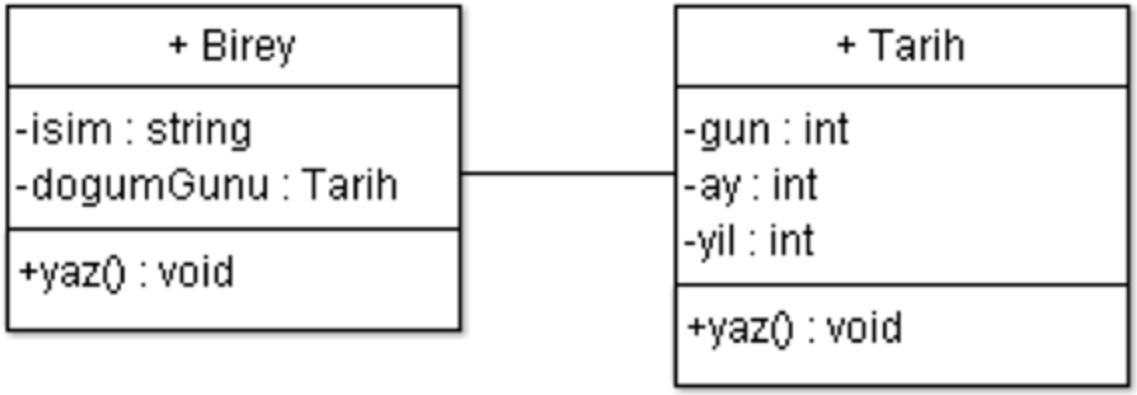
- Bileşim ilişkisi var olan ve UML sınıf diyagramı verilen sınıfları programlama
- Sınıf tanımını dikkate alarak bileşim ilişkisini UML sınıf diyagramında gösterme
- Arkadaş fonksiyonlar ve üye fonksiyonlar arasındaki farkı anlatma
- Arkadaş fonksiyon tanımı yapma
- Arkadaş sınıf tanımı yapma
- Sabit nesne tanımlama
- Sabit nesne ve isim sabitleri arasındaki farkı anlatma
- Sabit üye fonksiyonlar ile üye fonksiyonlar arasındaki farkı anlatma
- Sabit üye fonksiyon tanımlama
- t h i s göstergesinin kullanım amacını anlatma
- Statik sınıf üyeleri tanımlama
- Özel başlık dosyalarının kullanım amacını anlatma
- Sınıf tanımını özel başlık dosyasında yapma

1. Bileşim

- İki sınıfın arasında dört tip ilişki olabilir.
  - Sınıflar birbirlerinden bağımsız tanımlanabilir,
  - Sınıflar **arkadaş (`friend`)** olarak tanımlanabilir,
  - Sınıflar miras yoluyla ilişkilendirilebilir,
  - Sınıflar **bileşim** yoluyla bağlanabilirler.

1. Bileşim...

- *İng. Composition*
- Bir sınıf üyesinin başka bir nesne olması durumudur.
- Bir nesnenin içinde başka bir nesne yer alır.



Birey Sınıf Diyagramı

1. Bileşim…

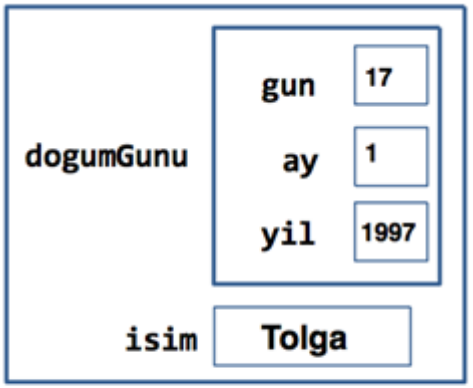
```
#include <iostream>
#include <string>
using namespace std;
class Tarih
{
  int gun,ay,yil;
  public:
     Tarih(int g,int a,int y):gun(g),ay(a),yil(y){}// Yapıcı fonksiyon tanımı
     void yaz();
};
```

1. Bileşim…

```
void Tarih::yaz()                          // Tarihi yazdıran fonksiyon
{
    cout<<gun<<"/"<<ay<<"/"<<yil<<endl;
}
class Birey
{
    string isim;
    Tarih dogumGunu;
    public:
        Birey(string,int,int,int);  // Yapıcı fonksiyon prototipi
        void yaz();
};
void Birey::yaz()                          // Birey nesne bilgisini yazdıran fonksiyon
{
    cout<<isim<<" "; dogumGunu.yaz();
}
Birey::Birey(string s,int g,int a,int y):dogumGunu(g,a,y)// Yapıcı fonksiyon
{
    isim=s;
}
int main()
{
    Birey insan("Tolga",17,1,1997);  // Nesne tanımı
    insan.yaz();
    return 0;
}
```

Çıktı

```
Tolga 17/1/1997
```



insan Nesnesi

2.1. Arkadaş Fonksiyonlar

- *İng. Friend*
- Sınıfın üyesi olmamasına rağmen o sınıfın **private** üyelerine erişme hakkına sahip fonksiyonlardır
- **friend** anahtar kelimesi ile fonksiyon prototipinde belirtilir

2.1. Arkadaş Fonksiyonlar…

```
#include <iostream>
using namespace std;
class Dene
{
    int no;
    public:
        Dene (int n=0):no(n){}        // Yapıcı fonksiyon tanımı
        friend void yaz(Dene&);       // Arkadaş fonksiyon prototipi
};
void yaz(Dene &x)                     // Arkadaş fonksiyon tanımı
{
    cout<<"numara = "<<x.no<<endl;
}
int main()
{
    Dene test1(7),test2(12);
    yaz(test1);                       // Arkadaş fonksiyonun çağırımı
    yaz(test2);
    return 0;
}
```

Çıktı

```
numara = 7
numara = 12
```

2.2. Arkadaş Sınıflar

- Bir sınıf başka bir sınıfın arkadaşı olarak tanımlanırsa tüm **private** üyelerine erişebilme hakkına sahip olur.

2.2. Arkadaş Sınıflar…

```
#include <iostream.h>
using namespace std;
class B;                                  // Ön tanım
class A
{
    int x;
    public:
        A(int _x):x(_x){}                 // Yapıcı fonksiyon tanımı
        friend class B;                   // Arkadaş sınıfı tanımı
};
class B
{
    int y;
    public:
        void fon(A& nsn);
};
```

```
void B::fon(A& nsn)
{
    y=nsn.x;
    cout<<"y="<<y<<endl;
}
int main()
{
    A bir(10);
    B iki;
    iki.fon(bir);
    return 0;
}
```

Çıktı

```
y=10
```

3. Sabit Nesneler

- *İng.* **Constant object**
- *Yapıcı fonksiyon ile ilk atamalar yapıldıktan sonra üyelerin değerini değiştiremediğimiz nesnedir.*

3. Sabit Nesneler…

```
class Eleman
{
    public:
        int no;
        Eleman(int _no):no(_no){}// Varsayılan yapıcı fonksiyon
        ...
};
int main()
{
    const Eleman kisi(123);        // Yapıcı fonksiyon çağırılır
    kisi.no=333;                   // X Sabit nesne olduğu için hata verir!
    return 0;
}
```

4. Sabit Üye Fonksiyonlar

- **const** olarak tanımlanan sabit üye fonksiyonlar, sınıfa ait veri üyelerinin değerini değiştiremez.
- **const** komutu, üye fonksiyonun hem prototipinde hem de tanımlamasında yer alır.
- Sabit üye fonksiyon içerisinden başka bir sabit olmayan ve dolayısı ile veri üyelerinin değerini değiştirebilen üye fonksiyonun çağrılması, derleme zamanında hata üretir.

4. Sabit Üye Fonksiyonlar…

```
#include "Daire.h"
#define PI 3.14
#include <iostream>
using namespace std;
class Daire
{
    int r;
    public:
        Daire(int _r):r(_r){}            // Yapıcı fonksiyon
        int rAl() const {return r;}
        void rAta(int _r){ r = _r;}
        float alan() const;
};
float Daire::alan() const          // Dairenin alanını hesaplayan sabit fonksiyon
{
     return PI*r*r;
}
int main()
{
    Daire d(3);
    float alan = d.alan();
    cout<<"Alan:"<<alan<<endl;
    return 0;
}
```

Çıktı

```
Alan:28.26
```

4. Sabit Üye Fonksiyonlar…

```
#include "Daire.h"
#define PI 3.14
#include <iostream>
using namespace std;
class Daire
{
    int r;
    public:
        Daire(int _r):r(_r){}
        int rAl() const {return r;}
        void rAta(int _r){ r = _r;}
        float alan() const;
        float alan();
};
float Daire::alan() const // Dairenin alanını hesaplayan sabit fonksiyon
{
    cout<<"Sabit alan() uye fonksiyonu"<<endl;
    return PI*r*r;
}
```

```
float Daire::alan()          // Dairenin alanını hesaplayan sabit olmayan fonksiyon
{
    cout<<"Sabit olmayan alan() uye fonksiyonu"<<endl;
    return PI*r*r;
}
int main()
{
    Daire d1(3);
    const Daire d2(3);
    cout<<d1.alan()<<endl;
    cout<<d2.alan()<<endl;
    return 0;
}
```

Çıktı

```
Sabit olmayan alan() uye fonksiyonu
28.26
Sabit alan() uye fonksiyonu
28.26
```

5. Dinamik Bellek Kullanımı

- Programın çalışması sırasında bellekten yer alınıp kullanılmasına **dinamik bellek** kullanımı adı verilir
- Bellekten yer alma işlemi **new** komutuyla, yeri geri verme işlemi **delete** komutu ile gerçekleştirir
- Bellekten alınan isimsiz nesnenin adresini bir göstergeye atayıp üyelerine işaret operatörü (->) ile erişebiliriz.

5. Dinamik Bellek Kullanımı...

```
#include <iostream>
using namespace std;
class Kare
{
    int kenar;
    public:
        void oku();
        int cevre();
};
void Kare::oku()
{
    cout<<"Kenari girin:";
    cin>>kenar;
}
int Kare::cevre()
{
    return 4*kenar;
}
```

```
int main()
{
    Kare* p=new Kare;                  // p tarafından gösterilen isimsiz nesne yaratılır
    p->oku();                          // p kullanarak oku() fonksiyonu çağırılır
    cout<<"Cevre="<<p->cevre()<<endl;  // p kullanarak cevre() fonksiyonu çağırılır
    delete p;
    return 0;
}
```

Çıktı

```
Kenari girin:5
Cevre=20
```

5. Dinamik Bellek Kullanımı...

- Program bitiminde dinamik nesne yokedilir

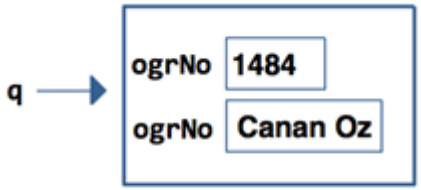
```
delete p;
```

- Dinamik nesne yok edilmez ise, p tarafından gösterilen yer başka değişkenler için kullanılamaz.

5. Dinamik Bellek Kullanımı…

- Bir dinamik nesne tanımı sırasında ilk değerleri atamak için yapıcı fonksiyona parametre yollanabilir.

```
class Kisi
 {
     int ogrNo;
     string isim;
     public:
         Kisi(int _ogrNo,string _isim):ogrNo(_ogrNo),isim(_isim){}
         void yaz();
 };
 void Kisi::yaz()
 {
     //..
 }
 int main()
 {
     Kisi *q =new Kisi(1484, "Canan Oz");     // q'nun gösterdiği nesne için
                                              // yapıcı çağırılır
      ...
     delete q;
     return 0;
 }
```

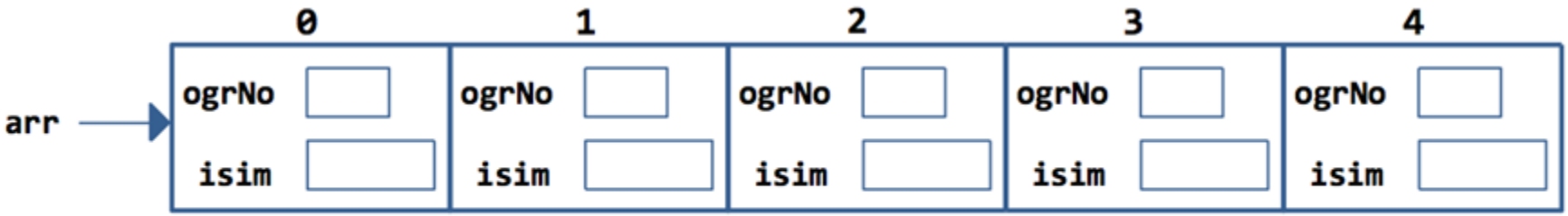


q göstergesi ile gösterilen dinamik nesne

5. Dinamik Bellek Kullanımı…

- C ve C++ dillerinde dizi adları zaten gösterge olarak tanımlandıkları için göstergeleri dizi gibi kullanabiliriz.

```
Kisi *arr=new Kisi[5];
```



arr Dinamik Dizisi

```
arr[2].yaz();
```

//arr göstergesiyle tanımladığımız dizinin 2. elemanının yaz() //fonksiyonuna ulaşır.

```
delete [] arr;
```

//arr dizisi silinir.

5. Dinamik Bellek Kullanımı…

```
#include <iostream>
using namespace std;
class Ders
{
    string ad;
    int ogrSayi;
    int *notlar;
    public:
        Ders();                                  // Yapıcı fonksiyon
        ~Ders (){delete [] notlar;}              // Yıkıcı fonksiyon dinamik diziyi siler
        void oku();
        void yaz();
 };
Ders::Ders()                          // Yapıcı fonksiyon dinamik dizi yaratır ve bilgileri okur
{
    cout<<"Ders adini ve ogrenci sayisini giriniz:";
    cin>>ad>>ogrSayi;
    notlar=new int[ogrSayi];
    cout<<"Ogrenci notlarini giriniz:";
    for (  i=0;i<ogrSayi;i++)
        cin>>notlar[i];
    cout<<endl;
}
```

5. Dinamik Bellek Kullanımı…

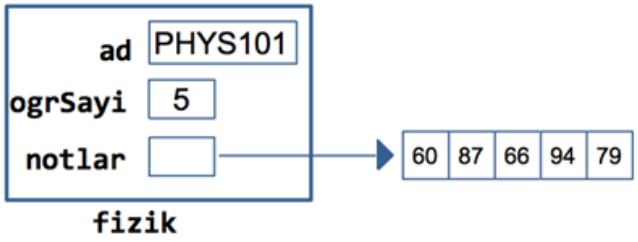
```
void Ders::yaz()                    // Ders bilgilerini yazdıran fonksiyon
{
    cout<<"Ders: "<<ad<<endl;
    cout<<"Ogrenci Notlari:";
    for (int i=0;i<ogrSayi;i++)
        cout<<notlar[i]<<" ";
    cout<<endl;
}
int main()
{
    Ders fizik,kimya;               // fizik ve kimya isimli 2 nesne oluşturulur.
    fizik.yaz();
    kimya.yaz();
    return 0;
}
```

Çıktı

```
Ders adini ve ogrenci sayisini giriniz:PHYS101 5
Ogrenci notlarini giriniz:60 87 66 94 79

Ders adini ve ogrenci sayisini giriniz:CHEM102 7
Ogrenci notlarini giriniz:99 65 45 73 79 43 28

Ders:PHYS101
Ogrenci Notlari:60 87 66 94 79
Ders:CHEM102
Ogrenci Notlari:99 65 45 73 79 43 28
```



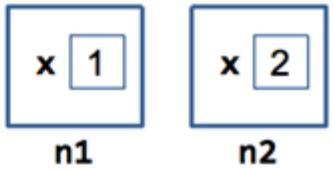
fizik Nesnesi

6. this Göstergesi

- Belleği etkin kullanmak için sınıflarda tanımlanan üye fonksiyonlar o sınıf tipindeki her nesnenin içine tekrar tekrar kopyalanmaz. Üye fonksiyonun tek kopyasını tüm nesneler paylaşır.
- Fonksiyonun hangi nesne için çağırıldığı ise **this** göstergesi tarafından belirlenir.
- Sistemde önceden tanımlanmış **this** göstergesi her zaman fonksiyonu çağıran nesnenin kendisini gösterir.

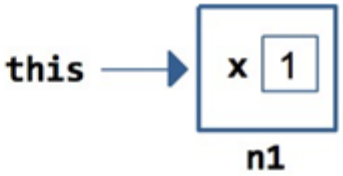
6. this Göstergesi...

```
#include <iostream>
using namespace std;
class A
{
    int x;
    public:
        A(int _x):x(_x){}
        void yaz();
};
void A::yaz()
{
    cout<<"x= "<<x<<endl;
}
int main()
{
    A n1(1),n2(2);
    n1.yaz();
    n2.yaz();
    return 0;
}
```



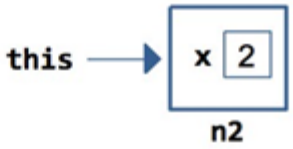
n1 ve n2 Nesneleri

```
n1.yaz();
```



this Göstergesi

```
n2.yaz();
```



this Göstergesi ile Nesneye Erişim

6. this Göstergesi...

- this göstergesi kullanarak ***Fonksiyon Zincirleme*** (İng. Function chaining) yapılabilir.
- Fonksiyon zincirleme, bir fonksiyonun döndürdüğü nesneyi kullanarak başka bir fonksiyon çağırmak anlamına gelir.

6. this Göstergesi...

```
#include <iostream>
using namespace std;
class Sayi
{
    int n;
    public:
        Sayi(int _n=0):n(_n){}
        Sayi& ekle(int);
        void yaz();
};
Sayi& Sayi::ekle(int x)
{
    n+=x;                          // n üyesine x değerini ekler
    return *this;                  // İçinde bulunduğumuz nesneyi döndürür
}
void Sayi::yaz()
{
    cout<<"n= "<<n<<endl;
}
int main()
{
    Sayi s(1);
    s.ekle(2).ekle(3).yaz();    // s nesnesinin n üyesine 2 ve 3 eklenir, yazdırılı
    return 0;
}
```

Çıktı

```
n= 6
```

7. Statik Sınıf Üyeleri

- *İng.* **Static class member**
- Bir sınıf tipindeki tüm nesnelerinortak kullanabildikleri üyelerdir
- **`static`** anahtar kelimesi ile tanımlanır

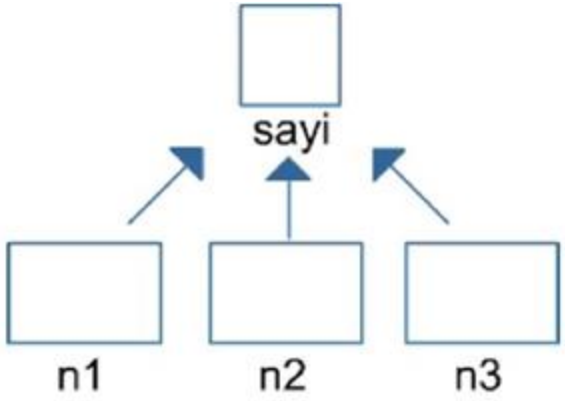
7. Statik Sınıf Üyeleri…

```
#include <iostream>
using namespace std;
class Bir
{
    public:
        static int sayi;     // Statik üyenin tanımı
        Bir()                // Yapıcı fonksiyon çağırıldığında sayi değeri arttırılır
        {
            sayi++;
        }
};
int Bir::sayi = 0;           // Statik üyenin ilk değer ataması
int main()
{
    Bir n1,n2,n3;
    cout<<"Nesnelerin sayisi: "<<n1.sayi<<endl;
    return 0;
}
```

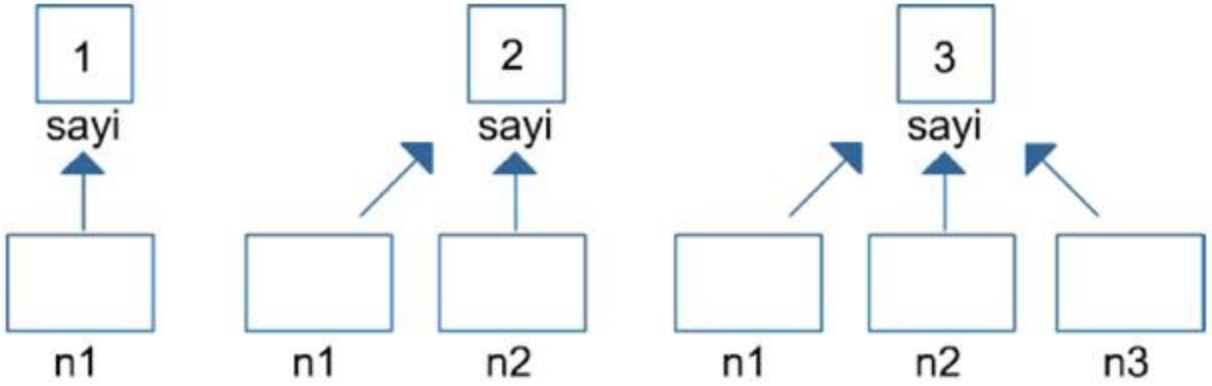
Çıktı

```
Nesnelerin sayisi: 3
```

7. Statik Sınıf Üyeleri…



`n1,n2 ve n3` Nesnelerinin Ortak `sayi` Statik Değişkeni



Yeni Nesne Yaratıldıkça `sayi` Statik Değişkenin Değerinin Güncellenmesi

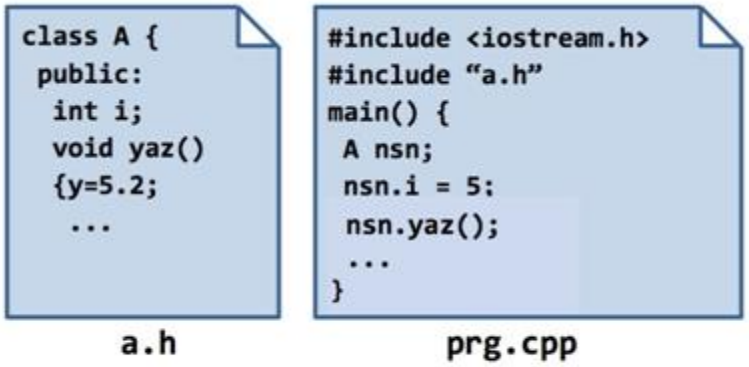
8. Özel Başlık Dosyası

- *İng.* ***Header file***
- Sınıf tanımlarını ve üye fonksiyonlarının prototiplerini içeren dosyadır

8. Özel Başlık Dosyası…

```
class A
{
      public:
         int i;
         void yaz()
         {
             y = 5.2;
             cout<<i<<" "<<y;
         }
      private:
         float y;
         int z;
};
```

```
#include <iostream>
#include "a.h"        // programın a.h dosyasının içeriğini görmesini sağlar.
int main()
{
     A nsn;
     nsn.i = 5;
     nsn.yaz();
     return 0;
}
```



Özel Başlık Dosyası ve Uygulama Programı Dosyası

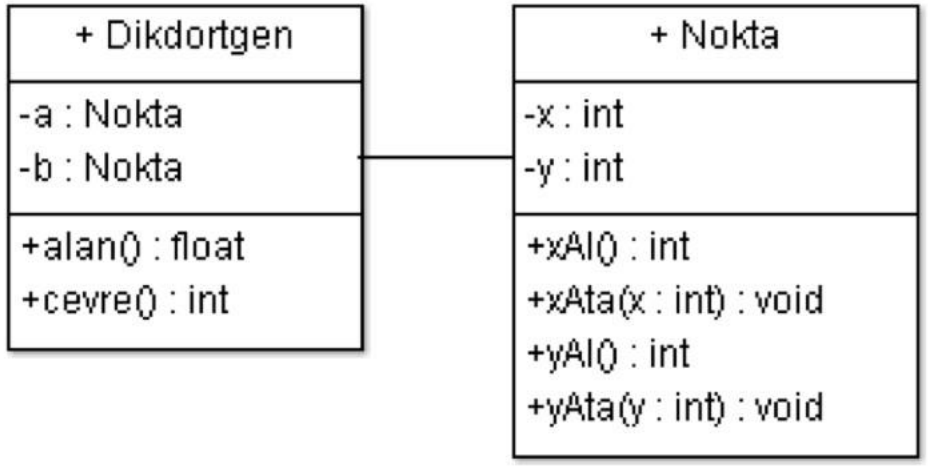
Örnek

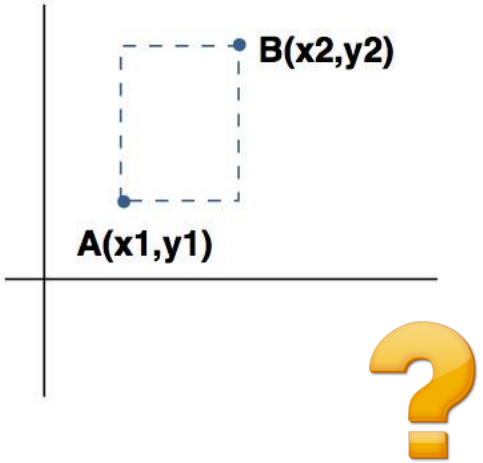
```
2.  #include <iostream>
    using namespace std;
    class Top
    {
        private:
            int val1;
            const int val2;
            int &val3;
            void change(){ cout<<"hello";}
        public:
            Top(int);
            int one() const{ cout<<"bye"; val1*=2;} //-----1
            int two(){return val1;}
    };
    Top::Top (int s):val2(s)
    {
        val1=s;
        val3=s; //-----2
    }
    int main()
    {
        Top t(5);
        const Top z(10);
        t.val1=8; //-----3
        int k=t.two();
        t.change(); //-----4
        cout<<z.two(); //------5
        return 0;
    }
```

1- sabit (const) fonksiyon bir veri üyesini değiştiremez.
2- referans değişkenlerinin ilk değerleri üyelere ilk atama yapılarak atanmalı
3- val1 private olduğu için sınıf dışından erişilemez.
4- change() private fonksiyon olduğundan sınıf dışından erişilemez.
5- sabit (const) nesne sadece sabit (const) fonksiyonları çağırabilir.

Ödev Sorusu

6. Aşağıda şekilde gösterilen ve iki nokta ile tanımlanan dikdörtgenin çevresini ve alanını hesaplayan bir program yazınız. Programda aşağıda sınıf diyagramları verilen `Dikdortgen` ve `Nokta` sınıflarını yaratınız ve bu sınıfları kullanarak örnek çıktıda verildiği gibi çalışan bir program yazınız.





```
// ---------------------- Uygulama.cpp ----------------------
#include <iostream>
#include "Nokta.h"
#include "Dikdortgen.h"
using namespace std;
int main()
{   int x,y;
    cout<<"a noktasi (x,y):";
    cin>>x>>y;
    Nokta a(x,y);
    cout<<"b noktasi (x,y):";
    cin>>x>>y;
    Nokta b(x,y);
    Dikdortgen d(a,b);
    cout<<"Cevre:"<<d.cevre()<<endl;
    cout<<"Alan:"<<d.alan()<<endl;
    return 0;
}
```

Örnek Çıktı

```
a noktasi (x,y): 4 5
b noktasi (x,y): 10 15
Cevre:32
Alan:60
```

Ödev Soruları

6.
```
// ---------- Nokta.h ----------
#ifndef NOKTA_H
#define NOKTA_H
class Nokta
{    int x, y;
     public:
          Nokta(int _x, int _y):x(_x),y(_y){}
          void xAta(int _x){x=_x;}
          void yAta(int _y){y=_y;}
          float xAl(){return x;}
          float yAl(){return y;}
          friend class Dikdortgen;
};
#endif
```

```
// ---------- Dikdortgen.h ----------
#include "Nokta.h"
using namespace std;
class Dikdortgen
{   Nokta a, b;
    public:
        Dikdortgen(Nokta _a, Nokta _b):a(_a),b(_b){}
        int alan();
        int cevre();
};
// ---------- Dikdortgen.cpp ----------
#include "Dikdortgen.h"
#include <iostream>
using namespace std;
int Dikdortgen::alan()
{    return abs(a.x-b.x)*abs(a.y-b.y);
}
int Dikdortgen::cevre()
{    return 2*(abs(a.x-b.x)+abs(a.y-b.y));
}
```