SAKARYA ÜNİVERSİTESİ

Programlamaya Giriş
HAFTA 9
C++ ile Nesneye Dayalı Programlama (NDP)

Prof. Dr. Cemil ÖZ

Doç. Dr. Cüneyt BAYILMIŞ

Dr. Öğretim Üyesi Gülüzar ÇİT

Konu & İçerik

- Nesneye Dayalı Programlama (NDP)
- Nesne (Object)
- UML ve Nesne Örnekleri
- Kapsülleme - Encapsulation
- Mesaj Gönderme
- Sınıf (Class)
- Erişim Belirleyicileri
- Get Ve Set Yöntemleri
- Sınıf ve Nesne Örnekleri
- Çok Şekillilik - Polymorphism
- Kalıtım - Inheritance
- Yapıcı Fonksiyonlar - Constructor
- Yıkıcı Fonksiyon - Destructor
- Çok Dosyalı Programlar
- Fonksiyona Parametre Olarak Nesne Gönderimi
- Static
- Const
- Nesnelerin Bellek Kullanımı
- Dizi Elemanı Olarak Nesne Kullanımı
- Çalışma Soruları
- Kaynaklar

Nesneye Dayalı Programlama (NDP)

- Yapısal teknikte programcı doğrudan probleme odaklanır ve problemin çözümüne ilişkin yöntemleri geliştirir.
- Nesneye dayalı programlama tekniğinde ise temel bileşen **nesne**dir ve programlar nesnelerin birlikte çalışmasından meydana gelir.
- Nesne hem veriyi hem de bu veriyi işleyen fonksiyonları içerir. Programcılar dikkatlerini nesneleri oluşturan sınıfları geliştirmeye yoğunlaştırır.
- Yapısal teknikte bir fonksiyon herhangi bir görevi yerine getirmek için veriye ihtiyaç duyarsa, gerekli veri parametre olarak gönderilir. NDP de ise yerine getirilmesi gereken görev nesne tarafından icra edilir ve fonksiyonlara verilere parametre gönderimi yapılmaksızın erişebilirler.

Nesneye Dayalı Programlama (NDP)…

- **Yapısal Programlama**
  - Sistem büyüdükçe ilişkiler ve bağımlılık daha da karmaşıklaşır.
  - Hata bulma zorlaşır.
  - Program içerisinde değiştirme, ekleme, çıkarma vs. yapmak zorlaşır ve beklenmeyen etkilere neden olabilir…

Nesneye Dayalı Programlama (NDP)…

- **Yapısal Programlama…**
  - Örneğin;

    Ogrenciler tablosunda ogrencinin dogum tarihi iki haneli

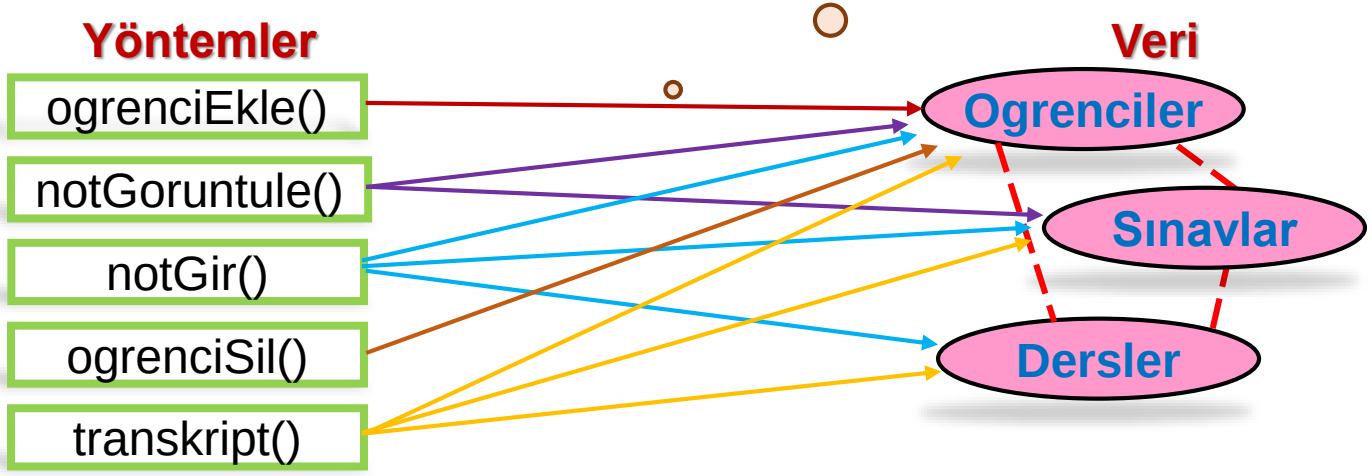
    Bu alanı 4 haneli yapmak istiyoruz…

    Ogrenciler tablosu Sınavlar ve dersler tablolarıyla ilişkili olduğundan beklenmeyen bir sorun çıkabilir…

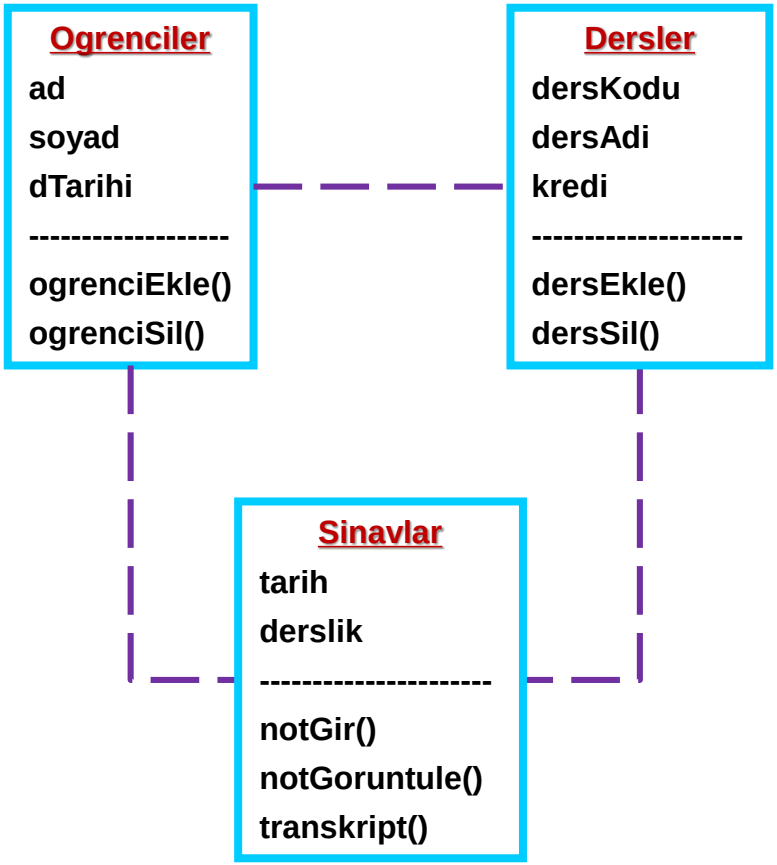
    Tüm yöntemler Ogrenciler tablosunu bir şekilde kullanıyor. Buyöntemlerde de hata olma ihtimali var..

    ogrenciEkle() yontemi kesinlikle sorun cikaracaktır…

Asıl problem, bileşenler arasındaki ilişkilerin net olarak anlaşılamaması



Nesneye Dayalı Programlama (NDP)…



Nesneye Dayalı Programlama (NDP)…

- **Özellikleri / Avantajları**
  - **Encapsulation, data abstraction, inheritance, and polymorphism.**
  - Problemler daha kolay tanımlanıp çözülebilir. Gerçek dünya düşünülerek geliştirilmiştir. Geliştirme süreci daha kolay olur.
  - Bilgi Gizleme (Information Hiding, Data abstraction)
    - Nesne içerisindeki işlemler (nasıl) diğer nesnelerden soyutlanır-sadece ne yapacağını bilirler. Nesne içerisindeki değişiklik diğer nesneleri etkilemez. Dolayısıyla bakım aşaması daha kolay olur.
    - Gereksiz ayrıntılarla uğraşılmaz, probleme odaklanılır (arabanın gitmesi için gaza basmak yeterli). Daha hızlı geliştirme süreci sağlar.
  - Modüler Programlama (Modular Programming)
    - Nesneler birbirinden tamamen bağımsız (veri + fonksiyon) (encapsulation, responsibility driven design)
    - Büyük ve karmaşık bir problem küçük parçalara ayrılarak daha kolay çözülebilir. Geliştirme ve bakım daha kolay olur.
    - Programların geliştirilmesi daha hızlı, geliştirilen bir nesne diğer programlarda da rahatlıkla kullanılabilir. Hata bulma-bakım daha kolaydır (Bisikleti başkasına verdiğimiz zaman da çalışır).
    - Geliştirme sürecinde grup çalışmalarına olanak sağlanır.

Nesneye Dayalı Programlama (NDP)…

- **Özellikleri / Avantajları…**
  - Kodların Tekrar Kullanımı (Code Reuse)
    - Nesneler başka programlara kolaylıkla aktarılabilir. Bakım ve geliştirme zamanı/maliyeti düşer
    - Kalıtım, Çok şekillilik
  - Hata Bulma – Bakım/Onarım (Maintainence)
    - Bileşenler arasındaki ilişkiler açık olduğundan (veri+fonksiyon aynı yapı içerisinde) güncelleme, hata bulma ve bakım daha kolay
  - Tasarım Desenler (Design Patterns)

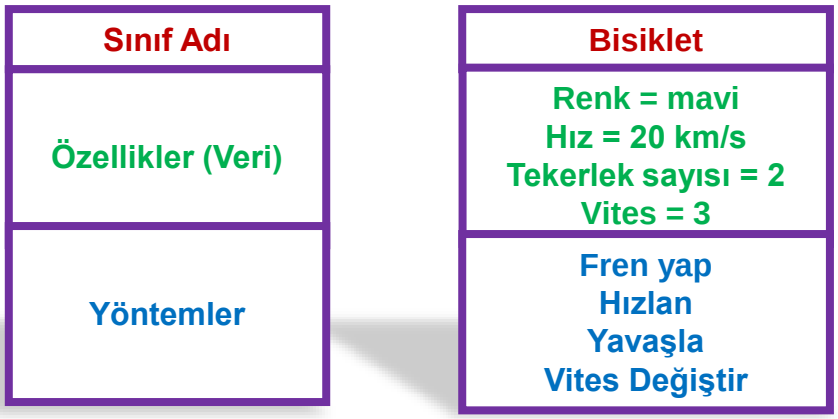
- Günümüzdeki en iyi yaklaşım. Gelecekte ?

Nesneye Dayalı Programlama (NDP)…

- **Nesne (Object)**
  - Gerçek hayatta çevremizde gördüğümüz her şey nesnedir.
    - İnsan, masa, sıra, bisiklet, araba, köpek v.s.
  - Nesneler iki özelliğe sahiptirler. Bunlar;
    - **Durum**
    - **Davranış**
  - **İnsan** için **durumlar**; adı, yaşı, boyu v.s. iken **davranışlar**; öğrenmek, anlamak, uyumak, konuşmak, koşmak v.s.
  - **Bisiklet** için **durumlar**; rengi, o anki vitesi, hızı, tekerlek sayısı, vites sayısı v.s. iken **davranışlar**; fren yapması, hızlanması, yavaşlaması, vites değiştirmesi v.s.
  - Yazılım nesneleri de durum ve davranışlara sahiptir. Nesnelerin durumları **özellik** olarak da adlandırılır ve **değişkenler** ile ifade edilir.
  - Davranışlar ise **fonksiyon** adı verilen ve nesne içerisinde yer alan alt programlar (yöntem) kullanılarak gerçekleştirilir.

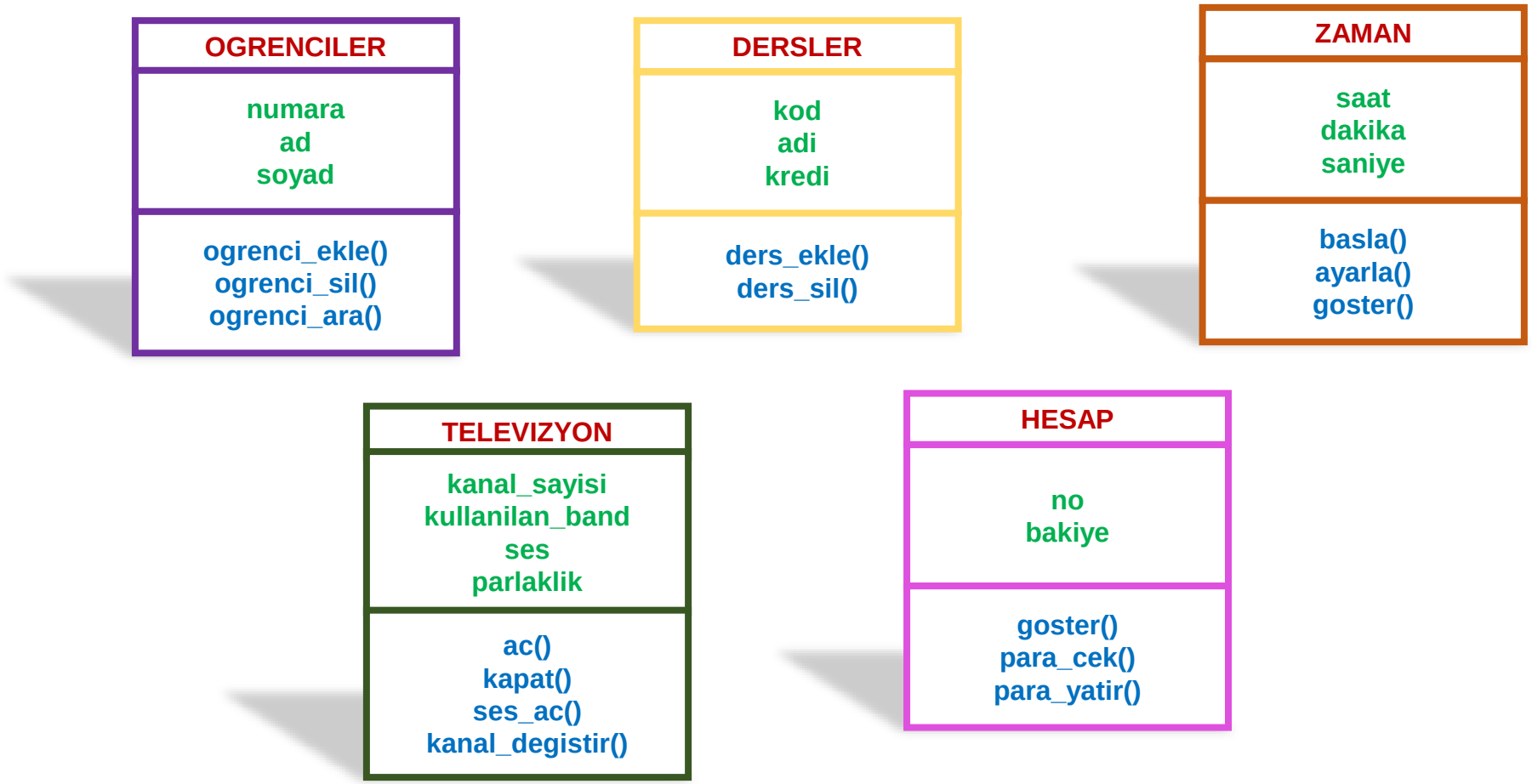
Nesneye Dayalı Programlama (NDP)…

- **Nesne (Object)…**
  - Nesne Tasarlanırken şu sorular sorulmalı:
    - **Özellik** belirlenirken
      - Nesnenin özellikleri ne olmalı (neye sahiptir) ?
    - **Davranış** belirlenirken
      - Nesne ne yapabilir? (Ne yapması istenir ?)
  - **UML (Unified Modelling Language)**
    - Bisiklet nesnesinin (sınıfının) UML ile gösterimi



Nesneye Dayalı Programlama (NDP)…

Nesne Örnekleri



Nesneye Dayalı Programlama (NDP)…

- **Nesneleri Nasıl Oluşturabiliriz?**
  - Nesneleri **sınıf (class)** ile oluştururuz.
- **Sınıf (Class)**
  - Sınıf bir nesnenin planı / tipi gibi düşünülebilir
  - Sınıf nesnenin davranışını ve özelliklerini belirler
  - Her nesnenin bir sınıfı vardır ve bir nesne oluşturulduğunda sınıfın bir örneği (**instance**) oluşturulmuş olur.
  - Bir sınıfa ait her nesne bellekte yer kaplar ve bu yerin adresi tanımlayıcı ya da referans olarak adlandırılan değişkende saklanır.
  - Aynı sınıfı kullanan birden fazla nesne oluşturulabilir ve bu durumda her nesneye farklı referans değişkenleri kullanılarak erişilir.

Nesneye Dayalı Programlama (NDP)…

- **Sınıf İsimlendirme Kuralları**
  - Sınıf isimleri harf, rakam ve '_' ifadelerinden oluşur.
  - İlk harfi rakam olamaz. Türkçe karakterler kullanılamaz.
  - Ayrılmış kelimeler (reserved words) kullanılamaz. if, else, for, final, vs.
  - Büyük harf-küçük harf duyarlılığı vardır (case-sensitive).
  - Sınıf isminin içerdiği veri ile ilgili olması büyük kolaylıklar sağlar.
    - HesapMakinesi
  - Sınıf isimlerini oluşturan kelimelerin baş harfi büyük olmalı.

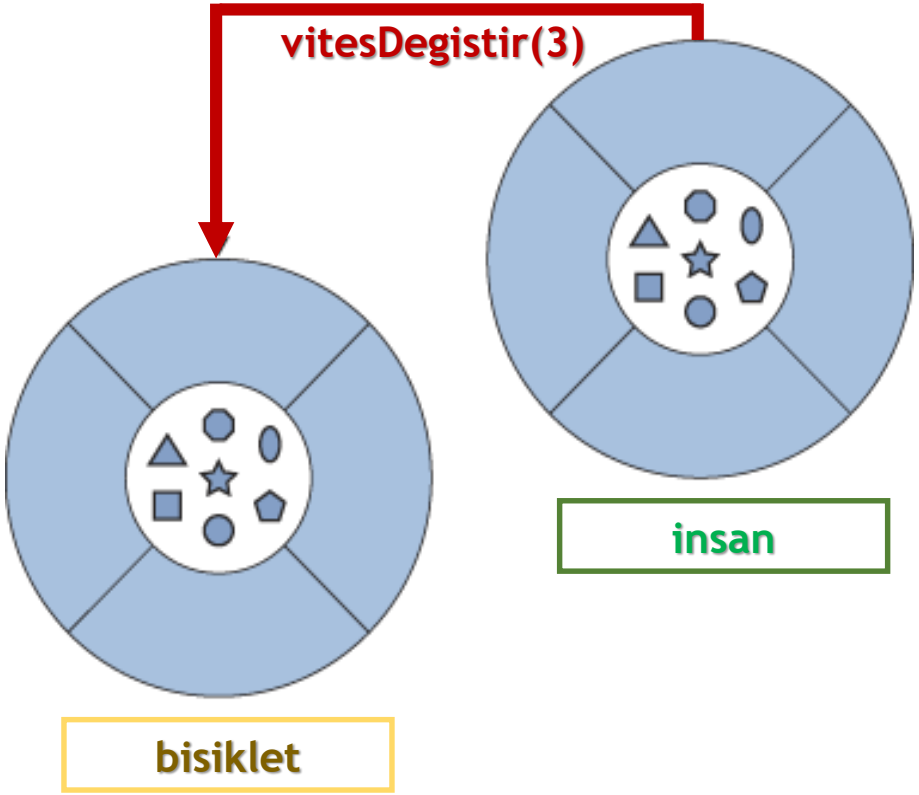
Nesneye Dayalı Programlama (NDP)…

- **Kapsülleme (Encapsulation)**
  - Bir sınıfa ait değişkenleri (verileri) ve yöntemleri bir araya toplayarak birleştirme işlemi
  - Kapsülleme özelliği programcıya iki temel avantaj sağlar
    - **Modülerlik:** Bir nesnenin kodu, diğer nesnelerden tamamen bağımsız olarak geliştirilir. Bununla birlikte geliştirilen bir nesne diğer programlarda da rahatlıkla kullanılabilir.
      - Bisikleti başkasına verdiğimiz zaman da çalışır.
      - string nesneleri her programda kullanılabilir.
    - **Bilgi Gizleme:** Bir nesne, diğer nesnelerin haberleşebilmek amacıyla kullanabileceği **public** arayüze sahiptir. Bununla birlikte diğer nesneler tarafından erişilmesine izin verilmeyen özel değişken ve yöntemleri de içerebilir. Bilgi gizlemenin en büyük avantajlarından biri de nesneye ait ayrıntıların nesne tarafından gizlenerek kullanıcının çözülmesi gereken probleme konsantre olmasının sağlanabilmesidir.
      - Kullanıcı tarafından bisikletin vites mekanizmasının bilinmesine gerek yoktur.

Nesneye Dayalı Programlama (NDP)…

Mesaj Gönderme

- **insan** nesnesinin **bisiklet** nesnesine ait vitesDegistir (3) yöntemini çağırması işlemi



Nesneye Dayalı Programlama (NDP)…

Sınıf (Class)

Sınıf Tanımı

Sınıf Adı: Otomobil

Veri:

yakıt miktarı _____

hız _____

plaka _____

Yöntemler:

Hızlan ⇒ (**Nasıl?** ⇒ gaz pedalına bas)

Yavasla ⇒(**Nasıl?** ⇒ fren pedalına bas)

Sınıfın örneği olan (sınıf tipindeki) nesneler

Örnek 1:

Nesne Adı: araba1

yakıt miktarı : 10 **lt**

hız : 55 **km / s**

plaka : "41 AD 44"

Örnek 2:

Nesne Adı : araba2

yakıt miktarı : 14 **lt**

hız: 0 **km / s**

plaka : "33 NC 3240"

Örnek 3:

Nesne Adı : araba3

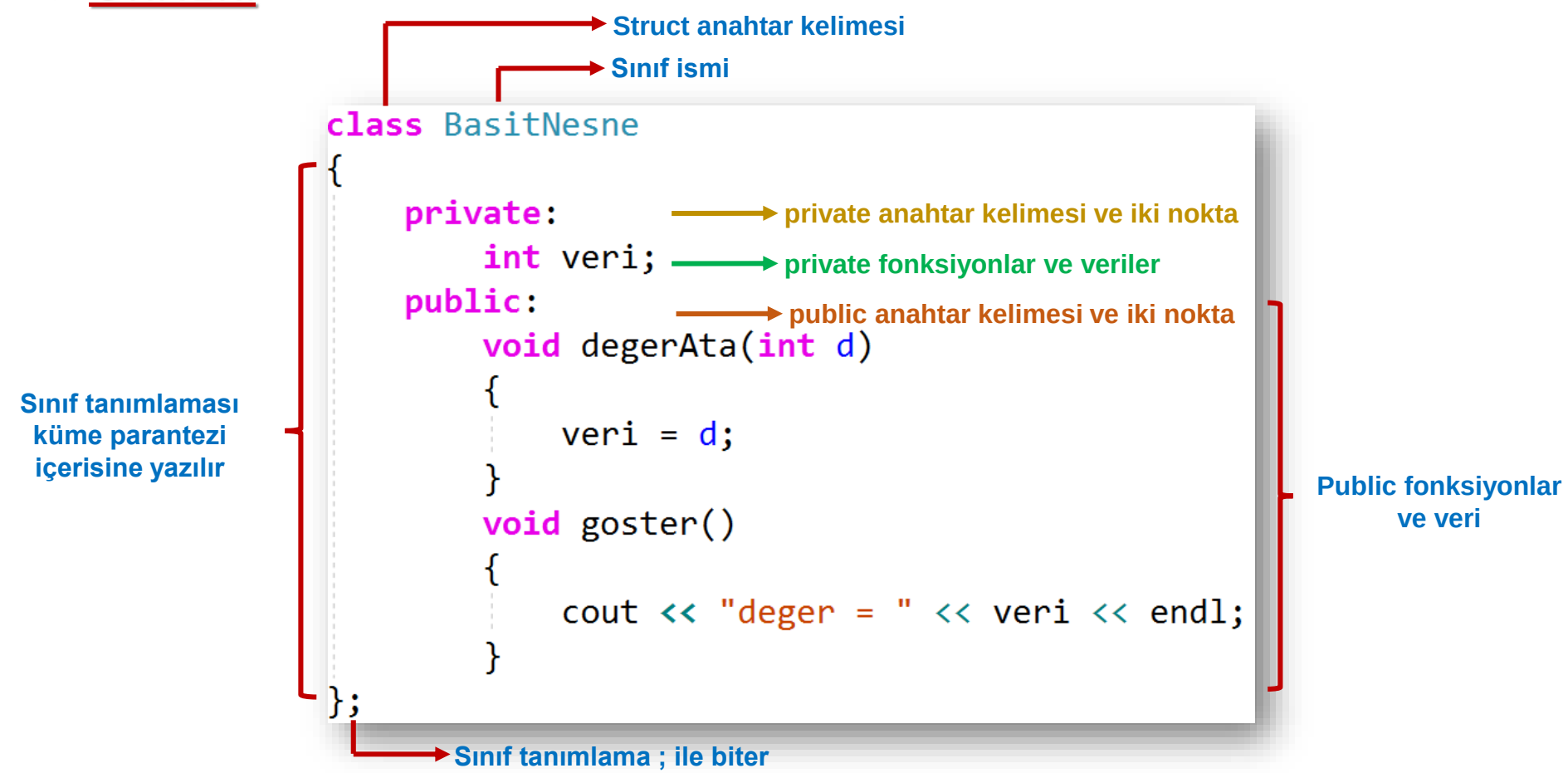
yakıt miktarı: 20 **lt**

hız : 75 **km / s**

plaka: "35 LF 44"

Nesneye Dayalı Programlama (NDP)…

ÖRNEK:



Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:**

```
class Sınıf_Adi
{
    private:
        //özel veriler ve fonksiyonlar
    public:
        // genel veriler ve fonksiyonlar
    protected:
        // korunmuş veriler ve fonksiyonlar
}  nesne_listesi;
```

Nesneye Dayalı Programlama (NDP)…

ÖRNEK:

```
class Arac {
    int max_hiz;
public:
    int model;
    char marka[10];
    void fren_yap(int ivme);
    void hizlan(int ivme);
    int hiz_Oku();
};
```

Sınıf içerisinde prototip olarak tanımlanmış fonksiyonların gövdesi (yaptıkları işlev) program içerisinde şu şekilde tanımlanır.

```
void Arac::fren_yap(int ivme)
{
    //...
}
```

Nesneye Dayalı Programlama (NDP)…

- **Erişim Belirleyicileri**
  - Erişim düzeyi değişken ya da fonksiyona diğer nesne ya d sınıflardan nasıl erişilebileceğini belirler.
    - **public:** Tüm diğer nesnelerden erişilebilir
    - **private:** sadece nesne içerisindeki üyeler tarafından erişilebilir. Private kısmındaki elemanlara public kısmında bulunan üye fonksiyonlar ile erişilebilir.
    - **protected:** Aynı dizindeki (alt dizinler de dahil) bulunan sınıflar tarafından erişilebilir. Bu kısımdaki elemanlara kalıtım yoluyla türetilen alt nesneler tarafından erişilebilir.

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [1]_personel.cpp

```
class Personel
{
private:
    string ad;
    string soyad;
    string adres;
    int sicilNo;
    double maas;    //Yıllık Maaş
```

Üye Değişkenler

```
public:

    void bilgiGoster()
    {
        cout<< "Ad Soyad    : " << ad << " " << soyad << endl
            << "Adres       : " << adres << endl
            << "Sicil No    : " << sicilNo
            << "Yıllık Maaş : " << maas << endl;
    }

    void bilgiGir()
    {
        cout << "Personelin Adını Giriniz:";
        cin >> ad;
        cout << "Personelin Soyadını Giriniz:";
        cin >> soyad;
        cout << "Personelin Adresini Giriniz:";
        cin >> adres;
        cout << "Personelin Sicil Numarasını Giriniz:";
        cin >> sicilNo;
        cout << "Personelin Yıllık Maaşını Giriniz:";
        cin >> maas;

    }
};//Personel Sınıfı Sonu
```

Üye Fonksiyonlar

```
Personel p1;
```

- P1 nesnesi yığın bellekte oluşturulur
- Nesneler kavramsal olarak veri üyeleri ve üye fonksiyonlar içerirken, aslında c++ nesneleri sadece veri içerir.
- Derleyici, sınıf üye fonksiyonlarının sadece bir kopyasını oluşturur ve o kopyayı tüm sınıf nesneleri arasında paylaşır.

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [2]_personel.cpp

```
class Personel
{
private:
    string ad;
    string soyad;
    string adres;
    int sicilNo;
    double maas;     //Yıllık Maaş

public:

    void setAdres(string adr)
    {
        adres = adr;
    }

    string getAdres()
    {
        return adres;
    }
```

```
Personel p1;
p1.setAdres("Sakarya Üniversitesi");
cout << p1.getAdres();
```

- **Veri Doğrulama**
- Sınıfın veri üyelerine doğrudan erişilmesi ve değiştirilmeleri uygun değildir.
- **set** ve **get** fonksiyonları kullanılarak sınıfın veri üyelerine erişilerek veri üyeleri değiştirilebilir veya değerleri döndürülebilir.
- Bu şekilde temiz ve bakımı kolay programlama yapılabilir.

Nesneye Dayalı Programlama (NDP)…

- **Veri Gizleme (Data Hiding)**
  - public
  - private
  - protected

```
class Data {
public:
    int x, y;
    void fonkA();
private:
    int w, y;
    int fonkB();
};

void Data::fonkA()
{
    //...
}

int Data::fonkB()
{
    //....
}
```

```
Data d1, d2;
d1.x = 3;        //geçerli
d1.w = 4;        //geçersiz
d2.fonkA();      //geçerli
d2.fonkB();      //geçersiz
```

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [3]_karmasik.cpp, [4]_karmasik.cpp

```
class Karmasik
{
private:

    double gercel;
    double sanal;

    float radyandanDereceye(float a)
    {
        return a * 180 / PI;
    }

public:
    void bilgiGoster()
    {
        cout << "Sayının Değeri : " << gercel << "+" << sanal << "i" << endl;
    }

    void bilgiGir()
    {
        cout << "Sayının Gerçel Kısmını Giriniz : ";
        cin >> gercel;

        cout << "Sayının Sanal Kısmını Giriniz : ";
        cin >> sanal;
    }

    void kutupsalaCevir()
    {
        cout << "\nSayının Kutupsal Karşılığı : ";
        cout << sqrt(pow(gercel, 2) + pow(sanal, 2)) << "  ";
        cout << radyandanDereceye(atan(sanal / gercel)) << endl;
    }
};
```

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [5]_ogrenci.cpp

```
class Ogrenci {
private:
    string ad;
    string soyad;
    int vize;
    int final;
    double ort;
public:
    void bilgiGir() {
        cout << "Öğrencinin;" << endl;
        cout << "    Adı         : "; cin >> ad;
        cout << "    Soyadı      : "; cin >> soyad;
        cout << "    Vize Notu   : "; cin >> vize;
        cout << "    Final Notu  : "; cin >> final;
    }
    void bilgiGoster() {
        cout << "    Adı Soyadı      : " << ad << " " << soyad << '\t'
             << "    Ortalaması      : " << ortalama() << '\t'
             << "    Başarı Durumu   : " << basari(ortalama()) << endl;
    }

    double ortalama() {
        return ort = vize * 0.5 + final * 0.5;
    }

    string basari(double ort) {
        if (ort >= 50)
            return "Gecti :-)";
        else
            return "Kaldı :-(";
    }

};
```

```
Ogrenci ogr1, ogr2;
ogr1.bilgiGir();
cout << "----------------------" << endl;
ogr2.bilgiGir();
cout << "----------------------" << endl;

cout << "*** Ogrenci Bilgileri ***" << endl;
ogr1.bilgiGoster();
ogr2.bilgiGoster();
```

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [6]_olcu.cpp

```
class mesafe    //sınıf tanımla
{
private:
    int metre;
    float cm;
public:
    void degerAta(int met, float sant)
    {
        metre = met;
        cm = sant;
    }

    void degerAl()
    {
        cout << " metre    : " ; cin >> metre;
        cout << " cm       : " ; cin >> cm;
    }

    void goster()
    {
        cout << metre << "  mt  " << cm << "cm\n" ;
    }
};
```

```
mesafe m1, m2;       // iki obje tanımla
m1.degerAta(5, 20); // m1 objesine değer ata
m2.degerAl();        // m2 objesine değeral

cout << "\n m1 değeri "; m1.goster();
// m1 objesinin değerlerini göster
cout << "\n m2 değeri "; m2.goster();
// m2 objesinin değerlerini göster
```

Nesneye Dayalı Programlama (NDP)…

- **Yapıcılar (Constructor)**
  - Bir **yapıcı fonksiyon** aşağıdaki özelliklere sahiptir:
    - Sınıf ile aynı ada sahiptirler.
    - Sınıf içerisinde hiç olmayabildiği gibi bir ya da çok sayıda da olabilir.
    - Geri dönüş (return) değeri yoktur.
    - Sadece bir defa, nesne oluşturulurken çalıştırılırlar.
  - Bir sınıf içerisinde hiç yapıcı tanımlanmamışsa, derleyici varsayılan bir yapıcıyı otomatik olarak tanımlar. Bu yapıcı nesne için bellekte yer açar. Varsayılan yapıcı temel tipteki veri üyelerine hiç bir başlangıç değeri atamadan sadece nesneyi oluşturur. Diğer sınıfların nesneleri olan veri üyeler için varsayılan yapıcı, her bir veri üyesine ait varsayılan yapıcıyı veri üyelerinin uygun bir şekilde başlatılmalarını garanti etmek üzere kapalı olarak çağırır. string veri üyesinin boş bir string şeklinde başlatılmasının nedeni budur. string sınıfı içerisindeki varsayılan yapıcı bunu sağlar.
  - Yapıcılar genellikle üye değişkenlere ilk değer ataması için kullanılır. Üye değişkenlere ilk değer ataması yapılmadan kullanılırsa mantıksa hatalar oluşabilir.

Nesneye Dayalı Programlama (NDP)…

Yapıcılar (Constructor)…

- Bir sınıf varsayılan yapıcıya iki şekilde sahip olabilir.
  - Derleyici yapıcısı olmayan bir sınıf için kapalı olarak oluşturur. Böyle bir yapıcı sınıfın veri üyelerini başlatmamakla birlikte diğer sınıfların bir nesnesi olan her veri üyelerinin varsayılan yapıcılarını çağırır. Başlatılmamış bir değişken tipik olarak "çöp" değer içerir.
  - Argüman almayan bir yapıcıyı siz tanımlayabilirsiniz. Böyle bir yapıcı diğer sınıfların nesnesi olan veri üyelerinin varsayılan yapıcılarını çağıracak ve sizin tarafınızdan belirtilen ek başlatma işlemlerini gerçekleştirecektir. Eğer argümanı olay yapıcı tanımlarsanız, C++ kapalı olarak o sınıf için varsayılan bir yapıcı oluşturmayacaktır.

```
class Karmasik
{
private:

    double gercel;
    double sanal;

public:
    Karmasik()
    {
        gercel = 0;
        sanal = 0;
        cout << "YAPICI CALIŞTI...\n";
    }

    Karmasik(double g, double s)
    {
        gercel = g;
        sanal  = s;
        cout << "İKİ PARAMETRELİ YAPICI CALIŞTI...\n";
    }
```

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [7]_ogrenci_yapici.cpp

```
class Ogrenci {
private:
    string ad;
    string soyad;
    int vize;
    int final;
    double ort;
public:
    // Yapıcı Fonksiyon nesne oluşturulurken
    // ve sadece 1 defa calistirilir
    Ogrenci()
    {
        ad    = "Şener";
        soyad = "Şen";
        vize  = 55;
        final = 100;
        cout << "Parametresiz Yapici Calisti ... \n";
    }
    Ogrenci(string a, string s, int v, int f)
    {
        ad = a;
        soyad = s;
        vize = v;
        final = f;
        cout << "4 Parametreli Yapici Calisti ... \n";
    }
```

```
Ogrenci ogr1;
Ogrenci ogr2("Cem", "Demir", 40,50);

cout << "*** Ogrenci Bilgileri ***" << endl;
ogr1.bilgiGoster();
ogr2.bilgiGoster();

ogr1.bilgiGir();
ogr1.bilgiGoster();
```

Nesneye Dayalı Programlama (NDP)…

- **Yıkıcılar (Destructor)**
  - Bir **yıkıcı fonksiyon** aşağıdaki özelliklere sahiptir:
    - Sınıf ile aynı ada sahiptirler. Solunda ~ isareti vardır
    - Sınıf içerisinde hiç olmayabilir.
    - Geri dönüş (return) değeri yoktur. Parametre gönderilmez.
    - Sadece bir defa, nesne yok edilirken otomatik olarak çalıştırılır.
    - Sadece bir yok edici fonksiyon tanımlanabilir.
  - Bir sınıf içerisinde hiç yıkıcı fonksiyon tanımlanmamışsa, derleyici varsayılan bir yıkıcıyı otomatik olarak tanımlar. Composition ve kalıtım için özel görevleri vardır.
  - Dinamik ayrılan bellek bölgesini boşaltır (string değişken yok edilirken içerisindeki yıkıcı heap bölgesinde ayrılan yeri boşaltır ⇒ delete []buffer;), dosya veya başka sistem kaynaklarının kapatılması/bırakılması için kodlar yazılmasını sağlar

Nesneye Dayalı Programlama (NDP)…

Yıkıcılar (Destructor)

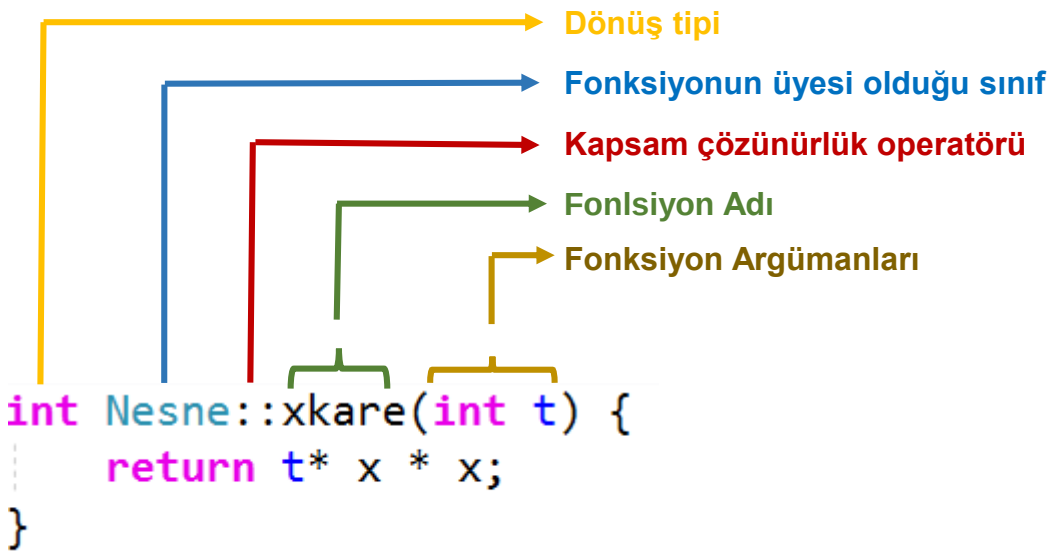
- Bir **yıkıcı fonksiyon** aşağıdaki özelliklere sahiptir:
  - Sınıf ile aynı ada sahiptirler. Solunda ~ isareti vardır
  - Sınıf içerisinde hiç olmayabilir.
  - Geri dönüş (return) değeri yoktur. Parametre gönderilmez.
  - Sadece bir defa, nesne yok edilirken otomatik olarak çalıştırılır.
  - Sadece bir yok edici fonksiyon tanımlanabilir.

```
class Karmasik
{
private:

    double gercel;
    double sanal;

public:
    Karmasik() : gercel(0.0), sanal(0.0)
    {
        gercel = 0.0;
        sanal = 0.;
        cout << "YAPICI CALIŞTI...\n";
    }

    Karmasik(double g, double s)
    {
        gercel = g;
        sanal = s;
        cout << "İKİ PARAMETRELİ YAPICI CALIŞTI...\n";
    }

    ~Karmasik()
    {
        cout << "YIKICI CALIŞTI...\n";
    }
```

- Bir sınıf içerisinde hiç yıkıcı fonksiyon tanımlanmamışsa, derleyici varsayılan bir yıkıcıyı otomatik olarak tanımlar. Composition ve kalıtım için özel görevleri vardır.
- Dinamik ayrılan bellek bölgesini boşaltır (string değişken yok edilirken içerisindeki yıkıcı heap bölgesinde ayrılan yeri boşaltır ⇒ delete []buffer;), dosya veya başka sistem kaynaklarının kapatılması/bırakılması için kodlar yazılmasını sağlar

Nesneye Dayalı Programlama (NDP)…

- **Fonksiyonlar…**
  - Fonksiyonlar / Metotlar iki türlü tanımlanabilir
    - class içerisinde (**inline**)
    - class dışında



Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [8]_olcu.cpp

```
class mesafe    //sınıf tanımla
{
private:
    int metre;
    float cm;
public:
    mesafe() :metre(0), cm(0.0)
    {   }

    mesafe(int met, float sant) : metre(met), cm(sant)
    {   }

    void degerAl()
    {
        cout << "\n metre gir     : ";    cin >> metre;
        cout << " cm gir        : ";     cin >> cm;
    }

    void goster()
    {
        cout << metre << "  metre  " << cm << " cm dir";
    }

    void toplauzun(mesafe, mesafe);  // prototip
};
```

```
void mesafe::toplauzun(mesafe  m2, mesafe m3)
{
    cm = m2.cm + m3.cm;

    metre = 0;

    if (cm >= 100.0)
    {
        cm -= 100.0;
        metre++;
    }

    metre += m2.metre + m3.metre;
}
```

```
mesafe mesafe1, mesafe3;
mesafe mesafe2(5, 4.3);

mesafe1.degerAl();
mesafe3.toplauzun(mesafe1, mesafe2);

cout << " \n mesafe1= "; mesafe1.goster();
cout << "\n mesafe2= "; mesafe2.goster();
cout << "\n mesafe3= "; mesafe3.goster();
cout << endl;
```

Nesneye Dayalı Programlama (NDP)…

Çok Dosyalı Programlar

- Kodların Tekrar Kullanımı
- Interface Implemetation
- Sınıf tanımı artık **başlık dosyası** (header file - main içermeyen dosya) içerisindedir ve bu dosyada tanımlanan sınıfı yeniden kullanmak isteyen her programa bu başlığı dahil edebiliriz.
- Sınıfa ait üye fonksiyonlar, ilgili başlık dosyası içerisinde yazılabileceği gibi, başlık dosyası ile aynı isimli ayrı bir kaynak dosyasında da yazılabilir.
- Birden fazla **include** edilen dosyalar içerisindeki sınıflar arasında aynı isme sahip olanlar olabileceği için çakışmadan dolayı hata oluşur. Çözüm:

```
#ifndef SınıfAdı_h
#define SınıfAdı_h
   // Sınıf kodları
#endif
```

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [9]_karmasik.cpp, KarmasikSayi.h, KarmasikSayi.cpp

```
#ifndef KARMASIKSAYI_H
#define KARMASIKSAYI_H

class KarmasikSayi
{
public:
    KarmasikSayi();
    KarmasikSayi(double, double);
    ~KarmasikSayi();

    double getGercel();     // const;
    double getSanal();      // const;

    void setGercel(double);
    void setSanal(double);

    void kartezyenGoster();
    void bilgiGir();

    void karmasikTopla(KarmasikSayi);
    KarmasikSayi karmasikTopla(KarmasikSayi, KarmasikSayi);
private:
    double gercel;
    double sanal;
};

#endif
```

```
#include <iostream>

#include "KarmasikSayi.h"

using namespace std;

int main()
{
    KarmasikSayi sayi1(3, 2);
    KarmasikSayi sayi2(6, 8);
    KarmasikSayi sayi3;

    system("pause");

    return 0;
}
```

Nesneye Dayalı Programlama (NDP)…

Fonksiyona Parametre Olarak Nesne Gönderme

- ÖRNEK: ⇒ [9]_karmasik.cpp

```
KarmasikSayi sayi1(3, 2);
KarmasikSayi sayi2(6, 8);
KarmasikSayi sayi3;

sayi1.karmasikTopla(sayi2);

sayi1.kartezyenGoster();
sayi2.kartezyenGoster();

sayi3 = sayi3.karmasikTopla(sayi1, sayi2);
sayi3.kartezyenGoster();


sayi1.bilgiGir();
sayi2.bilgiGir();
sayi1.kartezyenGoster();
```

Soru: Klavyeden ESC tuşuna basılıncaya kadar girilen karmaşık sayıların toplamını bulan programı yazınız. Karmaşık sayı işlemleri (bilgi girişi, toplama v.s.) için Karmasik sınıfı tipinde nesne kullanınız.

Nesneye Dayalı Programlama (NDP)…

➢Dizi Elemanı Olarak Nesne Kullanımı

➢ÖRNEK: ⇒ [10]_karmasik_dizi.cpp

```
#include "KarmasikSayi.h"

using namespace std;

int main()
{
    KarmasikSayi sayilar[5];

    sayilar[0].kartezyenGoster();

    for (int i = 0; i < 5; i++)
        sayilar[i].bilgiGir();


    for (int i = 0; i < 5; i++)
        sayilar[i].kartezyenGoster();

    system("pause");
    return 0;
}
```

Soru: KarmasikSayi sınıfı içerisine karmaşık sayının kutupsal koordinatlarıyla ilgili r ve teta değerlerini döndürmek üzere **rDondur** ve **tetaDondur** adlı iki üye fonksiyon ekleyiniz.

En fazla 50 eleman girileceğini düşünerek klavyeden ESC tuşuna basılıncaya kadar girilen karmaşık sayıları dizide saklayan ve girilen karmaşık sayıların r değerlerinin aritmetik ortalamasını bulan programı geliştiriniz…

Nesneye Dayalı Programlama (NDP)…

- **static**
  - **static** veriye tüm nesneler erişebilir.
  - Bir sınıftan oluşturulan tüm nesnelerin ortak bilgi paylaşımı yapmaları gerektiğinde üye değişken **static** olarak tanım kullanılır.
  - **static** fonksiyonlar nesne oluşturulmadan da kullanılabilirler.

Soru: Donusturucu adında bir sınıf tanımlayınız. Bu sınıf içerisinde dışarıdan aldığı parametreyi dönüştüren; inch2cm, cm2inch, radyan2derece, derece2radyan, F2C, C2F, dolar2TL, euro2TL … üye fonksiyonlar tanımlayınız. Tanımladığınız sınıftaki üye fonksiyonları (nesne oluşturmadan) kullanan ana programı yazınız.

Nesneye Dayalı Programlama (NDP)…

ÖRNEK: ⇒ [11]_static.cpp

```
class Araba
{
private:
    int renk;
    int model;
    double yakitMiktari;
public:
    static int sayi;

    Araba()
    {
        cout << ++sayi << endl;
    }

    static int getSayi()
    {
        return sayi;
    }

    ~Araba()
    {
        cout << sayi-- << endl;
    }
};
```

```
// statik üyeler burada başlatılabilir....
int Araba::sayi = 0;

using namespace std;

int main()
{
    cout << "ilk:" << Araba::getSayi();

    Araba a1, a2, a3;

    cout << "Aktif araba sayisi " << a1.getSayi() << endl;
    cout << "Aktif araba sayisi " << a2.getSayi() << endl;
    cout << "Aktif araba sayisi " << a3.getSayi() << endl;
```

Nesneye Dayalı Programlama (NDP)…

- **const**
  - Bir nesnenin değiştirilememesi gerektiğinde kullanılır. Neden?
    - Yetkiler ne kadar ayrıntılı belirlenebilirse yazılımlar o oranda kaliteli olur. Kodların sadece ihtiyaç duyulduğu kadarına erişilebilmesi çok önemlidir. Bu sayede hata önleme ve hatalardan kaçınma kolaylaşır.
  - **const** olarak tanımlanan nesneler içerisindeki üye fonksiyonların kullanılabilmesi için onların da **const** olması ve üye değişkenleri değiştirmemesi gerekir.
  - **const** fonksiyon üye değişkenleri değiştiremez.
  - Yapıcı ve yıkıcı fonksiyonlar **const** olamazlar.
  - **const** bir üye fonksiyon yine **const** olan bir üye fonksiyonu kullanabilir.
  - **const** bir üye değişkeni başlatmak için yapıcı içerisinde atama işareti kullanılamaz. Bunun yerine:

```
Olcu (int ft, float in) : metre(ft), cm(in)
```

  - **const** olarak oluşturulmayan bir nesne **const** olan fonksiyonları kullanabilir

Nesneye Dayalı Programlama (NDP)…

- **const…**
  - **!!!** Sınıf içerisindeki üye değişkenleri değiştirmeyen fonksiyonların **const** olarak tanımlanması olası hataları önleme açısından önemlidir.
    - ÖRNEK: Öğrenci Sınıfı
      - Öğrenciler modülü için **const ogrenci** nesnesi oluşturulabilir. Bu durumda öğrencilerin değişiklik yapması söz konusu değildir.
      - İdari personel modülünde ise nesne değişiklik yapılabilmesi için normal olarak tanımlanabilir

Nesneye Dayalı Programlama (NDP)…

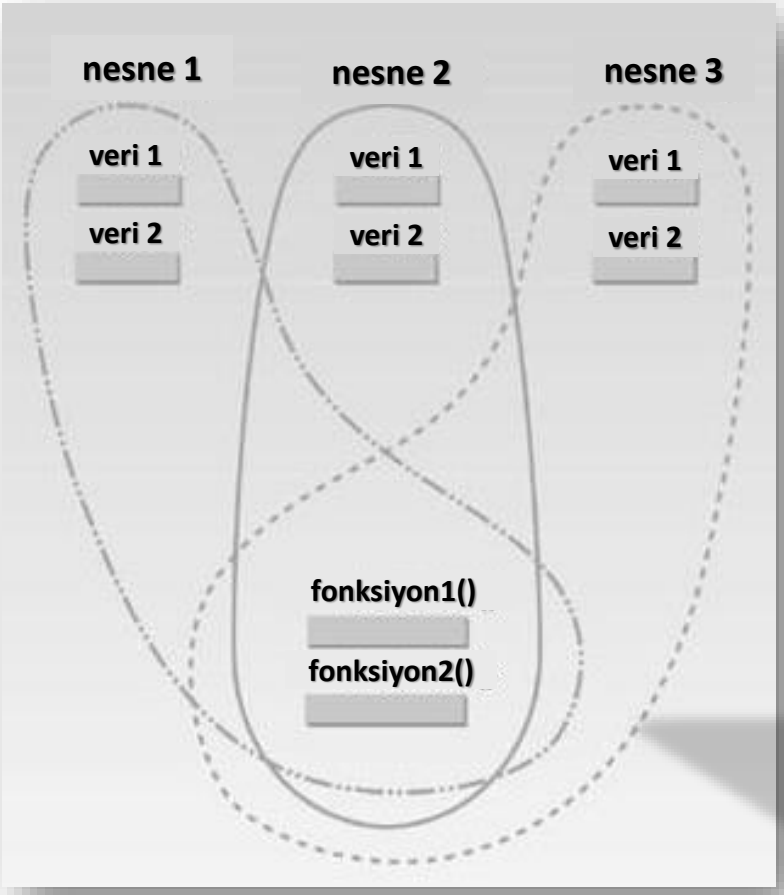
➢ **ÖRNEK:** ⇒ [12]_ogrenci_const.cpp

```
const Ogrenci o1("Ayse", "Yilmaz");
//o1.bilgiGir();//Hata, sadece const fonksiyonlar kullanılabilir
o1.bilgiListele(); //const olan fonksiyon çağrılır
```

```
class Ogrenci
{
private:
    string ad;
    string soyad;                // const string soyad;
public:                          // Kurucu Fonksiyon
    Ogrenci(string a, string s) : ad(a), soyad(s)
    {                            //ad=a; soyad=s;
    }

    void bilgiGir()              //const olmayan nesne kullanabilir
    {
        getline(cin, ad);
        getline(cin, soyad);
    }

    void bilgiListele() const  //const olan nesne bunu kullanır
    {
        cout << ad << "\t" << soyad;
    }

    void bilgiListele()          //const olmayan nesne bunu kullanır
    {
        cout << ad << "\tconst olmayan\t" << soyad;
    }
};
```

Nesneye Dayalı Programlama (NDP)…

Nesnelerin Bellek Kullanımı



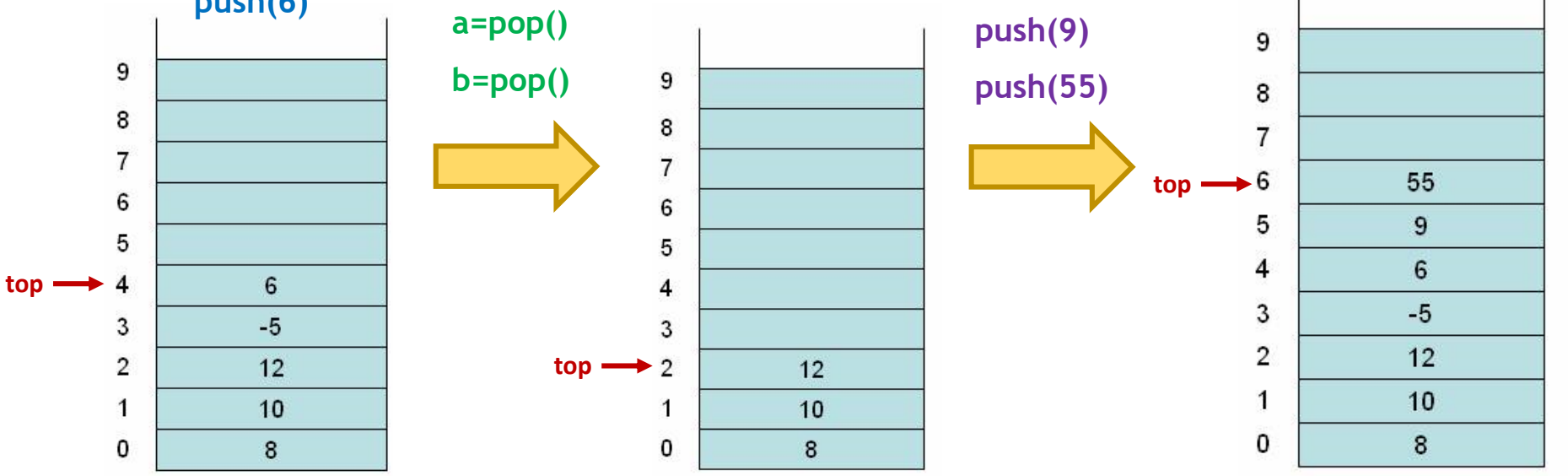
Araştırm Sorusu:

- Belleğin bölümleri nelerdir? (code, stack, heap, static v.s.)?
- Nesneye dayalı programlamada bellek organizasyonu nasıldır? (Örneğin; oluşturulan bir nesne belleğin hangi bölümündedir? Yerel değişken, global değişken statik üye v.s. hangi bölümlerdedir?)

Nesneye Dayalı Programlama (NDP)…

Yığın (Stack)

push(8)
push(10)
push(12)
push(-5)
push(6)



a=pop()
b=pop()

push(-5)
push(6)
push(9)
push(55)

Nesneye Dayalı Programlama (NDP)…

➢ **ÖRNEK:** ⇒ [13]_yigin.cpp

```
class Stack
{
private:
    enum { MAX = 10 };
    int st[MAX];
    int top;
public:
    Stack()
    {
        top = -1;
    }
    void push(int var)
    {
        st[++top] = var;
    }
    int pop()
    {
        return st[top--];
    }
};
```

```
Stack s1;

s1.push(11);                              // 11'i ekle
s1.push(22);                              // 22'yi ekle

cout << "1: " << s1.pop() << endl;   //22'yi çıkart
cout << "2: " << s1.pop() << endl;   //11'i çıkart

s1.push(33);
s1.push(44);
s1.push(55);
s1.push(66);

cout << "3: " << s1.pop() << endl;   //66
cout << "4: " << s1.pop() << endl;   //55
cout << "5: " << s1.pop() << endl;   //44
cout << "6: " << s1.pop() << endl;   //33
```

Çalışma Soruları

- **AsalSayi** sınıfı tanımlayınız. Bu sınıf içerisinde :
  - **asalSayiGir** fonksiyonu **sayi** üye değişkenine klavyeden değer girilmesini sağlamalı
  - **asalmi** fonksiyonu **sayi** üye değişkeninin asal olup olmadığı bilgisini geri döndürmeli .
  - **enYakinAsal** fonksiyonu parametre olarak aldığı sayıya en yakın büyük asal sayıyı geri döndürmeli
- Karakter katarı işlemlerinde kullanılacak **KarakterKatari** adlı sınıfı tanımlayınız. Bu sınıf içerisinde karakter katarını tutacak katar üye değişkenini tanımlayınız. Üye fonksiyonlar :
  - **get** ve **set** fonksiyonları
  - Klavyeden girilen karakter katarını üye değikene alacak olan **katarOku** fonksiyonu
  - Katar üye değişkenindeki bilgiyi tamamen büyüğe çevirecek **buyugeCevir** fonksiyonu
  - Katar üye değişkenindeki bilgiyi tamamen küçük çevirecek **kucugeCevir** fonksiyonu
  - Büyükse küçük, küçükse büyük (Ali->aLİ gibi) yapan **buyukKucukCevir** fonksiyonu
  - Kelimelerin ilk harfini büyüğe çeviren (boşluk karakteri kontrol edilmeli…) **ilkHarfBuyuk** fonksiyonu
  - Girilen karakter katarının uzunluğunu hazır fonksiyon kullanmadan döndüren **uzunluk** fonksiyonunu yazınız
  - Katar üye değişkenindeki bilgiyi ekrana tertten yazdıran **terstenYazdir** fonksiyonu

KAYNAKLAR

- Deitel, C++ How To Program, Prentice Hall
- Horstmann, C., Budd,T., Big C++, Jhon Wiley&Sons, Inc.
- Robert Lafore, Object Oriented Programming in C++, Macmillan Computer Publishing
- Prof. Dr. Celal ÇEKEN, Programlamaya Giriş Ders Notları
- Prof. Dr. Cemil ÖZ, Programlamaya Giriş Ders Notları