

# 315 programlama dilleri

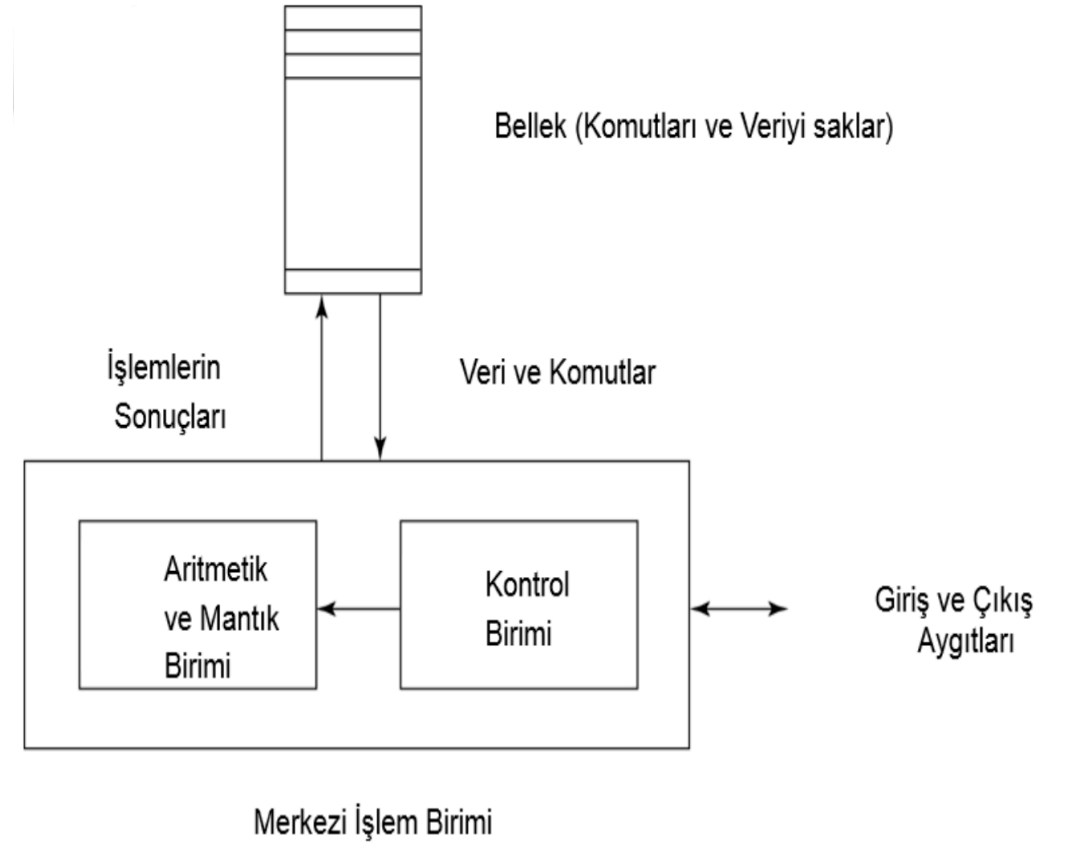
Yrd. Doç. Dr. Ahmet Arif AYDIN

Names, Bindings, Scopes

# Von Neumann Mimarisi

- Imperative diller Von nuemann mimarisi tabanlı olarak gelişmiştir
- Von nuemann mimarisinin iki temel bileşeni bulunmaktadır:

1. **Memory (Bellek)** : Komut ve verileri saklar
2. **Processor (işlemci)**: Belleğin içeriğinin değiştirilmesi için gerekli işlemleri sağlar



<https://onedrive.live.com/view.aspx?resid=6045030A20A1F6F7!321&thint=file%2cpptx&app=PowerPoint&authkey=!ADP2re3cv57lOns>

# Names

Program içerisindeki

bir varlığı tanımlamak için kullanılan

ve dilin karakterlerinden oluşan stringe name(isim) denir.

Alt programlar

parametreler

program yapıları

değişkenler

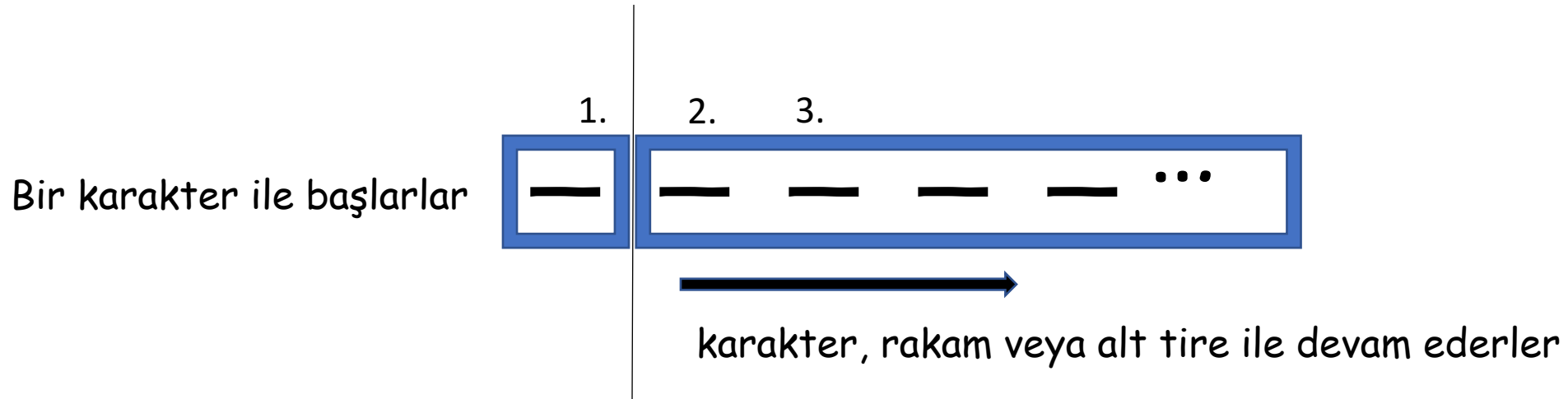
tanımlanırken **isim** kullanılır

# Names: isim uzunluğu

Programlama dili	İsim uzunluğu
Fortran 1 den önceki diller	1 karakter
Fortran 1	6 karakter
Fortran 77	6 karakter
Fortran 95+	31 karakter
C99	Limit yok (ilk 63 karakter önemli)
Java , C# , C++	Limit yok
Python	Limit yok (79 karakteri geçmemek önemli)

# Names: isim oluşturma kuralları

Genel olarak programlama dillerindeki isimler bir karakterle başlayıp karakter , rakam veya alt tire ile devam ederler.



# Names: BÜYÜK-küçük harf duyarlılığı

Özellikle C tabanlı dillerde

- büyük-küçük harf duyarlılığı bulunmaktadır ( case sensitive)
- isimler de büyük - küçük harf duyarlılığı vardır
- Büyük ve küçük karakterler birbirinden farklı olarak kabul edilir.

$A \neq a$

# Names: isim oluşturma kuralları

Underscore ( \_ ) kullanımı

Küçük büyük karışık harf kullanımı



programlama  
dilin stili

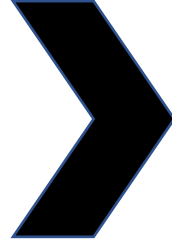
- Fortran 90
  - Sum of Salaries = SumofSalaries
- Java
  - CamelCase

# Names: BÜYÜK-küçük harf duyarlılığı

sınıf

Sınıf

SINIF



Birbirine benzemelerine rağmen

tamamen farklıdırlar

Case sensitive diller kimisine göre okunabilirlik (readability) e zarar vermektedir. Çünkü benzer görünen isimler birbirinden farklıdır.



# Names: BÜYÜK-küçük harf duyarlılığı

C , C++ , Java isimleri büyük-küçük harf duyarlıdır.

C

değişken isimleri küçük harf olmak zorundadır

Java

camel case kullanılmaktadır

```
int değer= Integer.parseInt("1234");
```

Parseint veya parseint olamaz...

C#

```
string title = "war and peace";
```

```
TextInfo textInfo = new CultureInfo("en-US", false).TextInfo;  
title = textInfo.ToTitleCase(title); //War And Peace
```

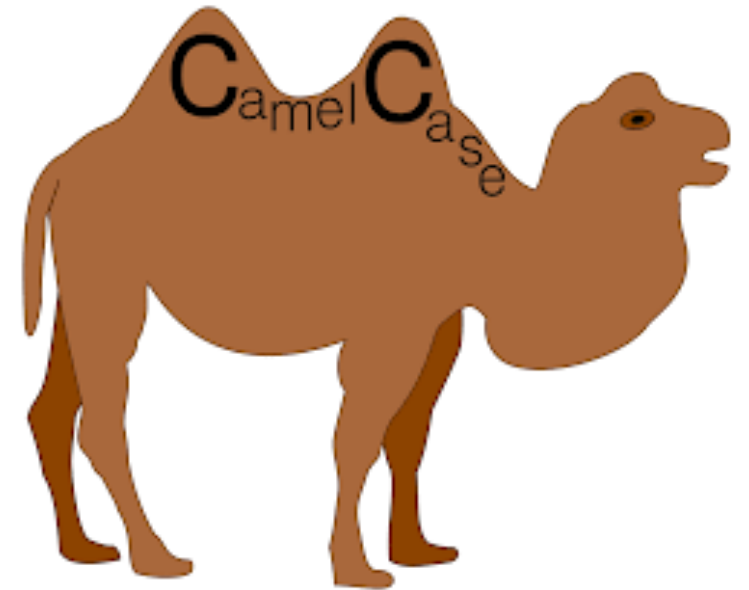
# Names: Java toCamelCase

```
static String toCamelCase(String s){
    String[] parts = s.split(" ");
    String camelCaseString = "";
    for (String part : parts){
        if(part!=null && part.trim().length()>0)
            camelCaseString = camelCaseString + toProperCase(part);
        else
            camelCaseString=camelCaseString+part+" ";
    }
    return camelCaseString;
}

static String toProperCase(String s) {
    String temp=s.trim();
    String spaces="";
    if(temp.length()!=s.length())
    {
        int startCharIndex=s.charAt(temp.indexOf(0));
        spaces=s.substring(0,startCharIndex);
    }
    temp=temp.substring(0, 1).toUpperCase() +
    spaces+temp.substring(1).toLowerCase()+" ";
    return temp;
}

public static void main(String[] args) {
    String string="HI tHiS is  SomE Statement";
    System.out.println(toCamelCase(string));
}
```

JavaCamelCase



<https://stackoverflow.com/questions/17078347/convert-a-string-to-modified-camel-case-in-java-or-title-case-as-is-otherwise-ca>

# Names: BÜYÜK-küçük harf duyarlılığı

- Windows case sensitive değildir. Change Directory komutu
  - Cd
  - CD
  - cD
  - Cd

Olarak yazılabilir ve istenilen işlem gerçekleştirilir.

# Names: BÜYÜK-küçük harf duyarlılığı

- Linux büyük küçük harfe duyarlıdır
  - Listelemek için **sadece**: ls

```
ls [OPTION]... [FILE]...
```

# Names: özel kelimeler

Özel kelimeler programları daha okunur hale getirir.

Programlama dillerinde ki özel kelimeler iki kategori altında incelenir:

1. Programlama diline ait özel **anahtar kelimeler** (keywords)
2. Programlama diline ait **ayrılmış kelimeler** (reserved words)

# Anahtar kelimeler (keywords)

Anahtar kelimeler kullanım biçimine göre ve konsepte göre değişmektedir.

- Fortan' da anahtar kelimeler kullanılmaktadır.
  - Integer elma (elma değişkenini tanımlamıştır)
  - Integer = 4 (değişken)

Özel kelimelerin tekrar kullanımı karışıklık oluşturabilir.

# Names: reserved words

Programlama diline ait ayrılmış kelimeler (reserved words) isim olarak kullanılamazlar.  
Ayrılmış kelimeler anahtar kelimelerden daha kullanışlıdır.

JAVA	Abstract, assert, boolean, break, case, catch continue, .....
C++	Const, double, float, int, short, struct, unsigned, break, continue, else ....

<http://www.jwrider.com/riderist/java/javaidrs.htm>

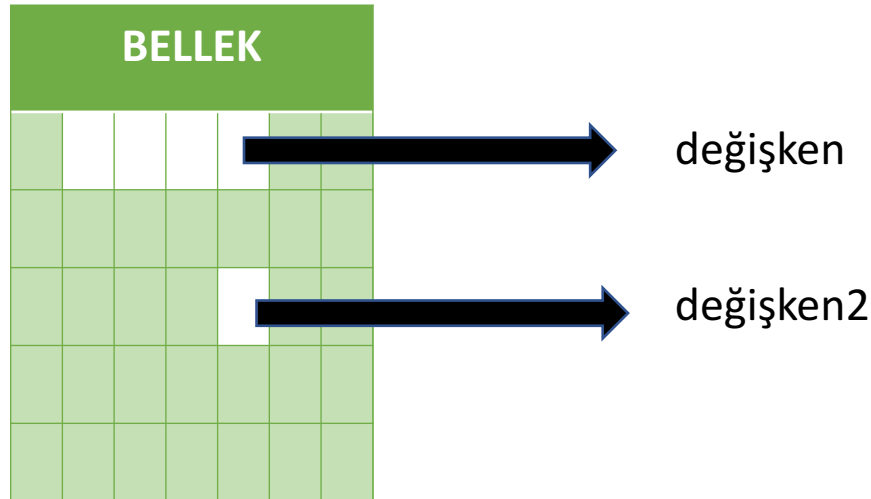
<http://ktutorials.com/cplusplus/keywords-and-reserved-words-in-cplusplus/>

Ayrılmış kelimelerin çok fazla olması da programcıların isim seçmelerini zorlaştırır.

- COBOL 300 adet reserved keywords

# Variables: Değişkenler

- Makine dilinden assembly diline ve yüksek dillere geçiş ile bellekteki hücrelerin fiziksel adresini kullanmak yerine değişkenler kullanılmaya başlanmıştır.
- Programlarda kullanılan değişkenler bellekte yer tutan hafıza hücresi veya hücreleri için bir abstractions'dır.
- Değişkenler daha okunabilir, yazılabilir ve bakımı yapılabilir programlar yazmaya olanak sağlar.





# Variables: Örnek

## PHP

- Bütün değişken isimleri \$ karakteri ile başlamak zorundadır.

```
<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
?>
```

# Variables: Örnek

Ruby : Sınıf değişkeni @@ sembolleri ile başlar, Global değişkenler \$ işareti ile başlar, örnek değişkenler (instance variables) @ işareti ile

```
class Customer
  @@no_of_customers = 0
  def initialize(id, name, addr)
    @cust_id = id
    @cust_name = name
    @cust_addr = addr
  end
  def display_details()
    puts "Customer id #@cust_id"
    puts "Customer name #@cust_name"
    puts "Customer address #@cust_addr"
  end
  def total_no_of_customers()
    @@no_of_customers += 1
    puts "Total number of customers: #@no_of_customers"
  end
end

# Create Objects
cust1 = Customer.new("1", "John", "Wisdom Apartments, Ludhiya")
cust2 = Customer.new("2", "Poul", "New Empire road, Khandala")

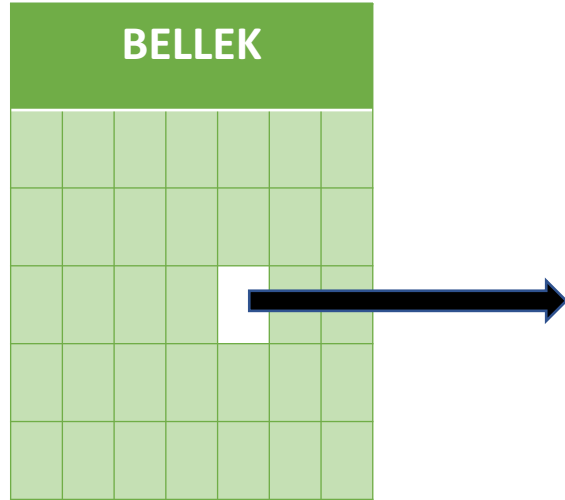
# Call Methods
cust1.total_no_of_customers()
cust2.total_no_of_customers()
```

```
$global_variable = 10
class Class1
  def print_global
    puts "Global variable in Class1 is #{$global_variable}"
  end
end
class Class2
  def print_global
    puts "Global variable in Class2 is #{$global_variable}"
  end
end

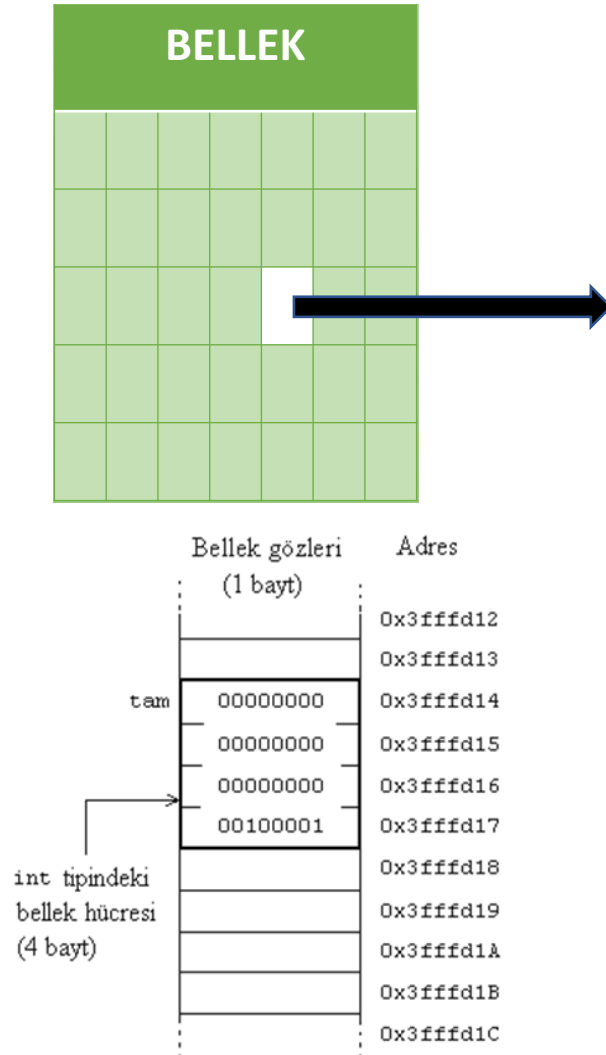
class1obj = Class1.new
class1obj.print_global
class2obj = Class2.new
class2obj.print_global
```

# Değişken Nitelikleri

- İsim (name) Bir değişkeni tanımlamak için kullanılır

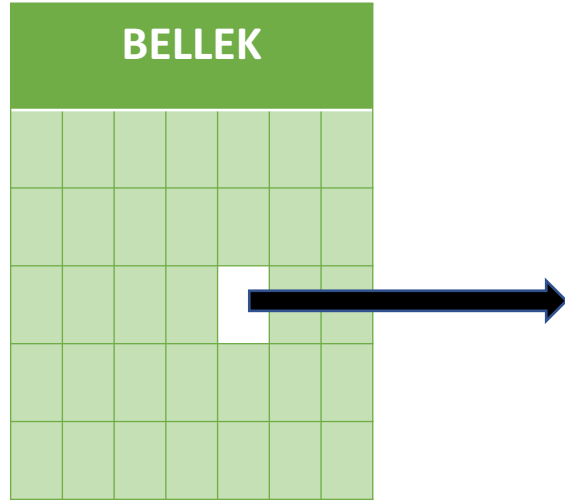


# Değişken Nitelikleri



- Adres (address): Makinanın bellek adresidir.
  - Farklı alt programlarda kullanılan aynı isimli değişken için 'sonuç' farklı bellek adresleri kullanılır. Bir değişken program içerisinde farklı zamanlarda farklı adresleri kullanabilir.
- Adres I-value olarak da isimlendirilir.
- Birden fazla değişkenin aynı bellek adresini kullanılması gerekiyorsa takma ad (**alias**) yardımıyla gerçekleştirilir.
- Alias kullanımı okunabilirliği zorlaştırmaktadır.

# Değişken Nitelikleri



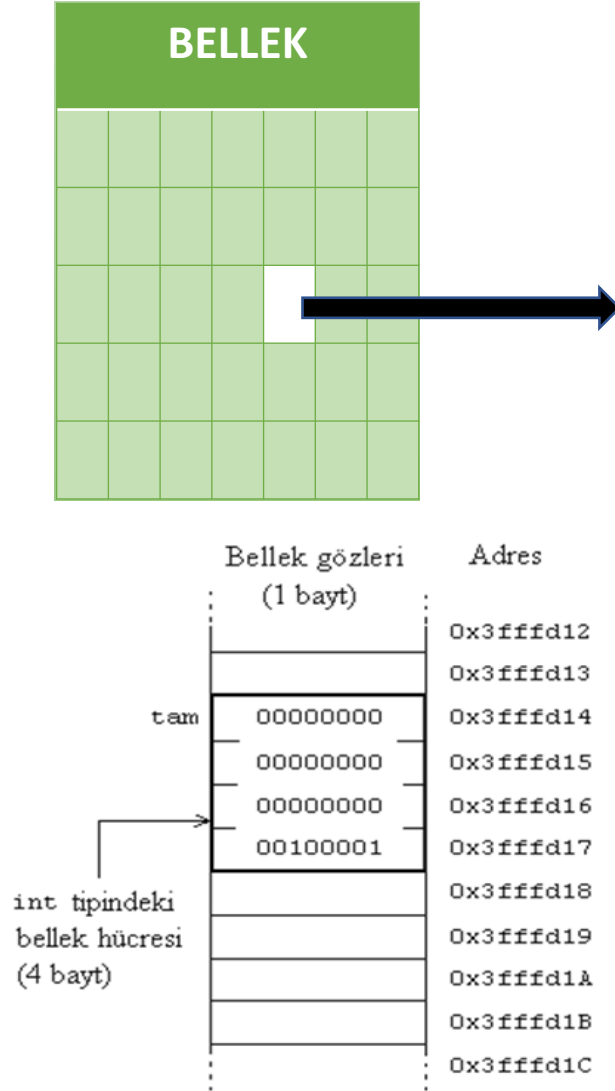
Tip (type)

- Değişkenin kaydedebileceği değerlerin aralığını tanımlar.
- Java int
  - -2147483648 ile 2147483647 arasındadır.

# Değişken Nitelikleri

## Değer(value)

- Değişkene verilen hafıza hücresinde tutulan değerdir.
- Bellek hücresi 1 byte (8 bit) uzunluğunda olduğundan değişkenlerin tipine göre birden fazla hücre bir değişkenin tanımlanmasında kullanılır.
- Değişkenin sakladığı degere r-value da denir



l-value     $A = 5$     r-value

# Değişken Nitelikleri

Yaşam süresi (lifetime)

- Bir bellek adresinin bir değişken ile ilişkili kaldığı zaman lifetime olarak tanımlanır.

Kapsam (scope)

- Bir program içerisinde ki bir değişkenin geçerli olduğu alan olarak tanımlanır.

# Bindings: Bağlanma

Bir program içerisinde bulunan  
bir varlık ile niteliğin ilişkilendirilmesine **binding** (bağlanma) denir.

Bağlanmanın gerçekleştiği zaman  
bağlanma zamanı (binding time) olarak tanımlanır ve  
programlamada çok önemli olan bir kavramdır.



# Bindings: Bağlanma Zamanı

Bağlanmanın gerçekleştiği zaman dilimleri aşağıda verilmiştir:

Language design time

language implementation time

compile time

load time

link time

run time

# Bindings: Örnek

int sayı;

sayı = sayı \* 3

- Dil tasarımı zamanında
  - sayı değişkeninin alabileceği muhtemel tipler
  - = ve \* sembollerin alabileceği anlamlar bağlanır
- Derleme tasarım zamanı
  - 3 sayısının anlamı bağlanır
  - sayı nın alabileceği muhtemel değerler bağlanır
- Derleme zamanı
  - sayı değişkeninin tipi bağlanması yapılır
  - \* işaretinin manası bağlanır
- Çalışma Zamanı
  - sayı değişkeninin işlem sonucuna bağlanır

# Niteliklerin Değişkenlere Bağlanması

## Statik Binding

- Bir bağlanma program çalıştırılmadan önce tanımlanmışsa ve programın çalıştırılması (execution) boyunca değişmemiş ise statik bağlanma olarak ifade edilir.

## Dynamic Binding

- Eğer bağlanma program çalıştırıldıktan sonra değişiyorsa dinamik bağlama olarak adlandırılır.

# Tip Baęlanması (Type Bindings)

Bir ok programlama dilinde bir deęişken tanımlanmadan nce tipinin baęlanması gerekmektedir.

- Statik Tip Baęlanması
- Dinamik Tip Baęlanması
- Tip ıkarımı (Type Inference)

# Static Type Bindings (explicit)

## 1- Açık-Belirtilen Tanımlama (Explicit Decleration)

- Program içerisinde değişkenlerin ve tiplerinin açık bir biçimde yazıldığı tanımlamadır
  - `Int değer; (Java , C , C++)`.

# Static Type Bindings(implicit)

## 2- Örtülü Tanımlama (Implicit Decleration)

- Değişken tipleri varsayılan kurallara (default conventions) göre belirlenir.
- **BASIC**
  - Son karakteri \$ ile biten tanımlayıcılar karakter tipi ile bağlanırlar
- **Fortran**
  - Bir tanımlayıcı I,J,K,L,M ,N veya i,j,k,l,m,n ile başlarsa o tanımlayıcı integer olarak kabul edilir veya real olarak tanımlanır.

# Type inference

## 3-Tip Çıkarımı (Type Inference)

Örtülü olarak (implicit) ve kullanılan değere göre değişkenin tip çıkarımı yapılır.

*C#*

```
var toplam = 0 ;  
var total  = 0.3 ;  
var isim   = "İstanbul" ;
```

int

Float

String

Statik değişkenler

# Dynamic Type Bindings

- Değişken tipi
  - bir tanımlama ifadesiyle veya isminin yazılması ile belirlenmez
  - atama işlemi sırasında ataması yapılan değere göre bağlanır
- Bu durum programlama esnekliği sağlar.
  - Javascript
  - PHP



# Dynamic Type Bindings

- Javascript
  - `Liste = [3.5, 2.3]` liste değişkeni 2 elemandan oluşan bir dizi olarak bağlanır
  - `Liste = 55` önceki tanımına bakılmadan liste sayısal bir değişkene dönüşür.
- Ruby
  - Değişkenlerin tipi tanımlanmaz
  - Bütün veriler ve değişkenler nesne olarak kabul edilir
  - Herhangi bir değişken herhangi bir nesneyi tanımlamada kullanılabilir

# Dynamic Type Bindings: Dezavantajlar

- Programın güvenilirliğini azaltır.
- Çünkü derleyicinin hata bulma özelliği kullanılmamış olur.
- Derleyici (Compiler) yerine Yorumlayıcılar (interpreter) kullanılır.
- Çalışma anında tip bağlanması belirlenmesi zaman alan maliyetli bir iştir.
- Herhangi bir değişkenin herhangi bir veri tipine bağlanması sağlar
  - Atama işlemlerindeki hatalı atamalar tespit edilemez
  - `i=5`
  - `y=[1,2,3]`
  - `i=y;` işlemi için hata vermez!

# Type inference

ML ( 1998 )

- Fun daireninalanı (r) = 3.14159 \*r\*r;
- Fun çarpma (k) = 10\*k;
- Fun kare(x)= x \* x ;

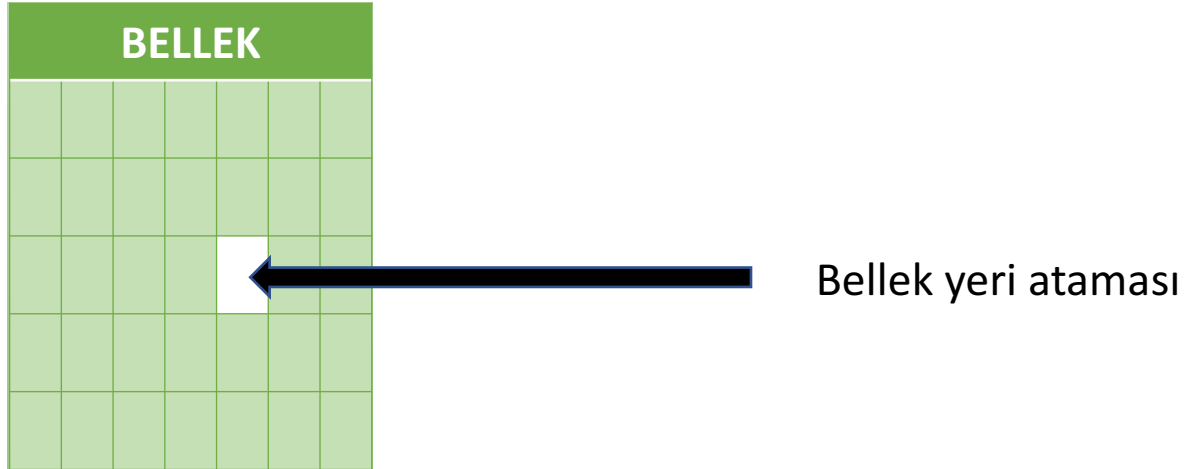
kare(3.4); HATA

Fun kare(x): real = x \* x ;

ML de bulunan varsayılan sayısal deger integer olduğundan real deger istenen fonksiyonların tanımlanması gerekmektedir.

# Storage Bindings (Bellek Bağlama)

Bir değişkenin kullanacağı bellek hücre veya hücrelerinin boş ve kullanıma açık olan bellekten alınması gerekmektedir. Bu işleme **allocation** denmektedir.



# Storage Bindings (Bellek Bağlama)

Bir değişkenin kullandığı bellek hücre veya hücrelerinin boş ve kullanıma açık hale getirilmesine **deallocation** denmektedir.

BELLEK						

Bellek yeri bırakılması

# Yaşam Süresi (Lifetime)

Yaşam süresi (lifetime)

Bir bellek hücre veya hücrelerinin bir değişkene başlanmasından ayrılmasına kadar geçen süre lifetime olarak tanımlanır

BELLEK						

# Bellek Bağlama: Statik Değişkenler

Bir programın çalışmasından önce belirlenen bellek hücrelerine bağlanan ve programın çalışması süresince aynı bellek hücrelerini kullanan değişkenlere statik değişkenler denir.

## Avantajlar

- Global olarak tanımlanan değişkenler statik olması uygun görülmüştür
- Statik değişkenlerin hafıza alanları direkt olarak tanımlıdır.
- Hızlı ve verimli çalışmayı arttırırlar
  - Bellek alanın allocation ve deallocation işlemleri zaman alır.

# Bellek Bağlama: Statik Değişkenler

## Dezavantajlar

- Hafıza alanının seçimindeki esneklik azalır.
- Sadece statik değişken tanımlayan dillerde recursive işlem yapamazsınız
- Bellek değişkenler arasında paylaşılamaz
  - iki alt program farklı zamanlarda çalıştırılmış olmalarına rağmen sadece kendi değişkenlerine tanımlanan alanı kullanır.



# Bellek Bağlama: Yığıt Dinamik Değişkenler

## Yığıt - Dinamik Değişkenler (Stack-Dynamic variables)

- Bellek bağlanma işlemi değişken tanımlamalarının yapıldığı kod kısmının çalışma anında tanımlanır (elaboration).
- Değişkenlerin tanımlandığı blok aktif olduğu müddetçe değişkenler yaşar.
- C++ ve Java metotları içerisinde tanımlanan değişkenler bu kategoridedir.

# Bellek Bağlama: Yığıt Dinamik Değişkenler

- Avantaj
  - Recursive yapıların oluşturulmasını sağlarlar
  - Az bellek harcanmasını sağlarlar
- Dezavantaj
  - Bellekten yer ayrılması (allocation) ve bellek biriminin serbest bırakılması (deallocation ) zaman alır.
  - Alt programlar geçmişini unuttur
  - Dolaylı adresleme yapılıdır

# Bellek Bağlama

## Açık yığın dinamik değişkenler (Explicit heap-dynamic variables)

- Programcılar tarafından tanımlanan istenildiğinde bellek üzerinde oluşturulup sonrasında istenildiğinde bellekten silinebilen değişkenlerdir.
- Tip bağlanması derleme zamanında , bellek yer ayrılması ise programın çalışma anında tanımlanır.

# Bellek Bağlama

## Açık yığın dinamik değişkenler (Explicit heap-dynamic variables)

```
int *ornekpointer; // pointer oluştur
...
ornekpointer = new int; // yığın dinamik değişken oluştur
...
delete ornekpointer; // yığın dinamik değişkeni siler
```

- Dinamik bellek yönetimi sağlar
- Yönetimi zor olduğundan güvenilir değildir

# Bellek Bağlama

## Örtülü Dinamik Değişkenler (Implicit heap-dynamic variables)

- Sadece değer aldıklarında belleğe bağlanırlar
- Tip ve bellek başlanması her değer aldığı anda değişirler

# Bellek Bağlama

## Örtülü Dinamik Değişkenler (Implicit heap-dynamic variables)

- Javascript
  - Yükseklik = [74, 84, 86, 90, 71];
  - Önce tanımlı olup olmamasından bağımsız olarak `suan` yükseklik değişkeni 5 elemanlı bir dizi olarak bağlanmıştır
- Avantaj : Esnek kod yazımını sağlarlar
- Dezavantaj
  - Dinamik özellikler çalışma anında izlendiğinden performans kaybına yol açarlar
  - Derleyicinin hata yakalama özelliğini kullanılamaz

# Scopes

Bir değişkenin bir program içerisinde  
erişilebilir ve görünür olduğu alana `scope` (kapsam) denir.

# Scopes: static scope

- Statik scope kavramı ALGOL 60 ile ortaya çıkmıştır.
- Değişkenin kapsamı statik olarak tanımlanabilmektedir
- Program çalışmadan önce tanımlanır.



# Scopes: static scope

- Statik kapsamlı diller alt programların iç içe tanımlanmasına imkan sağlar
  - Ada, JavaScript, Common LISP, Scheme, Fortran 2003+, F#, and Python
- C tabanlı diller alt programların iç içe tanımlanmasına izin vermez. Bu dillerde statik scope sınıf tanımlamaları ve blok yardımıyla sağlanır.

# Scopes: static scope

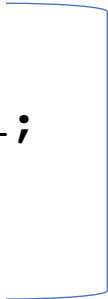
```
function anaprogram()  
{  
    function altprogram1()  
    {  
        var a = 5;  
        altprogram2();  
    }  
  
    function altprogram2()  
    {  
        var b = a;  
    }  
  
    var a = 3;  
    altprogram1();  
}
```

1. altprogram2 içerisinde a değişkeninin tanımı aranır
2. Eğer altprogram2 içerisinde a değişkeninin tanımı bulunmazsa bir üst (ancestor fonksiyona) bakılır.
3. Bir üst seviyedeki anaprogram() içerisinde bulunmazsa hata verir.
4. altprogram1 de tanımlanan a değeri altprogram2 içerisinde tanımlanan a değişkeninden farklıdır.

# Scopes: Blocks

- Bir programlama dilinin deyimlerinin ( if, for, while ) yardımıyla bir araya getirilerek oluşturulan yapıya blok adı verilir.
- Blok içindeki değişkenler blok için lokal değişken olarak tanımlanır.

# Scopes: Blocks

```
void program()  
{  
    int say1;  
    ...  
    while (...)   
    {  
        int say1;  
        say1++;  
        ...  
    }  
    ...  
}
```

- C de yazılan bu kod C++ için doğru iken Java ve C# için doğru değildir.
- While bloğu içerisinde tanımlanan
  - sayı değişkeni while bloğuna ait olan local bir değişkendir

# Scopes: Decleration order

- Bir çok programla dilinde değişkenlerin tanımlama sırası (decleration order) diğer işlemlerden önce gelir.
- Önce kullanılacak değişkenler tanımlanır sonra değişkenler üzerinde işlemler yapılır.
- Java ve C++ da bir değişken tanımlandığı yerden itibaren bloğun sonuna kadar geçerlidir.
- C# da tanımlandığı bloğun hepsi için geçerlidir.

```
void döngü( )  
{  
...  
    for (int i= 0; i < 10; i++)  
    { ... }  
    ...  
}
```

# Scopes: Global Scope

```
a = 1
def fonk1():
    print 'fonk1: ', a

def fonk2():
    a = 2
    print 'fonk2: ', a
def fonk3():
    global a
    a = 3
    print 'fonk3: ', a

fonk1()
fonk2()
fonk3()
```

Bir çok programlama dili global değişken tanımlamasına izin vermektedir.

- C, C++, PHP, JavaScript, and Python

# Scopes: Dynamic Scope

```
int b = 5;
int f1()
{
    int a = b + 5;
    return a;
}

int f2()
{
    int b = 2;
    return f1();
}

int main()
{
    f1();
    f2();
    return 0;
}
```

- Alt programların çalışma anında yazıldıkları sıraya göre değil karışık olarak çağırılmasıdır.
- Dinamik kapsamda çağırma stack inde bulunan en yakın değer alınır.

f1() 10 değerini gönderir  
f2() 7 değerini gönderir

# Scope and Lifetime

- Bir değişkenin kapsamı tanımlandığı blok içindedir
- Değişkenin yaşam süresi programın çalışması bitinceye kadardır.

```
void yazdır()  
{  
    ...  
} /* yazdır son */  
  
void hesapla()  
{  
    int toplam;  
    ...  
    yazdır();  
    ...  
} /* hesapla son*/
```

- Toplam değişkeninin kapsamı hesapla fonksiyonu ile sınırlıdır.
- Yazdır fonksiyonu hesapla içinden çağrılır ve hesapla fonksiyonu tamamlanıncaya kadar toplam değişkeni bellekte tutulur.



# Referencing Environments

Belirlenen noktalarda tanımlı olan bütün değişkenlerin listesine referencing environments denir.

```
k = 3; #global tanımlama
def ap1():
    x = 5; # local x
    y = 7; # local y
    ... (1)
def ap2():
    global k; # Global k atama yapılabilir
    c = 9; # yeni local
    ... (2)
def ap3():
    nonlocal c: # local olmayan c oluşturulur
    k = 11; # yeni local oluşturur
    ... (3)
```

(1) Local x ve y , ve global k

(2) Local c ve global k erişilebilir ve global k ' nin değeri değiştirilebilir

(3) Local olmayan c local k

# Named constants(Sabitler)

- Program içerisinde bulunan bir değişkenin değerinin bir değere sabitlenmesidir.
- Her seferinde 3.14159 kullanmak yerine Pi sabiti tanımlanabilir.
  - `#define M_PI`  
3.14159265358979323846

```
void örnek()  
{  
    int[] intList = new int[100];  
    String[] strList = new String[100];  
    ...  
    for (index = 0; index < 100; index++) {  
        ... }  
    ...  
    for (index = 0; index < 100; index++) {  
        ... }  
    ...  
    average = sum / 100; ...  
}
```