

ALGORİTMA VE PROGRAMLAMA I

HAFTA 5 PROGRAMLAMAYA GİRİŞ

Ders İçeriği: Hafta 5

- Programlama Ortamı
- Açıklama Satırları
- Değişkenler ve Veri Tipleri
- Sabitler
- Değişkenlerin Geçerlilik Alanı
- Veri Tipi Dönüşümleri

Programlama Ortamı

- Java, nesne tabanlı programlamanın en popüler ve etkili uygulama geliştirme ortamlarından biridir.
- Bu ortamda, tüm kodlar alt programlar/metotlar içinde yazılır.
- Metotlar, bir sınıf içinde tanımlanan fonksiyonlardır.
- En temel metodun ismi **main** metodudur.
- main metodu, programın başlangıç noktasıdır ve ilk çalıştırılan bloktur.
- Diğer metotlar, main metodundan çağrılır ve çalışmaları tamamlandıktan sonra kontrolü main metoduna geri dönerler.

Programlama Ortamı

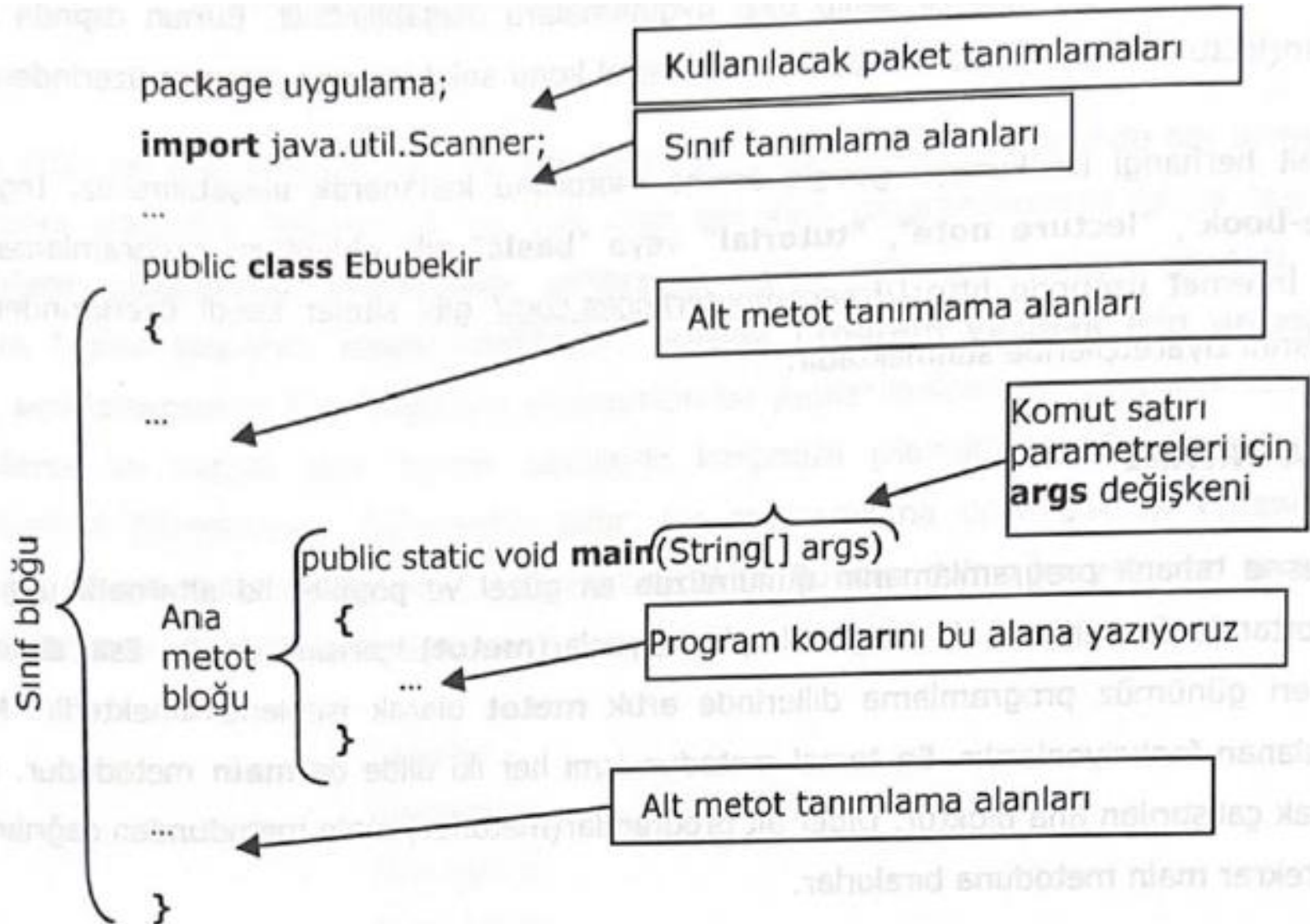
➤ Nesne Tabanlı Programlamada Sınıflar

- Sınıflar, nesne tabanlı programlamanın temel yapı taşlarından biridir.
- İçlerinde benzer özellikler (alanlar) ve işlevler (metotlar) barındıran yapılardır.
- Sınıflar, metotları ve alanları bir arada tutarak onların ilişkisini ve aidiyetini belirleyen bir çatı görevi görür.
- Sınıflardan yeni nesne örnekleri (instance) oluşturulabilir ve bu örnekler, sınıfın tüm özellik ve işlevlerini kullanıma hazır hale getirir.

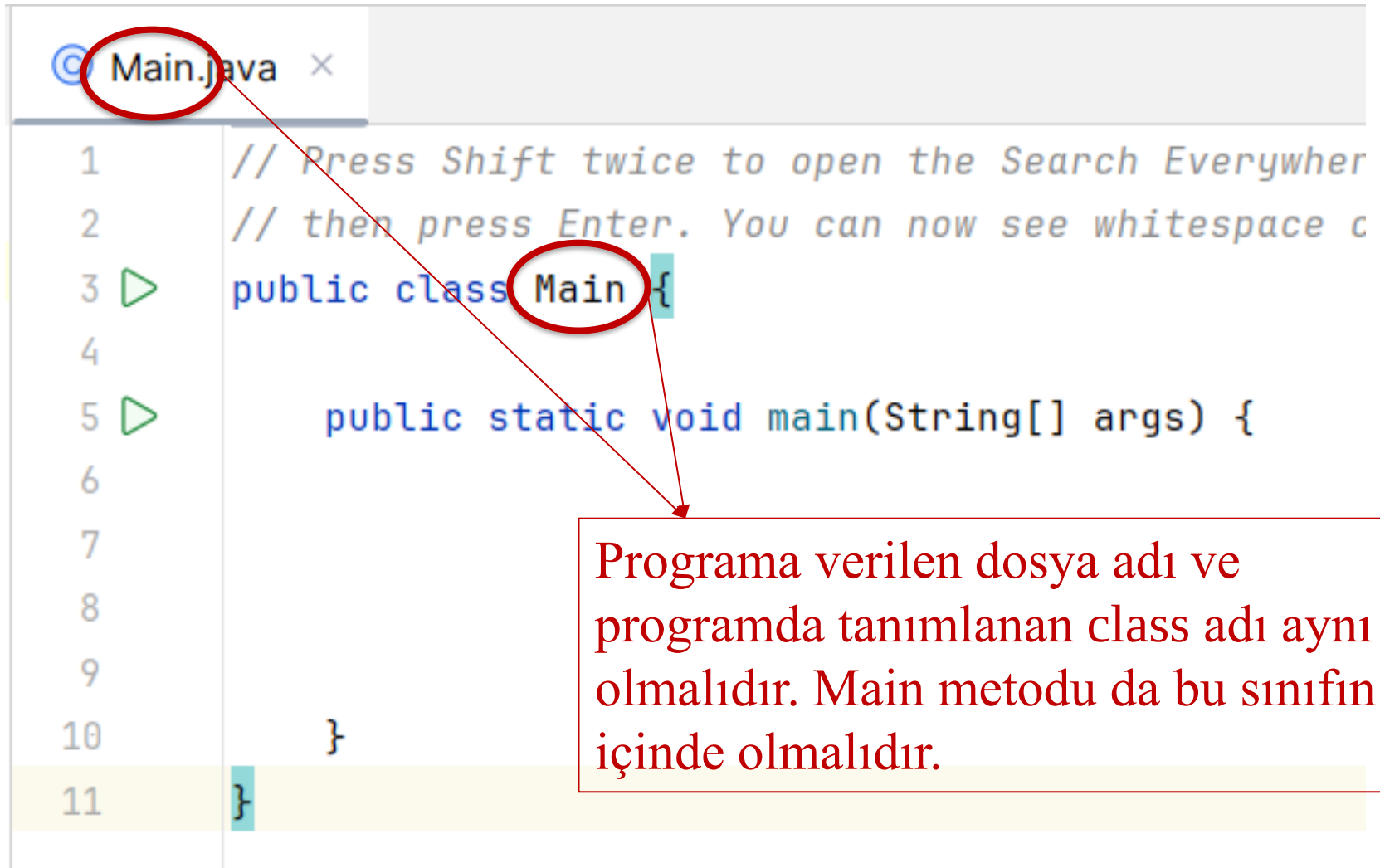
Programlama Ortamı

- Yazdığımız tüm kodlar mutlaka bir sınıf içinde ve ana blok olan main metodu içinde yer almalıdır.
- Başka metotlar kullanılacaksa, bu metotlar sınıf içinde tanımlanmalı ve ana programdan çağrılmalıdır.
- Java'daki sınıflar, paketler (package) içinde yer alır.
- Paketler, Java'da sınıf kütüphaneleri gibi çalışır ve kodun düzenli ve modüler olmasını sağlar.
- import anahtar kelimesi ile tanımlı paketler ve sınıf kütüphaneleri projeye dahil edilir ve kullanılır.

Programlama Ortamı



Programlama Ortamı



```
1 // Press Shift twice to open the Search Everywhere
2 // then press Enter. You can now see whitespace c
3 public class Main {
4
5     public static void main(String[] args) {
6
7
8
9
10    }
11 }
```

Programa verilen dosya adı ve programda tanımlanan class adı aynı olmalıdır. Main metodu da bu sınıfın içinde olmalıdır.

Açıklama Satırları

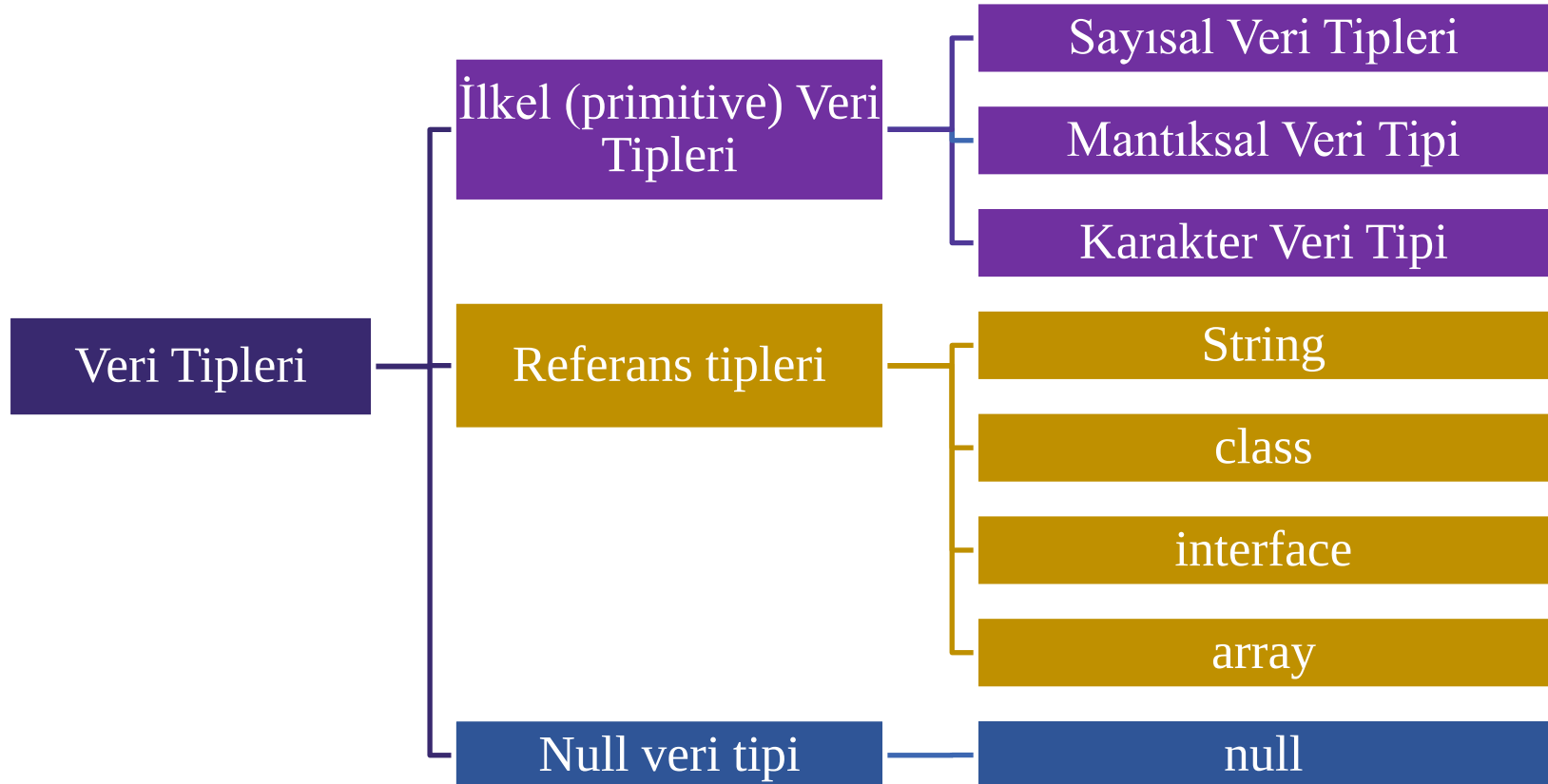
- Program yazarken kodların içerisine açıklayıcı küçük notlar yazılmak istenebilir.
- Bu açıklamalar kod olarak değerlendirilmez.
- Program kodları içerisinde istenen yerde (başta, sonda, arada) kullanılabilir.
- Bunun için aşağıdaki iki yapı kullanılır.

Yapı	İşlevi
//	Kendinden itibaren satır sonuna kadar geçerlidir
/* ... */	/* açılış işaretinden başlayarak */ kapanış işaretine kadarki alanda etkilidir. Çoklu satır açıklamalara uygundur.

Değişkenler ve Veri Tipleri

➤ Java'da Veri Tipleri

- Nesne yönelimli programlama dillerinde, genel olarak, her sınıf bir veri tipidir, her veri tipi bir sınıftır. Java bu genel kurala uyar.



Değişkenler ve Veri Tipleri

➤ Neden Veri Tipi?

- Programlarda farklı veri tipleri ile çalışmak gerekebilir.
- Her veri tipinin bellekte kapladığı alan (bit sayısı) ve yapılabilecek işlemler farklıdır.
- Örneğin, sayılarla dört işlem yapabilirken, metinlerle bu işlemleri yapamayız.
- Bu nedenle, Java ve benzeri diller verileri tiplere ayırır.
- Değişken tanımlarken, o değişkenin hangi veri tipini tutacağını belirtiriz.
- Bu sayede, ana bellekte o değişken için yeterli alan ayrılır ve sadece o veri tipine uygun işlemlerin yapılmasına izin verilir.

Değişkenler ve Veri Tipleri

➤ Java Sayısal Veri Tipleri

Grup	Java Tipi	Uzunluk (Byte)	Değer Aralığı
Tam Sayılar	byte	1 byte	-128 ... 127
	short	2 byte	-32 768 ... 32 767
	int	4 byte	-2 147 483 648 ... 2 147 483 648
	long	8 byte	-2^{63} ... $2^{63} - 1$
Ondalıklı Sayılar	float	4 byte	$1.5 * 10^{-45}$ - $3.4 * 10^{38}$
	double	8 byte	$5.0 * 10^{-324}$ - $1.7 * 10^{308}$

Değişkenler ve Veri Tipleri

➤ Java Karakter ve Metin Veri Tipi

Java Tipi	Uzunluk (Byte)	Değer Aralığı
char	2 byte	Bütün Unicode karakterleri
String	-	Birden fazla karakter (metin)

- Char tipindeki veriler tek tırnak (' ') içine yazılır. ('b')
- String tipindeki veriler ise çift tırnak (" ") yazılır. ("b")

Değişkenler ve Veri Tipleri

➤ Java Mantıksal Veri Tipi

Java Tipi	Uzunluk (Byte)	Değer Aralığı
boolean	1 bit	true veya false değerini alır

- Genellikle program akışının yönlendirilmesinde ve döngü yapılarında kullanılan önemli bir veri tipidir.

Değişkenler ve Veri Tipleri

➤ Değişkenler

- Değişkenler, programda kullanılan verilere kimlik kazandırmak için onlara verilen isimlerdir.
- Program yazarken sayısal, sözel ve mantıksal gibi birçok veri tipi kullanarak çeşitli problemlerin çözümüne ulaşırız.
- Verilerin hafızada tutulması ve işlenmesi, değişkenler sayesinde mümkündür.
- Örneğin, klavyeden girilen iki sayının toplanması için bu sayılar önce hafızaya alınır.
- İşlemci, RAM'deki verileri işler ve sonuçları tekrar hafızaya yükler.
- Yüksek seviyeli dillerde verilerin hafızaya alınması ve işlenmesi işlevini değişkenler üstlenir.

Değişkenler ve Veri Tipleri

➤ Değişken Tanımlamada Dikkat Edilecek Hususlar

- Değişken isimlerinde Türkçe karakterler (İ, ı, ö, Ö, ç, Ç, ğ, Ğ, ü, Ü) kullanılmamalıdır.
- Değişken isimleri sayı ile başlayamaz, ancak sayı içerebilir.
- Operatörler (+, -, /, *, %, (,), vb.) değişken isimlerinde kullanılamaz.
- Alt çizgi(_) içerebilir.
- Java dilinde özel bir anlam taşıyan sözcükler (if, for, break, Class, new...) kullanılamaz.
- Aynı tipe ait birden fazla değişken, aralarına virgül konarak yan yana tanımlanabilir.
- Değişken isimleri uzun (musteri_adi, musteriSoyadi. gibi) olabilir.
- Değişkenler tanımlanırken başlangıç değeri atanabilir..

Değişkenler ve Veri Tipleri

➤ Değişken Bildirimi (Tanımlama)

Veri_tipi Değişken_adı;

■ Geçerli Tanımlamalar

- `int a, b, tutar20;`
- `String musteri_adi, musterisoyadi;`
- `byte Fizik_notu_1;`

■ Geçersiz Tanımlamalar

- `int a+, b, tutar;       //Matematiksel operatör kullanılmış`
- `String müşteri_adı;   //Türkçe karakter kullanılmış`
- `double 4_derece;       //Sayı ile başlamış`

Değişkenler ve Veri Tipleri

➤ Değişken Bildirimi (Tanımlama)

- **Örnek:** Aşağıdaki değişkenlerin geçerli olması için veri tipleri ne olmalıdır?

```
myNum = 9;  
myFloatNum = 8.99f;  
myLetter = 'A';  
myBool = false;  
myText = "Hello World";
```

Sabitler

➤ Sabit Bildirimi (Tanımlama)

- Programda sıkça kullanılan, değişmeyen ve değişmeyecek değerleri kullanmak için sabitler tanımlanır.
- Java'da sabit tanımlama için **final** anahtar kelimesi kullanılır.
- Sabit tanımlanırken mutlaka değer atanması gerekir.
- Program akışı içerisinde sabitlerin değeri değiştirilemez.

Tanımlayıcı Veri_tipi Sabit_adı = Aldığı_Değer;

- Örnek:
 - `final String s = " Bucak Teknoloji Fakültesi ";`
 - `final double pi_sayisi = 3 . 14159265;`

Değişkenlerin Geçerlilik Alanı

- Değişkenlerin hüküm sürdüğü ve kullanılabildiği alanlar onların geçerlilik alanlarıdır.
- Eğer bir değişken kapsama alanı dışında kullanılmaya çalışılıyorsa sanki hiç tanımlanmamış gibi hata verir.
- Örneğin Java için hata çıktısı:
 - Cannot Find Symbol (Değişken bulunamadı)
 - Uncompilable source code (Derlenemeyen kod)

Değişkenlerin Geçerlilik Alanı

➤ Kapsamlı (Global) Değişkenler

- Her metot (alt program/fonksiyon) yapısı içerisinde geçerli olan en üst seviyeli ve en genel etki alanına sahip değişkenlerdir.
- En üst seviyede (sınıf yapısı içerisinde) fakat metot dışında tanımlanırlar.
- Global değişkenler tanımlanırken "**static**" anahtar kelimesi kullanılır.
- Amaç program başladığında otomatik oluşturulup ve sonlanana kadar bu değişkenin hafızada kalmasıdır.

```
static  Veri_tipi  değişken_adı;
```

Değişkenlerin Geçerlilik Alanı

➤ Kısıtlı/Yerel (Local) Değişkenler

- Sadece tanımlı olduğu alt program (metot veya blok) içinde kullanılabilen değişken türüdür.
- Alt programdan çıkıldığında, hafızadan silinirler.
- Bu özellikleri sayesinde hafızanın verimli kullanılmasına yardımcı olurlar.
- Metotlar içinde istenen blok yapısında tanımlanabilirler.
- Yerel değişkenlerin geçerlilik kuralları şunlardır:
 - Aynı blokta üst kısımda tanımlanan değişkenler, alt satırlarda kullanılabilir, ancak tersi doğru değildir.
 - Değişkenler tanımlanmadan önce kullanılamaz.

Değişkenlerin Geçerlilik Alanı

```
package uygulama;  
public class Main {  
    public static void main ( String[] agrs){  
        int numara;  
        numara = 7;  
        not = 200;  
        System.out.println(numara);  
        int not ;  
        not = 100;  
        System.out.println(not);  
    }  
}
```

**numara
değişkeninin
kapsama alanı**

Hata

**Blok
yapı**

**not
değişkeninin
kapsama alanı**

Değişkenlerin Geçerlilik Alanı

```
package uygulama;
```

```
public class Main {
```

```
    public static void main ( String[] args){
```

```
        int numara=7;
```

```
        if(numara==7){
```

```
            int not = 100;
```

```
            System.out.println(not);
```

```
            numara++;
```

```
            System.out.println(numara);
```

```
        }
```

```
        not++;
```

```
        System.out.println(not);
```

```
    }
```

```
}
```

Ana(üst)
blok.
"numara"
değişkeni
geçerli

if bloğu.
"not" ve
"numara"
değişkeni
geçerli

ana blokta tanımlanan
"numara" değişkeni

if bloğunda tanımlanan
"not" değişkeni

Hata! if bloğunda
tanımlanan "not"
değişkeni üst blokta
kullanılıyor.

Veri Tipi Dönüşümleri

➤ Tip Dönüşümü (Casting)

- Veri tiplerini bellekte kapladıkları byte uzunluğuna göre gruplandırdık.
- Program yazarken, bir değişkenin veri tipini başka bir tipe dönüştürmemiz gerekebilir.
- Tip dönüşümü, veriyi bellekte bir adresten başka bir adrese taşımak anlamına gelir.
- Genel kural:
 - Daha küçük bir belleğe sığan bir veri, daha büyük bir belleğe taşınabilir (örneğin, int tipinden double tipine).
 - Daha büyük bir belleğe sığan bir veri, daha küçük bir belleğe taşınamaz (örneğin, double tipinden int tipine), aksi takdirde veri kaybı yaşanabilir.

Veri Tipi Dönüşümleri

➤ İstemsiz (Implicit) Dönüşüm

- Aynı veya farklı gruplara ait kapsanan (küçük) tip kapsayan (büyük) tipe doğrudan (çevrime ihtiyaç olmadan) atanabilir.
- Örnekle açıklarsak, int tipi değişkenler ana bellekte 4 byte (32 bit), long tipi değişkenler ise ana bellekte 8 byte (64 bit) uzunluğunda bellek adreslerine sahiptir demiştik.
 - `int a = 123;`
 - `long b ;`
 - `b = a ;`
- Burada a'nın bellekteki adresinde yer alan 123 sayısı doğrudan b'nin bellekteki adresine taşınmış olur ve veri kaybı yaşanmaz.
- Zorlayıcı bir talep ve veri kaybı olmayan bu dönüşüme istemsiz dönüşüm denir.

Veri Tipi Dönüşümleri

➤ İstemli (Explicit) Dönüşüm

- Bazı durumlarda büyük veri tipinde tanımlanan bir değişkene atanacak değerlerin küçük veri tipinden asla büyük olmayacağı programcı tarafından biliniyorsa, derleyiciyi ikna ederek büyük tipi veriyi küçük tipe dönüştürebilir.
- Örnekle açıklarsak,
 - `int c;`
 - `long d = 456 ;`
 - `c = (int) d ;`
- Burada d'nin bellekteki adresinde yer alan 456 sayısı c'nin bellekteki adresine taşınmış olur.
- Bu taşıma 8 byte'lık bir yerden 4 byte'lık bir adrese yapıldığı için zorlayıcı bir talep içeren bu dönüşüme istemli dönüşüm denir.

Veri Tipi Dönüşümleri

➤ İstemli (Explicit) Dönüşüm

- Bu tür dönüşümler ancak sayısal veriler için yapılabilir.
- String ya da boolean tipleri arasında yapılamaz.
- İstemli dönüşümde veri kaybı olabilir.
- Programcının bu dönüşümü isteyerek ve veri kaybı olabileceğini bilerek yapması gerekir.
- Örneğin,
 - `int n ;`
 - `double x = 12.73;`
 - `n = (int) x;`
- Bu dönüşüm yapıldığında 12.73 sayısının kesir kısmı atılır ve dolayısıyla veri kaybı olur.

Veri Tipi Dönüşümleri

➤ İstemli (Explicit) Dönüşüm

▪ Sayısal – Sayısal Dönüşüm

- Aynı veya farklı gruplara ait küçük tip büyük tipe doğrudan dönüştürülebilir (atanabilir).

byte -> short -> char -> int -> long -> float -> double

- Aynı gruba ait büyük tip küçük tipe çevrilirken ise istemli dönüşüm yapısı kullanılır.

double -> float -> long -> int -> char -> short -> byte

Veri Tipi Dönüşümleri

➤ İstemli (Explicit) Dönüşüm

▪ Sayısal – Sayısal Dönüşüm

```
public class Main {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt; // Automatic casting: int to double  
  
        System.out.println(myInt);    // Outputs 9  
        System.out.println(myDouble); // Outputs 9.0  
    }  
}
```

Veri Tipi Dönüşümleri

➤ İstemli (Explicit) Dönüşüm

▪ Sayısal – Sayısal Dönüşüm

```
public class Main {  
    public static void main(String[] args) {  
        double myDouble = 9.78d;  
        int myInt = (int) myDouble; // Manual casting: double to int  
  
        System.out.println(myDouble);    // Outputs 9.78  
        System.out.println(myInt);       // Outputs 9  
    }  
}
```

Veri Tipi Dönüşümleri

➤ İstemli (Explicit) Dönüşüm

▪ Sayısal – Sözel Dönüşüm

- Java sayısal veri tiplerini metin veri tipine çevirmek için özel metotlara sahiptir:
- `String s = Byte.toString(a);` // a Byte tipini stringe çevirir.
- `String s = Int.toString(a);` // a Int tipini stringe çevirir.
- `String s = Long.toString(a);` // a Long tipini stringe çevirir.
- `String s = Double.toString(a);` // a Double tipini stringe çevirir.
- `String s = Float.toString(a);` // a Float tipini stringe çevirir.

Veri Tipi Dönüşümleri

➤ İstemli (Explicit) Dönüşüm

▪ Sözel - Sayısal Dönüşüm

- Özellikle dışardan (klavye) girilen tüm veriler sözel değişkenlere alınır.
- Bunlar eğer sayı olarak kabul edilecekse ve kullanılacaksa sözel-sayısal dönüşüm yapılmalıdır.
- `byte a = Byte.parseByte(s);` //String-Byte
- `int a = Int.parseInt(s);` //String-Integer
- `long a = Long.parseLong(s);` //String-Long
- `double a = Double.parseDouble(s);` //String-Double
- `float a = Float.parseFloat(s);` //String-Float