Java ile Nesne Merkezli Programlamaya Giriş

3. Bölüm
Java'nın Temelleri

Akın Kaldıroğlu

www.javaturk.org

Aralık 2016

Küçük Ama Önemli Bir Konu

- Bu dosya ve beraberindeki tüm, dosya, kod, vb. eğitim malzemelerinin tüm hakları **Selsoft Yazılım, Danışmanlık, Eğitim ve Tic. Ltd. Şti**.'ne aittir.
- Bu eğitim malzemelerini kişisel bilgilenme ve gelişiminiz amacıyla kullanabilirsiniz ve isteyenleri **http://www.selsoft.academy** adresine yönlendirip, bu malzemelerin en güncel hallerini almalarını sağlayabilirsiniz.
- Yukarıda bahsedilen amaç dışında, bu eğitim malzemelerinin, ticari olsun/olmasın herhangi bir şekilde, toplu bir eğitim faaliyetinde kullanılması, bu amaca yönelik olsun/olmasın basılması, dağıtılması, gerçek ya da sanal/Internet ortamlarında yayınlanması yasaktır. Böyle bir ihtiyaç halinde lütfen benimle, **akin.kaldiroglu@selsoft.academy** adresinden iletişime geçin.
- Bu ve benzeri eğitim malzemelerine katkıda bulunmak ya da düzeltme ve eleştirilerinizi bana iletmek isterseniz çok sevinirim.
- Bol Java'lı günler dilerim.

İçerik

- Bu bölümde şu konular ele alınacaktır:
  - Java'nın anahtar ve ayrılmış kelimelerini,
  - Yorumlar
  - İsimlendirme kuralları,
  - Sabiteler,
  - Veri tipleri, basit ve karmaşık veri tipleri,
  - Nesne ve referans kavramlarını,
  - Değişkenler ve kapsam,
  - Ve tip çevrimleri.

Anahtar Kelimeler

Java'nın Anahtar Kelimeleri (Keywords)

- Toplam 50 tane anahtar kelime vardır.
- * 1.2'de eklenmiştir, ** 1.4'de eklenmiştir,
- *** 1.5'de eklenmiştir,

Ayrılmış Kelimeler

- **goto** ve **const** dilde kullanılmamaktadır, bir anlamları yoktur ama ayrılmış (reserved) kelimedir.
- **null**, **true** ve **false** kelimeleri de ayrılmış Java sabiteleridir.

Yorumlar

Yorumlar I

- Java'da 3 farklı yorum yapısı vardır:
  - "//" (çift yatık çizgi) bulunduğu yerden itibaren satırı yorum yapar

```
double area = 3.14 * r * r;  // Pi is 3.14
```

  - "/* */" Birden fazla satırı blok olarak yorum yapar:

```
/*
 * Bu, birden fazla satırı yorum yapmanın yoludur.
 * Genelde yatık çizgi yıldız ile yıldız yatık
 * çizgi farklı satırlarda olur.
 */
```

Yorumlar II

- "/** */" Birden fazla satırı **Javadoc** yorumu yapar.
- Javadoc hakkında ileride daha geniş bilgi verilecektir.

```
/**
 * Bu birden fazla satırı <b>Javadoc</b> yorumu yapmanın
 * yoludur. Javadoc, bir Java aracı ve iç
 * dokümentasyon oluşturma formatıdır.
 * @see http://www.javaturk.org
 * @see http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html
 * @author Akin Kaldiroglu
 */
```

- "//" ve "/* */" kod içi dokümantasyon, JavaDoc ise API dokümantasyonu için kullanılır.

Comments.java

İsimlendirme

İsimler/Belirteçler (Identifiers)

- Bir programdaki isimlere, **belirteç** (**identifier**) denir.
- Java programlarındaki belirteçler, paketleri, sınıfları, enumları, arayüzleri, metotları, blokları ve değişkenleri isimlendirmek için kullanılır.
- Java'da isimlendirme ile ilgili, diğer dillerdeki gibi uyulması gereken kurallar vardır.

Büyük- Küçük Harf Ayrımı

- Java, küçük-büyük harf ayrımına sahiptir:
  - Araba ve araba farklı isimlerdir.
- Bu durum, böyle bir ayrıma sahip olmayan programlama kültüründen gelenlerin ilk başta zorluk çektiği konulardan birisidir.
- Dilde, nerede küçük harf nerede büyük harf kullanılması gerektiği, genel olarak kodlama geleneği (code conventions) denen belgelerde ifade edilirler.
  - Java'cılar da, kendisine uydukları çok detaylı bir kodlama geleneğine sahiptirler.

İsimlendirme Kuralları

- Java'da da isimlendirme kuralları şunlardır:
  - Anahtar ve ayrılmış kelimeler isim olarak kullanılamaz.
  - İsimler, bir rakam ile başlayamaz ama sonraki karakterlerde rakam içerebilir.
  - İsimler, **"'.,;:!?/\-|#&** gibi noktalama işaretlerini içeremez ama "**_**" kullanılabilir.
  - İsimler "**$**", "**€**", "**¢**", "**¥**", "**£**" ya da "**"**" gibi bir para birimi sembolleri ile başlayabilir.

İsimlendirme

- Programcılar, olabildiğince anlamlı isimler kullanmalıdırlar:
  - Bakıldığında ne işe yaradığı hakkında olabildiğince çok bilgi veren isim kullanmak Java'cılar için bir gelenektir.
  - Java'da isimlerin uzunluklarında bir sınır yoktur.
- İsimlerde, isimlendirme geleneğine göre büyük-küçük harf ayrımı yapılır.
- Java karakterleri Unicode olduğundan Türkçe, Arapça, İbranice, Rusça vb. dillerindeki harfler de kullanılabilir.

İsimlendirme Örnekleri

- Aşağıdaki isimler geçerlidir:

```
sayı, Sayı, sum_$, bingo,
$$_100, mål, grüß, değer
```

- Aşağıdaki isimler ise geçerli değildir:

```
48liler, all/clear, kaybol-buradan, true, true?
```

İsimlendirme Geleneği I

- Java'cılar, kodlama geleneği (code conventions) konusunda, üzerinde anlaşılmış bir yaklaşıma sahiptirler.
- İsimlendirme de bunun bir parçasıdır.
- Java'cılar "Camel Case" denen yaklaşımı kullanırlar.
  - Birleşik kelimeler, her kelimenin baş harfi büyük, diğerleri küçük olacak şekilde, araya boşluk ya da "_" koymadan birleştirilir :

```
LimitedLiabilityCompany
InterestAccruingCheckingAccount
JavaTürk
TürkiyeCumhuriyetMerkezBankası
```

İsimlendirme Geleneği II

- "Camel Case"in iki türü vardır:
  - Upper Camel Case'de ilk kelimenin baş harfi büyüktür.

```
LimitedLiabilityCompany
InterestAccruingCheckingAccount
```

  - Lower Camel Case'de ilk kelimenin baş harfi küçüktür.

```
limitedLiabilityCompany
interestAccruingCheckingAccount
```

İsimlendirme Geleneği II

- Yani:
  - Değişkenler, küçük harfle başlarlar ve deve notasyonuna uyarlar:

```
int accountNumber;
```

  - Metotlar, küçük harfle başlarlar ve deve notasyonuna uyarlar:

```
int getAccountNumber()
```

  - Sınıf (class), arayüz (interface) ve enum isimleri, büyük harfle başlarlar ve deve gösterimine tam olarak uyarlar:

```
CheckingAccount
```

NamingDemo.java

Sabiteler

Sabiteler (Literals) I

- Java'da dört farklı tipte sabite (literal) vardır:
  - Sayı olarak, tamsayı (integer) ve rasyonel ya da ondalıklı sayı (floating-point):

```
2000   -7   0.5
3.1415
```

  - Karakterler, iki tane tek kesme arasına alınır :

```
'a'    '*'    'ş'
'_'
```

  - Klavyenizde olmayan Unicode karakterleri için ilgili Unicode kodlarını kullanabilirsiniz.
    - Bu kodlar da birden fazla karakterden oluşsa bile tek karakter gibi yazılırlar:

```
char ch1 =
    '\u4eca';
```

Sabiteler II

- Mantıksal (logical) sabiteler:

```
true      false
```

- String (karakter dizisi) nesneleri, çift kesme arasında ifade edilir:

```
"Java"    "*** SON ***"    "3.14"
```

- Bir de sadece referanslara atanabilen **null** sabitesi vardır.

Veri Tipleri

Kuvvetli Tipli Bir Dil Olarak Java

- Java, *verilerin tiplerine önem veren* (strongly-typed) bir dildir.
- Tiplere önem veren dillerde, işlenen verilerle ilgili belli kurallar ve kısıtlar vardır.
  - Bu kurallar ve kısıtlar en temelde, verilerin tipleri üzerinden gerçekleşir.
- Java, her değişkenin bir tipinin olmasını zorunlu tutar.
  - Java derleyicisi ve sanal makinası, verilere, tip kural ve kısıtlarını derleme ve çalışma zamanında uygular.

Veri Tipleri

- Java'da yapısı itibariyle iki tür veri tipi vardır:
  - Karmaşık (complex), nesne (object) ya da referans (reference)
  - Basit (simple) ya da ilkel (primitive)
- Dolayısıyla da iki tür değişken ve bu değişkenlerde saklanan iki tür veri vardır.
- Değişkenler ve veriler, basit olsun karmaşık olsun, atama yapılır, metotlara geçilir, üzerlerinde işlem yapılır, metotlardan geri döndürülür.

Basit Veri Tipleri

Basit Veri Tipleri I

- Nesne-merkezli dillerde, idealde her değişkenin, bir karmaşık değişken olması beklenir.
  - Yani her verinin tipinin bir sınıf olması ve her değişkenin de bu sınıftan türetilen nesneyi göstermesi gerekir.
- Java'da bu idealden ufak bir sapma vardır:
  - **Basit** (ya da **ilkel**) **veri tipleri** (**primitive data types**)
- Basit tipler, sınıf değildirler, karmaşık değildirler, atomiktirler.

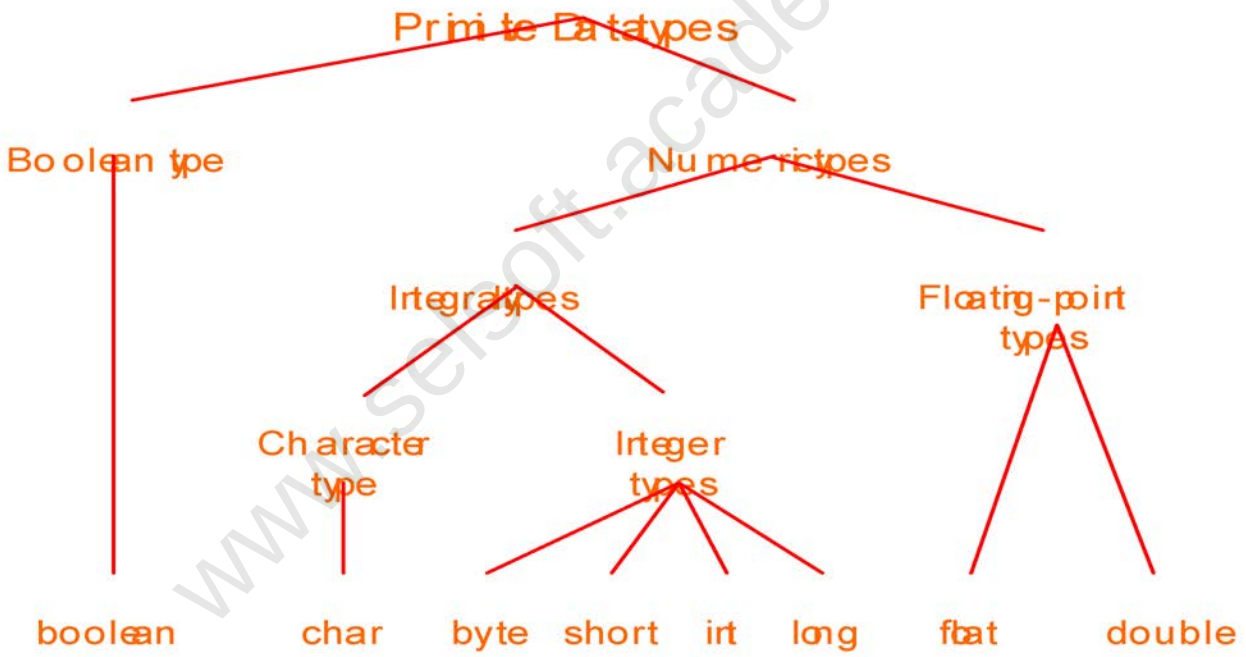
Basit Veri Tipleri II

- Java'ya, temel aritmetik ve mantıksal işlemleri kolayca ve etkin olarak yapabilmek amacıyla konmuştur.
- Basit tipler, nesne tiplerinin aksine atomiktirler, alt parçaları yoktur.
- İstenirse sarmalayan (wrapper) sınıflar aracılığıyla, nesne tipi haline getirilebilirler.
- Basit tipler vasıtasıyla oluşturulan değişkenlere *de* basit değişken (primitive variable) denir.

Basit Veri Tipleri III

- Basit tipler 8 tanedir:
  - Bir tanesi sayısal olmayan mantıksal (logical) bir tiptir: **boolean**
  - Diğer 7 tanesi sayısal tiplerdir.
- **char** dışında diğer bütün 6 sayısal tip de işaretlidir (signed), pozitif ve negatif değerler alabilir.
  - **char**, karakter tipidir ve işaretsizdir, sadece pozitif değer alır.
- Diğer 6 tipin 4 tanesi tamsayılar, 2 tanesi ise rasyonel (kesirli) sayılar içindir.
  - Tamsayılar: **byte**, **short**, **int** ve **long**
  - Rasyonel sayılar: **float** ve **double**

Basit Tipler



Mantıksal Tip

- **boolean** basit mantıksal tiptir.
- Değeri doğru ya da yanlış olan doğruluk durumunu ifade etmede kullanılır.
- Değişkenleri sadece **true** ya da **false** değerler alabilir:

```
boolean flag = false;
boolean kapali = true;
boolean created;
```

- C gibi bazı dillerde var olan, sayıların mantıksal değer ifade etmeleri (true için 1, false için 0) Java için geçerli değildir.

Karakter Tipi

- Karakter tipi **char** ile ifade edilir.
- Bir karakter sabitesi, iki tane tekli kesme "'" işareti arasında ifade edilir.
- Java, Unicode karakter setini kullanır.
  - Unicode karakterleri 16 bitliktir.
- Karakterlerin sayısal değerleri vardır ve pozitiftir:
  - Dolayısıyla karakterler 0 ile 2^16-1 = 65,535 arasında değerler alır.

```
char ilkKarakter = 'a';
char c;
```

Unicode Karakterleri

- Unicode karakter setinin ilk 128 karakteri 7 bitlik *ASCII*, ilk 256 karakteri ise 8 bitlik *ISO 8859-1* (*ISO Latin-1*) karakterleri oluşturur.
- Daha fazla bilgi için http://www.unicode.org

```
' '          \u0020          Boşluk (Space)
'0'          \u0030          0
'9'          \u0039          9
'A'          \u0041          A
'Z'          \u005a          Z
'ç'          \u00e7          ç
'ß'          \u00a7          ß
```

Unicode ve Türkçe Karakterler

- Java karakterleri Unicode olduğundan Java programlarında Türkçe, Arapça, İbranice, Japonca vs. karakterleri rahatça kullanabilirsiniz.
  - Derlerken "**–encoding XXX**" kullanarak "XXX"i kaynak dosyasında kullanılan encoding olarak gösterebilirsiniz.
- Türkçe ya da farklı dillerdeki karakterleri Eclipse gibi araçlarda düzgün görüntüleyebilmek için proje ya da araçta "encoding" ayarı yapmanız gerekebilir.
  - Eclipse'de proje üzerinde **sağ tık => Properties => Resource** sayfasında **Text File Encoding** 'in değerini **UTF-8** yapabilirsiniz.

UnicodeSelamDemo.java

- Koddaki “escape” karakterlerine dikkat edin.

UnicodeCharacters.java

- Bu uygulama görsel olarak Unicode karakterlerini listelemektedir.

TürkçeKarakterlerDemo.java

EnvironmentDemo.java

- Makinanızın dil ve yer ayarları ile makinanızda var olan karakter setlerini öğrenmek için **EnvironmentDemo.java**'yı çalıştırabilirsiniz.

Escape Karakterleri

- Özel escape işlemleri (escape sequence) için özel karakterler vardır:

\b	\u0008	Backspace
\t	\u0009	Horizontal tabulation
\n	\u000a	Linefeed
\f	\u000c	Form feed
\r	\u000d	Carriage return
\'	\u0027	Apostrophe-quote
\"	\u0022	Quotation mark
\\	\u005c	Backslash

- Klavyenizde olmayan Unicode karakterleri için ilgili Unicode kodlarını kullanabilirsiniz.
- Bu kodlar da birden fazla karakterden oluşsa bile tek karakter gibi ('\\') yazılırlar.

EscapesDemo.java

Tamsayı (Integer) Tipler I

- 4 tane tamsayı tipi ve özellikleri aşağıdadır.
  - Uzunluklar, bit cinsindendir.
- Bütün sınır değerleri de aralığa dahildir.

İsim	Uzunluk	Alt Limit	Üst Limit
byte	8	-128	127
short	16	-32768	32767
int	32	-2147483648	2147483647
long	64	-9223372036854775808	9223372036854775807

Tamsayı Tipler II

```
byte b = 8;
short s = 12;
int i = -5;
long l = 47;
```

- Tamsayı tiplerde varsayılan tip **int**'dir
  - Yani herhangi bir yerde tipi belirtilmeden "5" gibi bir tamsayı kullanıldığında bu bellekte 32 bit olarak temsil edilir.
- Bir tam sayıyı **long** olarak belirtmek isterseniz, sayının sağında "**l**" ya da "**L**" kullanabilirisiniz.
  - Fakat benzer şekilde **byte** ya da **short** tamsayı belirtmek mümkün değildir.

```
long l = 47L;
long l = 47l;
```

2lik, 8'lik ve 16'lık Sayılar

- Java'da tamsayıları yani **int** tipindeki değişkenler, 2'lik (ya da 2 tabanlı), 8'lik (ya da 8 tabanlı) ve 16'lık (ya da 16 tabanlı) olarak ifade edilebilir.
- Java'da "**0b**" ya da "**0B**" ile başlayan sayılar 2'lik, "**0**" ile başlayan sayılar 8'lik, "**0x**" ya da "**0X**" ile başlayan sayılar ise 16'lık olarak ele alınırlar:

```
0b1010 = 1*2^3+0*2^2+1*2^1+0*2^0 = 10

0132 = 1*8^2+3*8^1+2*8^0 = 64+24+2 = 90

0x5a = 5*16^1+10*16^0 = 80+10 = 90
```

BinaryOctalHexadecimalDemo.java

Rasyonel Sayılar

- Java'da ondalıklı sayıları belirtmek için 2 tip vardır:
  - **float**, 32 bittir
  - **double**, 64 bittir
- Ondalıklı sayıları kullanırken, onların bir tam sayı bir de ondalık kısmı olduğu ve sıklıkla ondalık kısmını tam olarak ifade etmenin mümkün olmadığını akılda tutmak gereklidir.
  - Tam sayılar sayılabilirler ama reel sayılar sayılamazlar.
- Java'nın **float** ve **double** tipleri IEEE 754 Binary Floating-Point Arithmetic standardına göre oluşturulmuştur.

float ve double I

- Rasyonel sayılarda varsayılan tip **double**'dır.
- Yani herhangi bir yerde tip belirtilmeden örneğin "3.14" gibi bir rasyonel varsa Java derleyicileri bunu 64 bitlik **double** olarak alırlar.
  - Gerekli olmasa da bu durumda sayının yanında "**d**" ya da "**D**" kullanılabilir.
- Bir rasyonel sayıyı **float** olarak belirtmenin yolu ise sayının yanında "**f**" ya da "**F**" kullanmaktır.

```
float pi = 3.14f;
float e = 2.71828F;
double cap = 1.22;
double length = 10.0d;
double alan = 88.15D;
```

float ve double II

- **float** ile ifade edilebilecek en küçük sayı 1.40e-45f, en büyük sayı ise 3.4028235e38f 'dir.

- **double** ile ifade edilebilecek en küçük sayı 4.9e-324, en büyük sayı ise 1.79769313486231570e308 'dir.

FloatingPointDemo.java

Üstel İfadeler

- Rasyonel sayılar, üstel ifade olarak da yazılabilir
- Bunun için "**e**" ya da "**E**" kullanılır:

```
double d1 = 1.6e-19;
float f1 = 6.02E23F;
```

Sayısal Değerlerde "_" Kullanımı - I

- Java SE 7 ile birlikte sayısal ifadelerin okunmasını kolaylaştırmak için "_" kullanımı gelmiştir:
  - "_", bütün sayısal sabitelerde, tam sayı ve rasyonel sayılarda, rakamların arasında kullanılabilir,
  - "**0b**"/"**0B**", "**0**", "**0x**"/"**0X**" ve ile başlayan ikilik, sekizlik ve on altıl tam sayılarla kullanılabilir.
- "_", muhakkak iki digit arasında kullanılmalıdır.

```
int i = 1_000_000;
double d = 2_577.930_113;
float f = 6.02E2_3F;
int b = 0b1010_1010_1010;
```

Sayısal Değerlerde "_" Kullanımı - II

- Şu durumlarda ve yerlerde "_" kullanılamaz:
  - Başta ve sonda,
  - "."nın yanında,
  - "f", "F", "d", "D", "l" ve "L"nin yanında.

UnderscoresDemo.java

PrimitivesDemo.java

- Bu örnekte,
  - Geçerli ve geçersiz değişken isimleri,
  - Değişkenlere atanabilecek değerler
  - Ve diğer bazı basit noktalar

ele alınacaktır.

Karmaşık Veri Tipleri

Sınıf, Nesne ve Referans

- Sınıflar, kategori tanımlarlar.
  - Bu kategorilere de **tip** adı verilir.
  - Aynı sınıftan üretilen nesnelerin tipleri aynıdır.
- Nesneler, **dinamik bellek**teki (**heap**) adacıklar olarak düşünülürse, Java'da bu adacıklara uzanan/erişen **referanslar** (**reference**) vardır.
  - Referans yerine tutaç, kulp (handle), işaretçi, gösterge (pointer) vb. isimler de kullanılır.

Nesne ve Referans

- C++'ta nesnenin fiziksel adresini bilir (pointer) ve onunla nesneye ulaşabilirsiniz.
- Java'da nesnelere, sadece ve sadece referanslarıyla erişilir.
  - Referanslar, nesnelere, soyut erişim mekanizmalarıdır.
  - Bir referans, bir T anında, sadece ve sadece bir nesnenin adresini tutabilir.
- Bir nesnenin birden fazla referansı olabileceği gibi, bir referans farklı zamanlarda aynı tipten birden fazla nesneye referans olabilir.
  - Burada önemli olan şey referans ile nesnenin tiplerinin aynı olmasıdır.
    - En azından şimdilik bunu bu şekilde ifade edeceğiz.

Referans Değişkeni (Reference Variable)

- Dolayısıyla, referanslar aynı zamanda değişkendirler (variable).
  - Bir sınıftan üretilen nesneleri gösteren değişkenlere referans değişkeni (reference variable) denir.
- Referansın tipi, gösterdiği nesnenin tipidir.
  - En azından şu an için böyle kabul edelim.
- Referans değişkenlerine, karmaşık değişkenler (complex variables) de denir.
  - Referans değişkenlerin tiplerine de karmaşık tipler (complex types) denir.

Basit ve Referans Değişkenler - I

- Basit değişkenlerle referans değişkenler arasındaki en temel fark, basit değişkenlerin değeri olmasına rağmen, referans değişkenlerin değerinin, gösterdikleri nesnenin adresi olmasıdır.
- Buna göre aşağıdaki örneklerdeki "j" ile "s1", "s2" ve "tmp"in değerlerinin ne olacağını düşünelim.

```
int i = 5;
int j = i;
i = 7;
```

What is the value of j here?

```
String s1 = new String("a");
String s2 = new String("b");
String tmp = s1; // tmp => "a"
s1 = s2;         // s1  => "b"
s2 = tmp;        // s2  => "a"
```

What are the values of s1 and s2 here?

Basit ve Referans Değişkenler - II

- Basit değişkenler, atanan değeri kendi içlerinde tutarlar.
- Dolayısıyla "j = i"de, "j", "i"nin değerine sahip olur.
  - Sonrasında "i" ve "j" bağımsız olarak farklı değerler alabilirler.
- Ama referanslarda durum farklıdır.
- Referanslar nesneleri gösterdiklerinden, "s1 = s2" ifadesinde "s1", "s2"nin değerine sahip olur, ama bu değer bellekteki bir nesnenin adresidir, "s2"nin gösterdiği nesnenin adresi.
- Bu sebeple, bellekteki bir nesnenin durumunda değişiklik olduğunda, o nesneyi gösteren tüm referanslar bu değişiklikten haberdar olurlar.
  - Çünkü aslında değişen nesnenin durumudur, referansın değeri değildir.

StringDemo.java

- Bu örnekte, referans ile nesne arasındaki ilişki gözlemlenecektir.
  - Referans ve nesne, farklı yapılardır.
  - Bir referans, farklı zamanlarda, kendi tipinden farklı nesneleri gösterebilir, dolayısıyla referans da bir değişkendir.
  - Bir referans, bir nesneyi göstermek zorunda da değildir, bu durumda referans, "null"dır.
  - Null olan bir referansa ulaşmak tehlikelidir
    - NullPointerException!

PrimitiveAndReferenceVariables.java

- Az önce anlatılanların etkilerini görmek amacıyla bu örneği çalıştırın.
- Bu örnekte bellekteki bir nesnenin durumunda değişiklik olduğunda, o nesneyi gösteren tüm referansların bu değişiklikten haberdar olduğunu gözlemleyin.
- Aşağıda soldaki kodun, sağdakiyle yer değiştirdiğinde ne olacağını tartışın:

```
Car tmpCar = car1;
car1 = car2;
car2 = tmpCar;
```

```
car1 = car2;
car2 = car1;
```

Null Referans

- Bir referansa **null** atandığında, o referans hiç bir nesneyi göstermez.

```
car1 = null;
```

- null olan, yani hiç bir nesneyi göstermeyen referans üzerinde, sanki bir nesneyi gösteriyormuşcasına yapılacak, örneğin metot çağrısı gibi işlemler **NullPointerException** hatasına sebep olacaktır.
- Bu yüzden referansların null olup olmadıklarını gerektiğinde kontrol etmek şarttır.

```
if(car1 != null)
   car1.go(50);
```

Değişkenler (Variables)

Değişkenler (Variables)

- Değişken (variable), bellekte bir veriyi (ilkel ya da karmaşık) saklamak ya da tutmak için kullanılır.
- Verinin yapısı ve bellekte nasıl saklanacağı, verinin tipi tarafından belirlenir.
- Java'da değişkenlerin tipi, basit ya da referans tip olabilir.
- Java'da değişkenler kullanılmadan önce tanımlanmalıdır.
- Değişken tanımlamak, değişkeni tanıtmak ve bir ilk değer atamakla gerçekleşir:

```
Değişken Tanımlama = Değişken Tanıtma + İlk Değer Atama
```

```
Variable Definition = Variable Declaration + Initialization
```

Değişken Tanıtma (Declaration)

- Değişkeni tanıtmak, tipi ve ismiyle, o değişkeni derleyicinin tanımasını sağlamak demektir.

```
tip isim;
```

- Değişken tanıtımında ilk değer ataması yapılmaz,
  - Çoğu zaman ilk değerin çalışma zamanında atanması beklenir.

```
int sayac;
boolean b;
String aTinyString;
Student student;
Course course;
```

İlk Değer Atama (Initialization)

- Derleyiciye ismi ve tipiyle tanıtılmış olan bir değişkene, tipiyle uyumlu bir ilk değer ataması yapılmalıdır,
- Ancak ilk değer atamasından sonra değişken tanımlanmış olur.
  - Aksi taktirde değişken kullanılamaz!

```
int sayac;
...
sayac = 5;
```

```
char karakter;
...
karakter = 'c';
```

Değişken Tanımlama (Definition)

- Değişken tanıtma ile ilk değer atamaya, değişken tanımlama (variable definition) denir.
- Java'da değişkenler tanımlanmadığı müddetçe kullanılamazlar.
- Tanıtma ve ilk değer atama bazen bir arada bazen de farklı yerlerde olur:

```
int counter = 5;
char character = 'c';

double ratio;
. . .
ratio = 2.0;
```

VariablesDemo.java

Değişkenlerin Rolleri

- Java'da değişkenler, basit olsun referans olsun, fonksiyonellik ya da rol açısından üçe ayrılırlar:
  - **Nesne değişkenleri** (**instance or object variables**): Nesnenin durumunu oluşturan değişkenlerdir.
  - **Sınıf değişkenleri** (**class variables**): Nesnelerin ortak durumunu ifade eden değişkenlerdir.
  - **Yerel değişkenler** (**local variables**): Geçici değişkenlerdir.
- İlk ikisine **member variables** denir ve sınıf blokunda tanımlanır.
- Öncelikle yerel değişkenleri inceleyeceğiz,
  - Nesne ve sınıf değişkenlerini ilerideki bölümlerde ele alacağız.

Nesne ve Sınıf Değişkenleri

- Nesne değişkenleri (instance ya da object variables), nesnenin durumunu oluşturan değişkenlerdir.
- Nesne değişkenleri, nesne-merkezli programlamada en önemli tipteki değişkenlerdir.
- Çünkü mesne değişkenleri, nesneyi betimlemekte, tarif etmekte kullanılan değişkenlerdir.
- Sınıf değişkenleri ise, o sınıftan oluşturulan nesnelerin ortak değere sahip durumlarını ifade etmede kullanılır.

Yerel Değişkenleri (Local Variables)

- Metot gibi sınıfın alt bloklarının içinde tanımlanan değişkenlere yerel değişken denir.
- Yerel değişkenler, nesnenin parçası değildirler,
  - Yani nesnenin durumunu oluşturmazlar dolayısıyla referanslar üzerinden ulaşılamazlar!
- Bu yüzden yerel değişkenlere aşağıdaki isimler de verilir:
  - Yığın değişkenleri (stack variables)
  - Otomatik değişkenler (automatic variables)
  - Geçici (fani) değişkenler (temporary variables)
- http://en.wikipedia.org/wiki/Call_stack

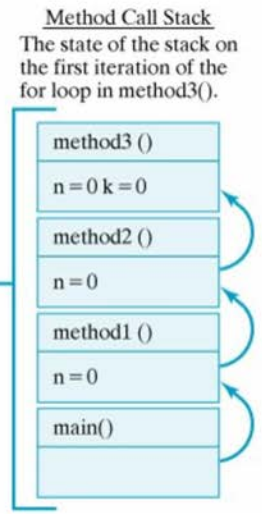
Stack Nedir ve Nasıl Çalışır? - I

- **Stack** (yığın), "last-in, first-out (LIFO)"şeklinde çalışan bir bellek yapısıdır.
- Programların çalışması (execution) sırasında çağrılan metotların değişkenleri için bellek ihtiyacı, stack'te oluşturulan ve frame (pencere) denen bellek alanlarıyla yönetilir.
  - Her girilen yeni metot için yığında en tepeye yeni bir pencere açılarak çalışma devam eder.
  - Bu pencere olarak açılan bellekte, o metodun parametreleri ve o metotta tanımlanan değişkenler, yani tüm yerel değişkenler, oluşturulur.
  - Metot çalışması bitince, kendisini çağıran metoda dönerken yığındaki penceresi de silinir.

Stack Nedir ve Nasıl Çalışır? - II

- Yandaki şekilde `main=>method1=>method2=>method3` şeklindeki bir çağrı zincirinin (call chain) oluşturduğu çağrı yığınının (call stack) bir anki durumu görülmektedir.

```
public class Propagate{
  public void method1 (int n) {
    method2(n);
  }
  public void method2 (int n) {
    method3(n);
  }
  public void method3 (int n) {
    for(int k = 0; k < 5; k + + ) { //Block1
      if(k % 2 = = 0) {             //Block2
        System.out.println(k/n) ;
      }
    }
  }
  public static void main(String args[]) {
    Propagate p = new propagate() ;
    p.method1(0) ;
  }
}
```



Kapsam (Scope)

Kapsam (Scope) - I

- Değişkenlerin görülebildiği ya da erişilebildiği alana, **kapsam** (**scope**) denir.
- Java'da kapsam, bloklar tarafından belirlenir.
- Java'da kapsam derleme zamanında belirlenir (static scoping) ve çalışma zamanında değişmez.
- Değişkenlerin kapsamları, tanıtıldıkları yerden, içinde bulundukları blokun sonuna kadardır.
  - Üye değişkenlerin kapsamı, tanıtıldıkları yerden sınıfın sonuna kadardır.
    - Buna metotlar gibi tüm alt kapsamlar dahildir.
  - Yerel değişkenlerin kapsamı, tanıtıldıkları yerden, o blokun sonun kadardır.
    - Buna tüm alt kapsamlar dahildir.

Kapsam (Scope) - II

- Aynı kapsam içinde birden fazla aynı isimde, üye ya da yerel değişken olamaz.
  - Fakat aynı kapsamda aynı isimde bir üye ve bir yerel değişken olabilir.
  - Bu durumda yerel değişken, üye değişkeni gölgeler (shadowing).
- Java bu durumdaki bir değişkene erişimde, en yakında tanımlanan değişkeni kullanır.
  - Yani yerel değişkeni!

ScopeDemo.java

Global Değişken

- **Global değişken** (**global variable**) programda bir yerde tanımlanıp, programın her yerinden ulaşılabilen değişkenlere denir.
  - C, C++ gibi bazı diller, global değişkenleri destekler.
- Global değişken Java'da yoktur.
  - Java'da doğrudan bu şekilde erişlebilen bir değişken yoktur.
  - Olsa olsa erişim belirteçleri (access modifier) yardımıyla bir sınıf y da nesne için tanımlanan değişkenin, başka sınıf ya da nesnelerden ulaşımı söz konusu olabilir.
- Bu yüzden üye değişkenler ancak "sınıfa global" olarak nitelenebilir.
- https://en.wikipedia.org/wiki/Global_variable

Tipleri Çevrimleri

Tip Çevrimleri I

- Bir değeri, sahip olduğu tipten farklı tipteki bir değişkene atamak istediğinizde **çevrim** (**conversion**) oluşur.
- Bu kısımda sadece basit tipler arasındaki çevrimleri ele alacağız.
  - Karmaşık tipler arasındaki çevrimleri ileride inceleyeceğiz.
- Java, kuvvetli tipli (strongly-typed) bir dil olduğundan, gerek basit gerek ise karmaşık tipler arasındaki çevirimlerle ilgili hem derleme hem de çalışma zamanında pek çok kontroller yapar.

```
long uzun = 5;
int tamsayi = uzun;
boolean b = 12;
```

Tip Çevrimleri II

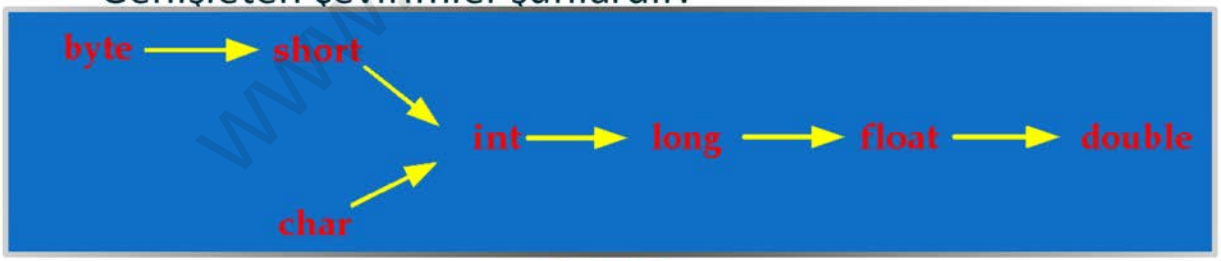
- Basit tipler arasında iki türlü çevrim olabilir:
  - İmkansız çevirim.
  - Daha küçükten daha büyüğe: Genişleten (widening)
  - Daha büyükten daha küçüğe: Daraltan (narrowing)

```
long uzun = 5;          // Genişleten çevirim
int tamsayi = uzun;      // Daraltan çevirim
boolean b = 12;         // İmkansız çevirim
```

- Önce şunları bilmeliyiz:
  - boolean tip ile ilgili hiçbir çevrim söz konusu değildir,
  - byte ve short ile char arasında da bir çevrim olamaz çünkü byte v short işaretli ama char işaretsizdir.
  - Tamsayılar, rasyonel sayılara çevrilebilirler.

Genişleten (Widening) Çevrimler I

- Bit yapısı açısından daha dar bir tipten daha geniş bir tipe olan çevirimlere denir.
- Java'da genişleten çevirimler otomatik olarak yapılır, çevirimin olması için atama yapmak dışında fazladan bir şeye gerek yoktur.
- Genişleten çevrimler hiçbir zaman çalışma zamanı hatası vermezler.
  - Genişleten çevirimler şunlardır:



Genişleten Çevrimler II

- Genişleten çevirimlerde bilgi kaybının olmayacağı gibi yanlış bir varsayım vardır:
  - Tamsayıların ve rasyonel sayıların kendi içlerindeki çevirimlerde bilgi kaybolmaz.
  - Ama tamsayılardan rasyonel sayılara yapılan çevirimlerde bilgi kaybı olabilir.
    - Çünkü rasyonel tipler olan float ve double, virgülden sonrasındaki, ondalık kısım için yer ayırır.

WideningConversionDemo.java

Daraltan (Narrowing) Çevirimler I

- Bit yapısı açısından daha geniş bir tipten daha dar bir tipe olan çevirimlere denir.
- Daraltan çevirimler şunlardır:

- Java'da daraltan çevirimler otomatik olarak yapılmaz.
- Daraltan çevirim yapabilmek için **çevirme operatörü** (**cast operator**) olan "**()**" kullanılır:
  - Atama yapılırken, yeni tip, çevirme operatörü içine yazılır.
  - Bu, Java'nın sizden bir teyit/onay istemesidir.

Daraltan Çevirimler II

- Çevirme operatörü ile çalışma zamanında bir hata oluşmaz.
- Fakat çevirme sonucunda bir veri kaybı olabililir.
  - Rasyonel tiplerden tamsayı tiplerine yapılan çevirimlerde, en az önemli olan bitler kaybolur,
  - Dolayısıyla, rasyonel sayının virgülden sonraki kısmı atılır.
  - Ayrıca negatif değer halinde char tipine çevirim anlamsız sonuçlar doğuracaktır.

```
long lo = 5;
int i = lo;          // Bu bir derleyici
hatasıdır.
int i = (int) lo;
float pi = 3.14;     // Bu bir derleyici
hatasıdır.
float pi = 3.14f; // Ya 'f' ('F') ya da cast
```

NarrowingConversionDemo.java

final

Değeri Sabit Olan Değişkenler

- Değeri sabit olan değişken oluşturmanın yolu **final** anahtar kelimesini kullanmaktır.
- final kullanılarak tanıtılan basit değişkenlerin değeri değiştirilemez.

```
final int i = 5;
i = 8;    // Compile-time error.
```

- Eğer final olacak değişkenin değerinin çalışma zamanında belirlenmesi gerekiyorsa, değişken boş final (blank final) olarak tanıtılabilir.

```
final int i;
i = 8;    // Not a compile-time error.
i = -3;    // Compile-time error.
```

final Referans

- Nesneler doğrudan final yapılamaz, ancak nesnelerin alanları final yapılabilir.
  - Bu şekilde durumu değişmeyen nesne elde edilir.
- Referansın final olmasının anlamı, basit değişkenlere göre biraz farklıdır.
- final referanslar, gösterdikleri nesneden başka bir nesneyi gösteremezler.

```
final Car c = new Car();
c = new Car();     // Compile-time error.
```

```
final Car c;
c = new Car();
c = new Car();     // Compile-time error.
```

FinalVariableDemo.java

Bir Soru?

- Aşağıdaki kod problemli midir sizce?

```
int x = 5, y = 9;
final int i;

if(x > y)
    i = 6;
else
    i = 7;
```

- final olan değişken, örnekteki gibi, yerel bir değişken olduğu müddetçe bir problem yoktur.
- Üye değişkenler için ise durum farklıdır, ileride ele alınacaktır.

Kaynak Kod Yapısı

İfade (Expression) I

- Tek bir sonuç üretecek şekilde, değişkenler, operatörler, ve metot çağrılarından oluşan yapıya **ifade** (**expression**) denir.
- İfadelerde, birden fazla ifade bir araya getirerek, **birleşik ifade** (**compound expression**) oluşturulabilir.
- Bir ifadede üretilen sonucun tipi, o ifadede kullanılan değişkenlerin tiplerine, operatörlere ve metotlara bağlıdır.
- İfade tipleri olarak, aritmetik, mantıksal, atama, metot çağrısı vb. ifadeler sayılabilir.

```
sayac = 5;                  // Simple expr.
int sonuc = 2 * sayac;      // Compound expr.
boolean b = (x > y) & a.f(5); // Compound expr.
```

İfade II

- Java'da ifadelerin değerleri, çalışma zamanında (run-time) belirlenir.
- Dolayısıyla Java derleyicileri, ancak çalışma zamanında belirlenecek bilgileri, derleme zamanında bilseler bile bir işlem yapmazlar.
- Bu durumda oluşabilecek sıkıntılı ya da ters köşe durumlardan programcı sorumludur.
- Birim testi (unit test) yapmak, bu gibi durumları ortaya çıkarır.

ValuesOfExpressions.java

Cümle (Statement) I

- Bir ya da birden fazla ifadenin bir araya gelerek, sonunda ";" olacak şekilde, çalıştırılabilecek bir yapı oluşturmasına **cümle** (**statement**) denir.
  - Cümleler, dillerde en küçük çalışma birimidirler (unit of execution).
- Bazı cümleler, ifadelerin sonuna ";" getirmekle oluşturulabilirler:
- Atama ifadeleri:
- ++ ve -- ifadeleri:
- Metot çağrısı ifadeleri:
- Nesne yaratma ifadeleri:

```
sayac = 5;
a++;
x.f(5);
new A();
```

Cümle II

- Tanıtma cümleleri (declaration statements)

```
int sayac;
boolean b;
A a;
```

- Tanımlama cümleleri (definition statements)

```
int sayac = 4;
boolean b = 2;
A a = new A();
```

- Akış kontrolü cümleleri (control flow statements)

```
if(x > y)
    System.out.println("x büyük");
```

Blok I

- Sıfır ya da daha fazla sayıdaki cümlenin, bir parantez çifti "**{**" ve "**}**" arasında oluşturduğu yapıya **blok** (**block**) denir.
- Bloklar, bir cümlenin kullanıldığı her yerde kullanılabilirler,
  - Cümleler "**;**" ile biter ama bloklar "**;**" ile bitmez
- Java, "blok yapılı" (block-structured) bir dildir.

```
if(x > y)
    System.out.println("x büyük");
// ya da
if(x > y){
    System.out.println("x büyük");
}
```

Blok II

- Java'da en geniş blok, class blokudur:
  - Bir sınıf içinde, tanıtım, ilk değer atama ve tanımlama blokları ile, yerel sınıf blokları (local classes), metot blokları bulunabilir.
  - Metot bloklarında, farklı bloklar, akış kontrol blokları vs. bulunabilir.
  - Blok içinde istediğiniz derinlikte iç içe bloklar oluşturulabilir.
  - *if* ya da *for* gibi akış kontrolü yapıları için de bir metot içinde bloklar oluşturulabilir.

Java Kaynak Kodunun Yapısı (Tekrar)

- Sınıfın bütün öğeleri sınıf bloku içinde tanımlanır,
  - Sınıfın, sadece paket (**package**) bilgisi ile koduna dahil edeceği (**import**) diğer yapıların ifadeleri sınıfın dışında tanımlanır.
- En yaygın sınıf niteleyicisi **public**'tir ve sınıfa her yerden erişilebileceğini ifade eder.
- Her **public** sınıf (arayüz ya da enum), *kendi ismini taşıyan* ve **.java** uzantısına sahip olan bir dosyada yer almalıdır.
  - Dolayısıyla her .java kaynak kodu dosyasında *en fazla bir tane* **public** sınıf olabilir.

```
<package cümlesi>

<import cümlesi>*

public class MyClass{
    ...
}

class SınıfİsmiN{
...
}
```

Kaynak kodun ismi: **MyClass.java**

Sınıflar ve Kaynak Kod Dosyaları

- Bir .java kaynak kod dosyasında bir tane public sınıf olabilmesine rağmen istenilen sayıda public olmayan sınıf olabilir.
- Fakat bu durum birden fazla kişinin olduğu projelerde problem çıkarır.
- Bu yüzden her Java sınıfını kendi ismine sahip bir .java kaynak kod dosyasında tutmak daha uygundur.
- Sizce, içinde hiç bir sınıf hatta hiç bir kod olmayan bir Java kaynak kodu olabilir mi?

Main Metot (Tekrar)

- Java'da pek çok sınıfınız olsa bile en az bir tanesi **main** isimli özel bir metota sahip olmalıdır.
- main metota sahip olan sınıf JVM'e geçilerek çalıştırılabilir.
  - main metot, sistemin çalışmaya başladığı yerdir.
- Dolayısıyla diğer sınıfların nesneleri bu metotta oluşturulur ve üzerindeki metotlar burada çağrılarak sistem çalışmaya başlar.
- main metotun arayüzü (interface) aşağıdaki gibidir:

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

BlocksDemo.java

Özet

- Bu bölümde Java dilinin en temel özelliklerine giriş yaptık.
- Bu amaçla,
  - Anahtar ve ayrılmış kelimeleri,
  - Yorumları,
  - İsimlendirme kurallarını
  - Basit ve karmaşık veri tiplerini,
  - Değişkenleri, kapsamlarını ve **final** değişkenleri,
  - Tip çevrimlerini,
  - Ve Java kaynak kod yapısıyla, ifade, cümle ve blok yapılarını, işledik.

Ödevler

Ödevler I

- *Car* ve *CarTest* sınıflarını örneklerinizin bulunduğu projeye taşıyın.
  - *CarTest* sınıfındaki main metotta 2 tane Car nesnesi oluşturup, değişkenlerine uygun değerler atayın.
  - Oluşturduğunuz nesnelerin *accelarate()* metodunu kullanarak, farklı değerlerde hıza sahip olmasını sağlayın.
  - İki nesnenin de 1000 km'lik bir mesafeyi bu hızlarda giderek ne kadar sürede kat edeceğini yerel değişkenler ve Car nesneleri yardımıyla bulun ve ekrana yazdırın.

Ödevler II

- *Variables* isimli bir sınıf oluşturup, sınıfa bir main metot ekleyin.
  - main metot içinde, birisi String tipinden olmak üzere 4 farklı tipte yerel değişkeni, ilk değer atamadan oluşturun.
  - main metot içinde yerel değişkenlerini print edin. Derleyicinin hatasını yorumlayıp gerekli düzeltmeleri yapın.
  - main metot içinde bir blok oluşturup, blokta ilk değer atayarak bi değişken oluşturup, blok dışında print etmeye çalışın. Hatayı düzeltip programı tekrar çalıştırın.
- Değişkenlere anlamlı ve kod geleneğine uygun isimler verin.

Ödevler III

- *Casts* isimli bir sınıf oluşturup, içine koyacağınız *main* metot içinde, farklı basit tiplerde değişkenler tanımlayarak daraltan ve genişleten çevrimler yapın.
  - Daraltan çevrimleri cast operatörü olmadan yaptığınızda aldığınız hatayı yorumlayın.
  - Çevrimlerden önce ve sonra değişkenleri basarak, değerlerindeki değişimleri gözlemleyin.