**Java Dilinde Veri Tipleri**

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

Java dilinde veri tiplerini üç ana gruba ayırabiliriz.

Veri Tipleri

* İlkel (primitive) Veri Tipleri
* Referans tipleri
* Null veri tipi

Bunların ilk ikisi kendi aralarında alt gruplara ayrılır.

1.İlkel Veri Tipleri

Sayısal Veri Tipleri

Mantıksal (boolean) Veri Tipleri

Sayısal veri tipleri

Tamsayı veri tipleri

byte

short

int

long

char

Kesirli sayı tipleri (floating point types)

float

double

Referans Tipler

class

interface

array

Null Tip

null

Klâsik dillerde *tamsayılar, kesirli sayılar, boolean, void, char gibi yalın veri tipleri ile string, array* gibi bileşik veri tipleri ait olduğu programlama diline gömülü *(built-in-types)* öğelerdir; herbiri bir anahtar sözcükle belirtilir. Oysa, Nesne Yönelimli Programlama dillerinde bu tiplerin herbirisi bir sınıftır. Ancak, program yazarkan ilkel veri tipleri çok sık kullanılır. O nedenle, nesne yönelimli diller bu tür veri tiplerine bir ayrıcalık tanır. Java dili, bu veri tiplerinin, onlara ait nesneler yaratılmaksızın kullanılmasına olanak sağlar. Böyle olması hem program yazmayı basitleştirir, hem de hız kazandırır. Söz konusu sınıfları ileride ele alacağız; konu o zaman daha iyi anlaşılacaktır. Şimdilik, *ilkel veri tipleri* diye adlandırılan aşağıdaki tipleri, klâsik dillerde kullanıldığı gibi nesne yaratmaksızın kullanabileceğiz.

Java derleyicisi, gerektiğinde, tanımlanan her ilkel veriyi ait olduğu sınıfa otomatik olarak gömer (wrapping, boxing). Kutulama (boxing) bir ilkel değişkenin ait olduğu sınıfa konulup bir nesneye ait değişken haline getirilmesidir. Başak bir deyişle, ilkel tipin referans tipine dönüştürülmesidir. Kutu-açma (unboxing) ise tersine yapılan işlemdir; yani kutulanmış bir değişkenin tekrar ilkel veri tipi haline dönüştürülmesidir. Her iki işlemi, programcının müdahalesi olmaksızın, java derleyicisi kendiliğinden yapar. Örneğin, Integer sınıfı int ilkel veri tipini bir öğe (değişken) olarak içerir; buna Integer sınıfının int tipini sarmalaması (wrapping) denilir.

Aşağıdaki tablo ilkel veri tiplerini ve onların gömüldüğü sınıfları göstermektedir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| İlkel Veri Tipi | Uzunluk | Minimum Değeri | Maximum Değeri | Gömüldüğü Sınıf |
| char | 16-bit | Unicode 0 | Unicode 216-1 | Character |
| byte | 8-bit | -128 | +127 | Byte |
| short | 16-bit | -215 (-32,768) | +215-1 *(32,767)* | Short |
| int | 32-bit | -231 (-2,147,483,648) | +231-1 (2,147,483,647) | Integer |
| long | 64-bit | -263 (-9,223,372,036,854,775,808) | +263-1 (9,223,372,036,854,775,807) | Long |
| float | 32-bit | 32-bit IEEE 754 floating-point numbers (kesirli sayı) | | Float |
| double | 64-bit | 64-bit IEEE 754 floating-point numbers (kesirli sayı) | | Double |
| boolean | 1-bit | **true** ya da **false** | | Boolean |
| void | - | - | | Void |

**Neden Veri Tipi?**

Bir programda farklı veri tipleriyle işlem yapmamız gerekebilir. Örneğin, tamsayılar, kesirli sayılar, karakterler (harfler ve klavyedeki diğer simgeler), metinler (string), mantıksal (boolean) değerler (doğru=true, yanlış=false) ilk aklımıza gelen farklı veri tipleridir. Bu farklı veri tiplerinin büyüklükleri (bellekte kaplayacakları bit sayısı) ve onlarla yapılabilecek işlemler birbirlerinden farklıdır. Örneğin, sayılarla dört işlem yapabiliriz, ama metinlerle yapamayız. O nedenle, Java ve başka bazı diller verileri tiplere ayırır. Değişken tanımlarken onun hangi tip veriyi tutacağını belirtir. Böylece, ana bellekte o değişkene yetecek bir yer ayırır ve o veri tipine uygun işlemlerin yapılmasına izin verir.

**Değişken Nedir?**

Bir bilgisayar programında, değişkenler programda kullanılan verileri birbirlerinden ayırmak için onlara verilen adlardır (kimlik). Her veriye ayrı bir ad vermek onların birbiriyle karışmasını önler. Bu, günlük yaşamda da öyledir. Örneğin, bir futbol takımındaki oyuncuların hepsinin adı Ahmet olsa, hiçbir spiker maçı anlatamaz. Nitekim, birden çok Ahmet varsa, spiker, onları soyadlarıyla ayırır ya da takma adlar kullanır.

Derleyiciler açısından, değişkenlere verilen adların kısa ya da uzun olması, anlamlı ya da anlamsız olması önemli değildir. Ancak, kaynak programı yazan ve okuyanın değişkenlerin neleri temsil ettiklerini kolay anlayabilmesi için, onlara anlamlı adlar koymak uygun olur. Tabii, kısa programlarda bu bir sorun yaratmaz, tek harften oluşan adlar bile verilir ve çoğunlukla rahatlık sağlar. Ama uzun programlarda, değişkenlerin temsil ettikleri veriyi belirtecek adlarla adlandırılmaları kolay anlaşılır ve tercih edilen bir usuldür.

Teknik açıdan, değişken, ana bellekte belli bir veri tipine ait değerlerin girilebileceği bir adrestir.

Değişkenler programın ham veri tiplerini taşıyan araçlardır. Java dilinde *global* değişken yoktur. Her değişken bir sınıf içinde ya da sınıftaki bir blok içinde tanımlıdır. Sınıf içinde tanımlı olanlar tanımlandığı *sınıfın bir öğesi (class member)* olur. Sınıf içindeki bir iç-blok içinde tanımlananlar o iç-blokun *yerel değişkeni* olur.

Her değişkene, ana bellekte, o değişkenin tutacağı veri tipine yetecek büyüklükte bir yer ayrılır. Bu yere ana bellekte *değişkenin adresi* denir. Her değişkene bir ad verilir. Bu ad kullanılarak, değişkene değer atanabilir, atanan değer okunabilir ve atanan değer değiştirilebilir (güncellenebilir). Bir değişkene erişim demek, o değişkene değer atama, atanan değeri okuyabilme ve atanan değeri istendiğinde değiştirebilme yeteneklerine sahip olmak demektir.

Java *tip-korumalı* bir dildir. Başka bir deyişle, Java dili programda kullanılacak değişkenlerin tutacağı veri tiplerini kesin sınırlarla birbirlerinden ayırır. Bu yönüyle C ve C++ dillerine benzer. Dolayısıyla, bir değişken bildirimi yapılırken, o değişkenin veri tipi kesinlikle belirtilir. Daha sonra o değişkene atanan veriler, belirtilen veri tipinin dışına çıkamaz. Bunun bir istisnası, bazı durumlarda bir değişkenin tuttuğu verinin başka bir tipe dönüştürülmesidir. Dönüşüm (casting) denilen bu yöntemi ileride açıklayacağız.

Değişken Bildirimi

Java’da bir sınıf içinde bildirimi yapılan değişkenleri üçe ayırabiliriz:

1. Sabitler

2. Dinamik değişkenler (anlık değişkenler, instance variables , non-static members)

3. Statik değişkenler (sınıf değişkenleri, class variables, static variables)

Class Daire

{

final double PI = 3.14159;

short yarıçap ;

static double daireninAlanı;

…

}

Bu örnekte PI bir sabit, yarıçap *dinamik (anlık)* bir değişken, daireninAlanı ise *statik* bir değişkendir.

Sabitler

final nitelemli bir değişken java’da bir *sabit* (constant) yerine geçer.

Sabitler, program koşarken değer değiştiremezler. O nedenle, final nitelemli değişkene, bildirimi yapılırken ilk değeri verilmelidir. Atanan bu ilk değer, program boyunca değişmeden kalır. Öteki değişkenlerden ayırmak için, Java’da *final* değişkenlerin adlarını büyük harflerle yazmak bir gelenektir.

Dinamik Değişkenler (instant variables)

Dinamik (anlık) değişkenler, sınıf içinde bildirimi yapılan ve static nitelemi olmayan değişkenlerdir. Dinamik değişkenler için şu kurallar geçerlidir:

* Nesne yaratılırken her anlık değişkene ana-bellekte bir yer açılır. (allocated on object creation)
* Nesne yokedilince, anlık değişkene açılan yer de yok olur (de-allocated on object destruction)
* Her anlık değişkene, her nesne içinde bir yer açılır (initialized only once)
* Anlık değişkene erişim, ait olduğu sınıfın verdiği izinle sınırlıdır (only accessible where their **class** permits)

Yukarıdaki Daire sınıfına ait bir nesne yaratıldığında, dinamik yarıçap değişkeni o nesne içinde yer alır. Nesne yaratılmadan yarıçap değişkeni için bellekte bir yer ayrılmaz. Demek ki kaç nesne yaratılmışsa, o kadar yarıçap değişkeni yaratılmış olur. Dolayısıyla, nesne sayısı kadar yarıçap değişkeni vardır. Her birisine, ait olduğu nesne içinde, ötekilerden farklı değer atanabilir. Elbette, dinamik değişkene ilk değer atanabilir. Bu değer yaratılan her nesnede yer alır. Tabii, program koşarken bir nesne içindeki dinamik değişkenin değeri değiştirilebilir. Dolayısıyla, herhangi bir anda *dinamik* değişkenin tek bir değeri yoktur; dinamik değişkenin değeri ait olduğu sınıfta aldığı değerdir.

daire d1 = new Daire();

deyimi Daire sınıfına ait d1 adlı bir nesne yaratır.

d1.yarıçap = 3;

deyimi d1 nesnesi içindeki yarıçap değişkenine 3 değerini atar. Benzer olarak,

daire d2 = new Daire();

deyimi Daire sınıfına ait d2 adlı bir nesne yaratır.

d2.yarıçap = 5;

deyimi d2 nesnesi içindeki yarıçap değişkenine 5 değerini atar. Değişkeninin bu değerleri ait oldukları nesne içinde ayrı ayrı yaşarlar.

d1 nesnesi yok olunca, o nesne içindeki dinamik değişken ( yarıçap değişkeni) de bellekten silinir; ama d2 içindeki yarıçap değişkeni yaşamaya devam eder. Onun silinmesi için d2 nesnesinin bellekten silinmesi gerekir.

Statik Değişkenler (sınıf değişkenleri, class variables, static variables)

Sınıf değişkenleri, sınıf tanımında bildirimi yapılan ve static nitelemi olan değişkenlerdir. Sınıf değişkenleri için şu kurallar geçerlidir:

* Program başlayınca bellekte her sınıf değişkenine sabit bir yer açılır (allocated at program entry)
* Sınıf değişkenlerine açılan yerler, program bitene kadar etkin kalırlar, ancak program bitince bellekten silinirler (de-allocated at program exit)
* Sınıf değişkenleri, program boyunca yalnız bir kez yaratılırlar (initialized only once)
* Sınıf değişkenine erişim, ait olduğu sınıfın verdiği izinle sınırlıdır (only accessible where their **class** permits)

Sınıf değişkeni için bellekte bir tek yer ayrılır, sınıfa ait farklı nesnelerde kopyaları olamaz. Tabii, farklı iki sınıfta aynı adlı sınıf değişkenleri olabilir. O durumda, her sınıfın kendisine ait sınıf değişkeni için bir yer açılacaktır.

Sınıf değişkenleri nesneye bağlı olmadan yaratıldıkları için, onlara nesne yaratılmaksızın erişilir. Bu demektir ki, sınıf değişkenlerine henüz hiçbir nesne yaratılmadan erişilebileceği gibi, bütün nesneler yok edildikten sonra da erişim devam edebilir.

Yukarıdaki örnekteki daireninAlanı *statik* bir değişkendir. Ana bellekte (stack bölgesinde) bu değişkene bir tek yer ayrılır. Sınıf içindeki bütün deyimler buna erişebilir. Tabii, *statik* değişkene ilk değer atanabileceği gibi, program koşarken de değeri değiştirilebilir. Ancak, herhangi bir anda *statik* değişkenin bir tek değeri vardır.

**Örnek.**

**Daire.java**

**package** Bölüm05;

**public** **class** Daire {

**static** **final** **double** *PI* = 3.14159;

**short** yarıçap;

**static** **double** *daireninAlanı*;

**double** alanBul(**int** r) {

**int** yçap = r;

*daireninAlanı* = *PI* \* yçap \* yçap;

**return** *daireninAlanı*;

}

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

Daire d1 = **new** Daire();

*daireninAlanı* = d1.alanBul(3);

System.*out*.println(*daireninAlanı*);

System.*out*.println(Daire.*daireninAlanı*);

System.*out*.println();

Daire d2 = **new** Daire();

*daireninAlanı* = d2.alanBul(5);

System.*out*.println(*daireninAlanı*);

System.*out*.println(Daire.*daireninAlanı*);

}

}

Çıktı:

28.274309999999996

28.274309999999996

78.53975

78.53975

Programı satır satır inceleyiniz. d1 ve d2 nesneleri için bulunan daireninAlanı değişken değerlerinin nasıl atandığına dikkat ediniz.

System.*out*.println(Daire.*daireninAlanı*);

deyimi yerine

System.*out*.println(d1.*daireninAlanı*);

deyimi yazılırsa ne olur. Deneyerek görünüz.

Uyarı:

Statik değişkene bağ-operatörü (.) [scope operator] ile erişilmek istendiğinde, sınıf adı kullanılır; nesne adıyla statik değişkene erişilemez.

Statik değişkenler, Pascal ve C dillerindeki global değişkenlere benzer. Ama onlara göre daha işlevseldirler. Çünkü şu avantajları vardır:

İstenirse onlara erişim farklı düzeylerde kısıtlanabilir.

Bütün programda olmak yerine istenen sınıf içinde tanımlanabilir.

Dinamik değişkenlere,

a. bildirimleri sırasında ilk değerleri atanabilir,

b. program koşarken değerleri atanabilir,

c. constructor içinde değerleri atanabilir.

Statik değişkenlere,

d. bildirimleri sırasında ilk değerleri atanabilir,

e. program koşarken değerleri atanabilir,

Java bildirimi yapılan her değişkene, onun tipine uyan öndeğeri atar. Programın öndeğerlerle iş yapması istenmiyorsa, kullanılmadan önce değişkenlere programın gerektirdiği asıl değerleri atanmalıdır.

Yerel değişkenler (local variables)

Yerel değişkenler bir metodun ya da bir blokun iç değişkenleridir. Her metot, sınıf içinde bir bloktur. Ayrıca { } parantezleri içinde yazılı olan deyimler sınıf içinde bir blok oluşturur. Örneğin, döngüler, yönlendirmeler blok örnekleridir. Metodun yerel değişkenleri için şu kurallar geçerlidir.

* Metot çağrıldığında bellekte kendilerine birer yer açılır (allocated on method entry)
* Metodun işi bitince, yerel değişkenler de bellekten silinir (de-allocated on method exit)
* Metodun her çağrılışında, her yerel değişkene yeniden yer açılır (initialized every time)
* Yerel değişkenlere yalnızca ait oldukları metot erişebilir (only accessible in their methods)

Parametreler

Metot imzasında metot adından hemen sonra yazılan () parantezleri içine metodun bağlı olduğu değişkenlerin bildirimi yapılır. Bu değişkenlere, metodun parametreleri denilir. Örneğin,

**double** dikDörtgeninAlanı(**int** a,b) {

**int** uzunluk = a;

**int** genişlik = b;

*Alan* = *uzunluk \* genişlik*;

**return** *Alan*;

}

metodunda ( ) parantezi içindeki a ile b parametrelerdir. uzunluk ve genişlik ise metodun *yerel (iç)* değişkenleridir.

Atama

Atama deyimi değişkene değer veren ifadedir. Değişkene değer atarken, değişken ile değeri arasına (=) simgesi konulur. Bu nedenle (=) simgesine atama operatörü diyoruz. Örneğin, yukarıdaki bildirim yapılmışken,

alan = 15 ;

alan = 3 \* 5 ;

alan = uzunluk \* genişlik ;

deyimleri birer atama deyimidir. Birincisi alan değişkenine 15 literal (sabit) değerini atarken, ikincisi 3\*5 işleminin değerini alan değişkenine aktarıyor. Üçüncüsü ise, uzunluk ve genişlik değişkenlerinin değerlerini çarpıp, sonucu alan değişkenine aktarıyor. Her üç atama deyimi geçerlidir.

Buraya kadar söylediklerimizi özetleyelim:

int faizOranı;

Bu deyim, int tipinden faizOranı adlı bir değişkenin bildirimidir. Bazı kaynaklarda buna değişken tanımı denilir. Biz bildirim sözcüğünü yeğlemekle birlikte, yerine göre her ikisini eş anlamlı kullanacağız. Bazı durumlarda, değişkene, hemen bildirim anında bir ilk-değer vermek gerekebilir. Bu durumda,

int faizOranı = 6 ;

yazılır. Bu bildirim int tipinden faizOranı adlı bir değişken bildirmekte ve ona ilk-değer olarak 6 tamsayısını atamaktadır. Bu söylediklerimizi genelleştirirsek, değişken bildirimi için aşağıdaki sözdizimini kullanacağız:

veri\_tipi değişken\_adı ;

veri\_tipi değişken\_adı = ilk\_değer ;

biçimindedir. Tabii, bildirim anında ilk değer vermek zorunlu değil, isteğe bağlıdır.

Aynı veri tipinden olan birden çok değişken bildirimini tek bir deyimle yapabiliriz:

int m, n, r ;

*Yerel* değişkenler bir metodun ya da bir blokun içinde tanımlanır. *Anlık* ve *sınıf* değişkenleri sınıfın içinde ama metotların içine düşmeyecek biçimde tanımlanır. Anlık ve sınıf değişkenleri arasındaki tek fark, sınıf değişkenlerinin önüne static nitelemesinin konmasıdır. Anlık değişkenlerin final nitelemesi alması durumunda, o değişken ait olduğu sınıfta sabit olur.

Aşağıdaki *kodumsu (pseudocode)*, değişken bildiriminin sözdizimini göstermektedir. Değişken bildiriminde *[]* içindekiler yazılmayabilir; yani ilk değer verilmeyebilir. Ancak, final nitelemeli değişkenlere bildirim sırasında ilk değerleri mutlaka verilmelidir.

class Ad {

// Anlık değişken

erişim\_belirtkesi veri\_tipi değişken\_adı *[ = ilk\_değer]*;

// Sınıf değişkeni

erişim\_belirtkesi **static** veri\_tipi değişken\_adı *[ = ilk\_değer]*;

veri\_tipi metot\_adı() {

// Yerel değişken

veri\_tipi değişken\_adı *[ = ilk\_değer]*;

}

**Java’da adlandırma kuralları**

Java kaynak programında kullanılan değişken, metot, sınıf, paket ve arayüzlerin her birisine bir ad (identifier) verilmelidir. Adlar onların kimlikleridir. Daima o kimlikleriyle çağrılırlar. Başka bir deyişle, adlar ilgili öğeye erişmeyi sağlayan işaretçilerdir. Verilen adlar, işletim sistemine ve derleyiciye bağlı olarak bazı kısıtlara tabidir. Java’daki kurallar aşağıdadır:

· Anahtar ve saklı sözcükler ad olamaz.

· Adlar ya bir harf ile başlamalı ya da ($) simgesi veya (\_) simgesiyle başlamalıdır.

· İlk harften sonrakiler harf, rakam, ($) simgesi ya da (\_) simgesi olabilir.

· Java’da adlar büyük-küçük harf ayrımına duyarlıdır.

· ad içinde boşluk karakteri olamaz.

Derleyiciler açısından, değişkenlere verilen adların kısa ya da uzun olması, anlamlı ya da anlamsız olması önemli değildir. Ancak, kaynak programı yazan ve okuyanın değişkenlerin neleri temsil ettiklerini kolay anlayabilmesi için, onlara anlamlı adlar koymak uygun olur. Tabii, kısa programlarda bu bir sorun yaratmaz, tek harften oluşan adlar bile verilir ve çoğunlukla rahatlık sağlar. Ama uzun programlarda, değişkenlerin temsil ettikleri veriyi belirtecek adlarla adlandırılmaları, kaynak programın kolay anlaşılır olmasını sağlar. Dolayısıyla tercih edilmesi gereken bir usuldür.

Çoğu bilgisayar programlarında değişkenlere verilen adları ifade etmek için *değişken, alan, attribute, parametre* gibi terimler kullanılır. Java için bunlar geçerli olmakla birlikte, javayı yaratanlar değişken yerine *property* sözcüğünü kullandılar. Biz de ona uymak için değişken yerine eş anlamlı olarak *nitelem* sözcüğünü kullanacağız.

Yukarıdaki kısıtlara göre, adlar harf, alt çizgi (\_) ya da dolar ($) simgesiyle başlar. Bu demektir ki, javada adlar rakamla başlayamaz. Harf, (\_) ve ($) dışındaki diğer karakterlerden herhangi birisiyle başlayamaz. Ad, ilk karakterden sonra harf, rakam, alt çizgi ve dolar simgelerinden istenilenlerle devam edebilir. Söylenenler dışındaki karakterler kullanılamaz. Örneğin, artı +, -, \*, “, @, ), < gibi karakterler ad içinde yer alamaz. Ad uzunluğu (adda kullanılacak karakter sayısı) için javada bir sınırlama yoktur. Varsa işletim sisteminin kısıtlarına uyulmalıdır. Örneğin, DOS işletim sistemi sekiz harften uzun adları birbirinden ayıramaz.

Adlandırma kuralına uymak koşuluyla, java öğelerine istediğiniz adı verebilirsiniz. Gereksiz yere uzun adlar vermeyiniz. Çünkü o öğelerle işlem yapmak için verdiğiniz adları eksiksiz ve doğru yazmak zorundasınız.

Değişken adlarını küçük harfle, sınıf adlarını büyük harfle başlatmak, bir gelenektir. Bu geleneğe uyunuz.

Ayrıca, programda tanımladığınız değişken, sabit, sınıf, metot gibi varlıkların adlarını, işlevlerini ima edecek biçimde koymak program okumayı kolaylaştıran iyi bir alışkanlık olacaktır.

Bazı durumlarda, adlar birden çok sözcükten oluşabilir. O zaman sözcükler arasına alt izgi (\_) simgesi koymak izlenen yöntemlerden birisidir. Ama Java dilinde (\_) simgesini kullanmak yerine, deve-notasyonu denen yöntem tercih edilir. Bu stilde, farklı sözcüklerin ilk harfleri büyük yazılarak sözcükler bitişik olarak yazılır. Örneğin, aşağıdakiler geçerli birer değişken adıdır.

faizOranı, sicilNo, öğrencininAdıVeSoyadı, sigortalınınDoğumTarihi

Aynı değişkenleri

faiz\_oranı, sicil\_no, öğrencinin\_adı\_ve\_soyadı, sigortalının\_doğum\_tarihi

diye de yazabilirsiniz. Ama Java ilk yöntemi tercih etmektedir.

Aşağıdakiler javada ad olarak kullanılamazlar:

Faiz Oranı, sicil No, ad+soyad, 2.sınıf, ders-notu, Ahmet’inKitabı

Geçerlik Bölgesi

Her değişken tanımlı olduğu blokta geçerlidir. Bunun anlamı şudur. Değişken sınıfın bir öğesi ise, yani sınıf bloku içinde bildirilmişse, bütün sınıf içinde geçerlidir. Ona sınıfın her metodu, her bloku erişebilir. Sınıfa ait nesne bellekte kaldığı sürece değişken kendi adresini korur; yani varlığını sürdürür.

Sınıfın bir iç-blokunda tanımlı olan değişken yalnız o blok içinde geçerlidir; ona tanımlı olduğu blok içinden ve o blokun alt bloklarından erişilebilir. Blokun işi bitince bellekteki adresi silinir. Örneğin bir for döngüsünün sayacı döngü bloku içinde geçerlidir. Döngü başlarken ona bellekte bir yer ayrılır, döngü bitince bellekten silinir. Benzer olarak, bir metot içinde tanımlı olan bir değişkene metot çağrılınca ana bellekte bir yer ayrılır, metodun işi bitince bellekten silinir.

Bu nedenle, bir metot, bir döngü ya da { } parantezleriyle belirlenen bir blok içinde tanımlı değişkene *yerel değişken* diyoruz. Hiç bir iç-blok içinde olmayıp sınıf içinde tanımlı değişkenlere de *sınıf değişkenleri* ya da *sınıf öğeleri* (class member) diyoruz.

İç-içe bloklar olduğunda, dış bloktaki değişkene iç bloktan erişilebilir. Ama, dış bloktan iç bloktaki değişkene erişilemez. Bir sınıf içinde tanımlı metot, döngü ya da { } ile belirli bloklara göre, sınıfın kendisi de bir bloktur ve öteki blokların hepsini içine alır; yani en dıştaki bloktur. Dolayısıyla, sınıf öğelerine iç blokların her birinden erişilebiliyor olması doğaldır.

**Java Dilinde sabit nedir?**

Java dilinde C dilindeki gibi sabit öğe yoktur. Ancak, sabitin işlevini gören bir değişken bildirimi yapılabilir. Örneğin,

static final int FAİZORANI = 8 ;

static final float AMORTİSMAN = 12.50;

Genel söz dizimi şöyledir:

static final veri\_tipi SABİTİN\_ADI = atanan\_değer ;

Gelenek olarak, öteki değişkenleden ayırt etmek için, sabit adları büyük harflerle yazılır. Değer mutlaka atanmalıdır. Tabii, her öğe için olduğu gibi, sabitlerin de bir sınıf içinde tanımlanması gerektiğini biliyoruz. Örnekse,

class Matematik {

static final float PI = 3.1416;

static final float E = 2.7;

...

}

Sabitler de değişkenler gibi veri tutan sınıf öğeleridir. Dolayısıyla her sabite, ana bellekte tutacağı veri tipine yetecek kadar bir yer ayrılır. Bunun değişkenden tek farkı şudur: Değişkenlere atanan değerler program çalışırken değiştirilebilir (güncelleme). Ancak, sabitlere atanan değerler değiştirilemez. Ayrıca, sabit bildirimi yapıldığı anda değeri atanmalıdır.

Java dilinde sabitler static nitelemesini aldığı için, ait olduğu sınıfın bir nesnesi yaratılmadan kullanılabilir. Ana bellekte java sabitlerine bir tek yer ayrılır. Dolayısıyla, ona erişen bütün öğeler aynı değeri paylaşırlar. Java dilinde static nitelemesinin işlevini ileride daha ayrıntılı ele alacağız.

Sabit Gerekli mi?

Sabit kullanmadan, sabitin işleme girdiği her yerde onun değerini koyarak program yazmak mümkündür. Örneğin, yukarıda bildirimi yapılan yıllıkFaizOranı bir bankanın mudilerinin alacağı faizleri hesaplayan programda kullanılıyor olsun. Yıllık faizin hesaplandığı her işlemde yıllıkFaizOranı yerine 8 kullanılabilir ve program aynı sonucu verir. Ancak faiz oranını değiştirip 9 yapmak istediğinizde, kaynak programın bütününde 8 yerine 9 yazmak gerekecektir. On bin mudisi olan bir bankada bu iş, hem uzun zaman alıcı hem de bazıları unutulabileceği için, hataya açıktır. Onun yerine yıllıkFaizOranı adlı bir sabit kullanılırsa, kaynak programda onun değerini 9 ile değiştirmek, bütün programı güncellemeye denktir. Bunu yapmak büyük bir zaman kaybını önleyeceği gibi, programda hata oluşmasının da önüne geçmiş olacaktır.