JAVA PROGRAMLAMA DİLİ

Java, Sun Microsystems tarafından geliştirilen nesne yönelimli bir programlama dilidir. Ağ (network) ortamı düşünülerek ve yazılımın platform bağımsız olarak çalışması (değişik bilgisayar türlerinde ve değişik işletim sistemlerinde çalışması) düşünülerek geliştirilmiştir. Bu nedenle, diğer dillerden farklı olarak, aynı zamanda kendisi de bir platformdur.

Java teknolojisi, C++'ın dezavantajlı olan yanlarının Smalltalk, Eiffel, Objective C gibi dillerle desteklenmesi sonucu, ilk olarak 1995 yılında ortaya çıktı. İlk olarak 1.0 sürümü çıkartıldı. Ardından sırasıyla 1.1 ve 1.2 sürümleri çıkartıldı ve 1.2 sürümüyle birlikte gelen ciddi değişikler, bu dilin Java 2 olarak anılmaya başlamasına neden oldu. Şu anda yaygın kullanılan sürümün Java 1.4.2 olmasının yanında Java 5.0 da çıkarılmış durumda ve yeni sürümleri çıkmaya devam etmektedir.

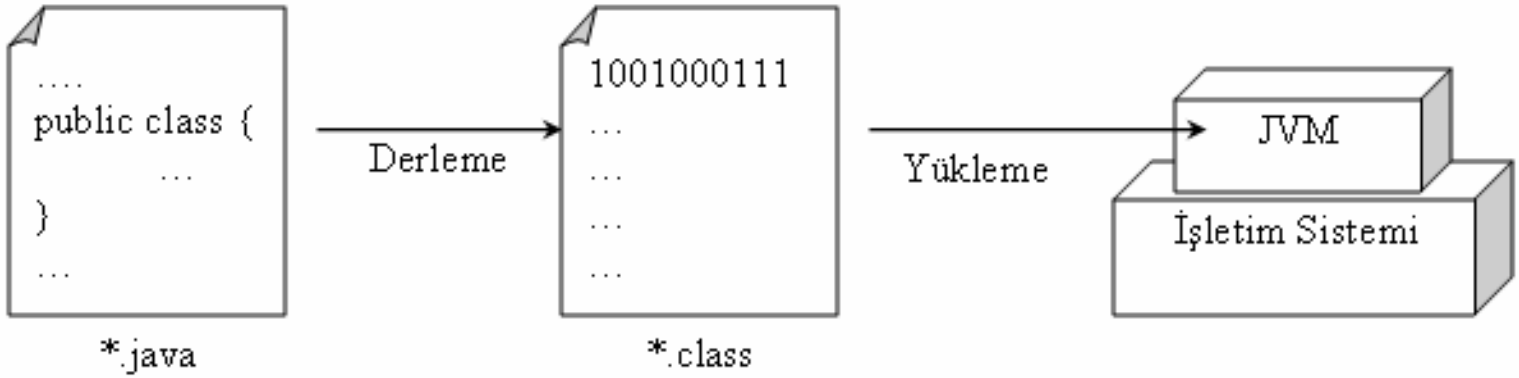
Java uygulamalarınızı yazmak için, Notepad gibi bir program dışında özel olarak geliştirilmiş görsel arayüze sahip olan programlar da vardır. Bu tür programlarda, kodları hem yazar, hem derler hem de çalıştırabilirsiniz. örnek: Eclipse Platform (free platform), Java Builder (BORLAND), Net Bean (SUN), Visual Age (IBM), vb.

Java diliyle birçok uygulama geliştirmek mümkündür:
Grafiksel Kullanıcı Arayüzü (Graphical User Interface: GUI) uygulamaları,
Applet'ler,
Dağıtılmış bileşenler (EJB, RMI, CORBA, vb.),
Web tabanlı uygulamalar (Servlet, JSP, vb.),
Veritabanı erişimli uygulamalar,
Cep telefonu, Akıllı kart uygulamaları.

Java platformları:
Standart Java
Enterprise Java
Gömülü cihazlar (embedded devices) için Java

JAVA NASIL ÇALIŞIR?

Java kodları, Notepad gibi basit bir kelime işlemci programı ile yazılır. Yazılan kodlar, .java uzantısıyla kaydedilir. Bu haliyle kodlar metin halindedir. Bu kodların çalışması için önce derlenmeleri gerekir. Derlemek için, bir derleyici gerekir. Bunun için J2SE - SDK (Java 2 Platform Standart Edition – Software Development Kit) yüklenmiş olması gerekir. J2SE'de derlenen kodlar, .class uzantılı hale gelir. Bu haliyle kodlar ikili kod (binary code) şeklindedir. Derlenen kodlar, çalıştırılmak için Java Virtual Machine (JVM) adı verilen ortama yüklenir. JVM, J2SE'nin içinde bulunmaktadır. Derlenen dosya çalıştırıldığında, kodlardan istenilen işlemler gerçekleştirilir; uygulama çalışmış olur.



Derleme için yazılan kod:

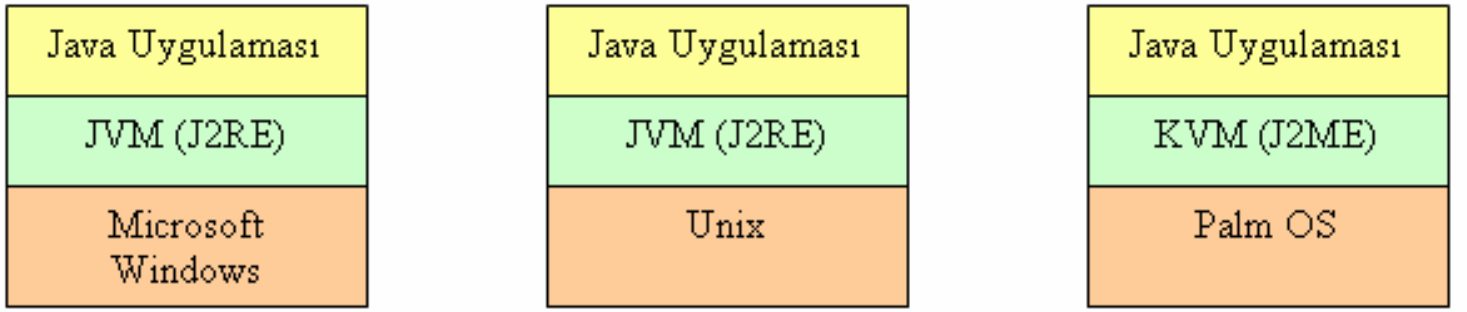
javac *.java

Çalıştırma için yazılan kod:

java *

Derlenen kodlar çalıştırılmak üzere JVM'e yüklenirken; sınıflar yüklenir, byte kodların kontrolü yapılır ve yorumlama başlar.

JVM, Java dilinin işletim sistemi tarafından anlaşılmasını sağlar. Bu sayede, Java ile yazılan bir program herhangi bir işletim sisteminde çalışır hale gelir. JVM, bir anlamda sanal bir işletim sistemi gibi düşünülebilir. JVM yüklü olmayan işletim sistemlerinde, Java kodları çalışmaz. Bu nedenle, Flash programında hazırlanan .swf dosyalarının çalışması için Flash oynatıcısının gerekmesi gibi, Java uygulamalarının çalışması için de sisteme JVM yüklenmesi gerekir. Günümüzde bazı tarayıcılar JVM destekli olarak geliştirilmiştir. İşletim sistemine göre, kullanılan JVM türü de değişebilir.



KAYNAK ÖNERİLERİ:

- "Java ile Temel Programlama", Bora Güngören, Seçkin Yayıncılık
- "Java Uygulamaları", David Flanagan, Pusula Yayıncılık
- "Just Java", Peter van der Linden, Prentice Hall
- "Java in a Nutshell", David Flanagan, O'Reilly
- http://tr.sun.com/training/courses

JAVA UYGULAMALARINDAN BİR ÖRNEK: APPLET UYGULAMALARI

Applet'ler, Java'nın ilk uygulamalarındandır. Applet'ler, tarayıcılarda çalıştırılan mini programlardır. HTML ile bir arada kullanılırlar. Ancak, günümüzde applet ile birlikte farklı script dilleri de kullanılmaktadır: JavaScript, VBScript vb.

PROGRAMLAMA DİLLERİ

Java, nesne yönelimli bir programlama dilidir. Java'nın özelliklerini incelemeden önce, nesne yönelimli olma kavramını programlama dillerinin tarihine bakarak incelemek yerinde olur.

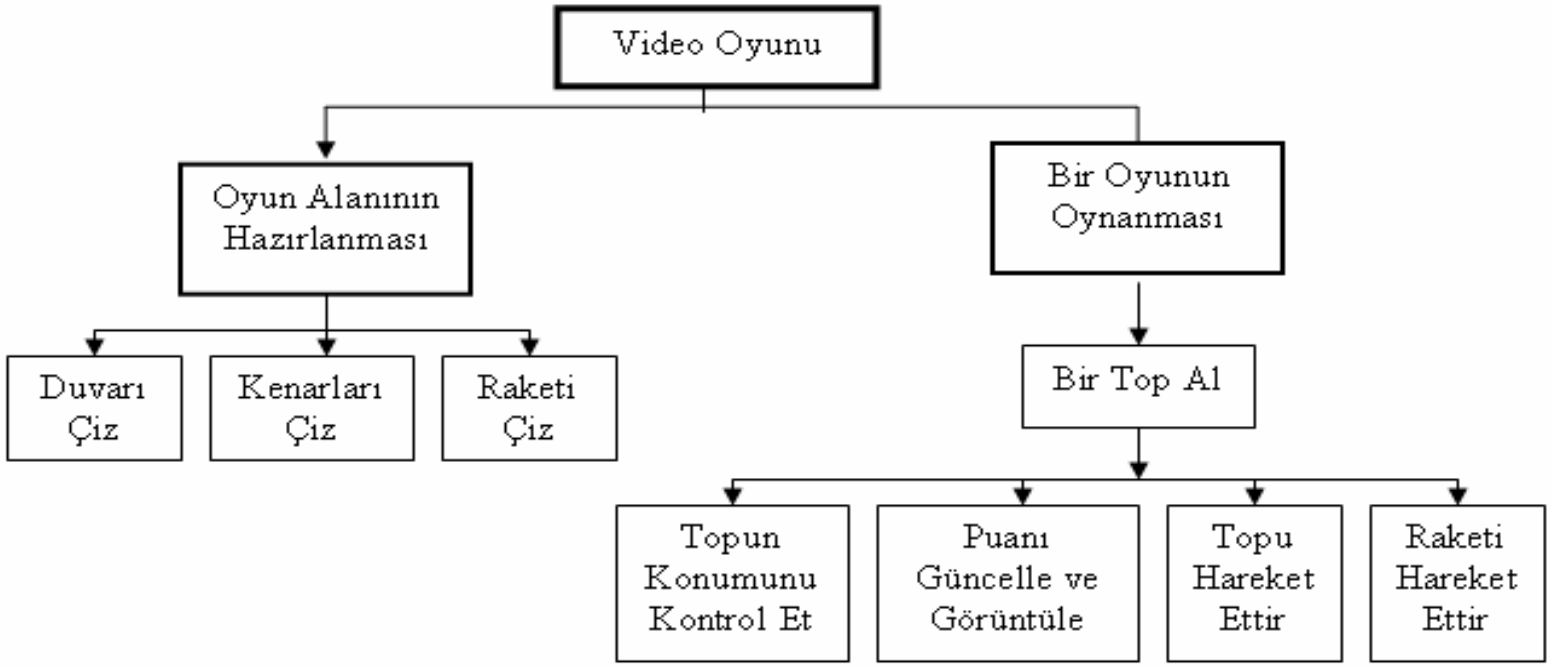
Geliştirilen yüksek seviyeli ilk programlama dili, 1956 yılında IBM tarafından geliştirilmiş olan FORTRAN dilidir. Fortran'ı 1958 yılında ALGOL ve COBOL dilleri izler. 1960'lı yıllarda CPL, BCPL, Simula gibi çeşitli diller de tasarlanmıştır. 1970 yılında, C programlama dili ve 1972 yılında PASCAL programlama dili tasarlanmıştır. Bu diller, yapısal diller olarak adlandırılır. Bu dillerdeki algısal karmaşıklığı ortadan kaldırmak amacıyla, 1980 yılından itibaren nesne yönelimli programlama geliştirilmeye başlanmıştır. C++, 1980'li yılların ilk yarısında tasarlanmıştır. Nesne yönelimli ilk programlama dili olduğu söylenebilir. Java ise, C++'ın dezavantajlı olan yanlarının Smalltalk, Eiffel, Objective C gibi dillerle desteklenmesi sonucu, ilk olarak 1995 yılında ortaya çıktı.

Yapısal programlama dillerinde, veriler ve metotlar ayrı ayrı tutulur. Veriler veri yapılarına gönderilir; kod blokları metotlara ve/veya fonksiyonlara gönderilir. Dolayısıyla, yapısal programlama geliştirmede, fonksiyonlara bağımlılık ve kodları takip etme zorluğu gibi dezavantajlar ortaya çıkar.

Yapısal programlama geliştirme sürecinde:

- fonksiyonlar belirlenir
- fonksiyonlar arasındaki ilişkiler belirlenir
- fonksiyonların özellikleri belirlenir

Örnek:



Önceki slayttaki örnekte, çok bilinen bir oyun olan tuğla oyununa ait yazılımın yapısal bir dille geliştirilmesi için tasarlama sürecine ait plan yer almaktadır. Her kutu, belli bir fonksiyonu işaret eder. Kutular arası oklar, fonksiyonlar arasındaki ilişkileri temsil eder. Fonksiyonlar, özellikleri belirlenmiş olan işlemlerdir.

Nesne yönelimli programlama geliştirmenin en önemli özelliği, nesnelerin kullanımıdır. Görselleştirilebilen, kavramsallaştırılabilen veya modellenebilen her şey nesne olabilir.

örnek:

müşteri bilgileri, müşteriler, hesaplar, para, vb.
öğrenciler, notlar, dersler, vb.
ürünler, ürün bilgileri, stok bilgileri, marketler, vb.

Nesneler, gerçekleştirilecek olan işlem kendisine bildirildiğinde tepki verirler ve bir cevap döndürürler. Gerçekleştirilecek işlemi mesaj; mesajı alan nesneyi de alıcı olarak adlandırabiliriz. Mesaj alıcıya gönderilir ve alıcı mesaja tepki verir. İşlem bu şekilde gerçekleştirilir.

örnek:

ilkAraba.renkVer();
(ilkAraba nesnesine, renkVer işlemi bildirilir.)

Nesneler, veri ve metotlar içerir. Veri, bir sınıfın sahip olduğu özelliklerdir. Metot ise, bir sınıfın sahip olduğu işlemlerdir.

örnek:

ilkAraba.renkVer();
(ilkAraba nesnesi, bir araba olduğu için tekerlek, direksiyon, renk vb. verilerine sahiptir. renkVer metodu ilkAraba nesnesi için tanımlı olduğundan, ilkAraba nesnesi renkVer metoduna sahiptir.)

Nesneler belirli özellikler taşırlar ve ortak özelliklerine göre sınıflandırılırlar. Sınıflandırılmış bir nesne grubunun özelliklerini taşıyan nesne şablonuna "sınıf" denir. Başka bir deyişle sınıf, belirli özelliklere sahip olan nesneler için bir şablondur. Her sınıf, kendisine ait olan verileri ve metotları tanımlar.

Bir sınıfa ait olan özellikleri taşıyan belirli tipte bir nesnesi, o sınıfın bir "örneği" olarak adlandırılır.

örnek:

"Araba" bir sınıfsa:
ilkAraba, bir örnektir.
ilkAraba'nın renk özelliği, bir veridir.
renkVer, ilkAraba'nın rengini belirleyen bir metottur.

Bu örnek üzerinden gidersek, bir araba renk, direksiyon, kapı, tekerlek, vb. gibi özelliklere ve ileri gitme, geri gitme, durma, silecek çalıştırma, vb. gibi işlemlere sahiptir.

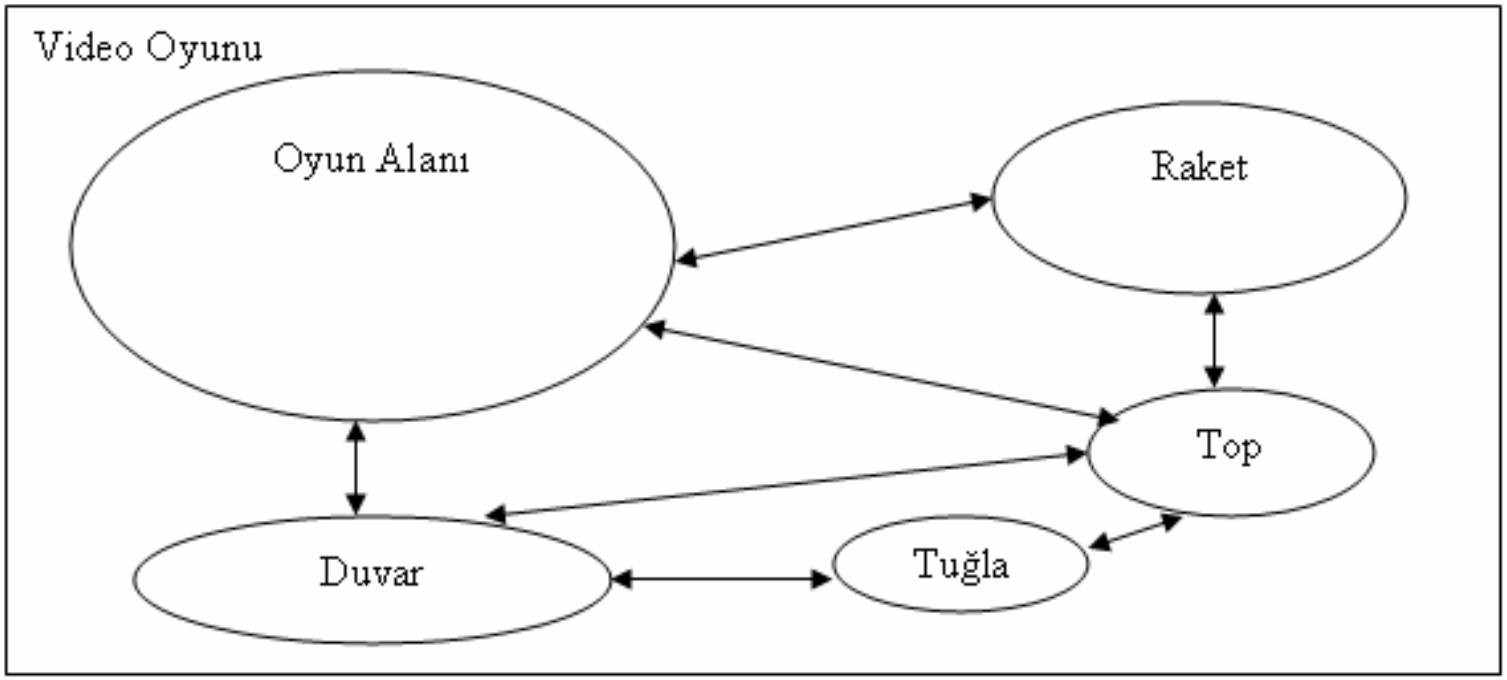
Dolayısıyla bu özelliklere ve işlemlere sahip olan bir sınıf tanımlar ve adını da “Araba” koyarsak, tüm bu özellikler “Araba” sınıfının birer verisi; tüm işlemler de birer metodu olur. Bu sınıfa ait bir örnek tanımlar ve adına da “ilkAraba” dersek, bu örnek de ait olduğu sınıfın tüm verilerine ve metotlarına sahip olacaktır.

Burada önemli olan, bir sınıfı tanımlarken o sınıfın verilerini ve metotlarını doğru ve eksiksiz tanımlamaktır.

Nesne yönelimli programlama geliştirme sürecinde:

- nesneler belirlenir
- nesneler arasındaki ilişkiler belirlenir
- nesnelerin özellikleri belirlenir

Örnek:



Yukarıdaki örnekte, yine tuğla oyununa ait yazılımın bu kez nesne yönelimli bir dille geliştirilmesi için tasarlama sürecine ait plan yer almaktadır. Her elips, belli bir nesneyi işaret eder. Elipsler arası oklar, nesneler arasındaki ilişkileri temsil eder. Nesneler, özellikleri belirlenmiş sınıflara ait olan örneklerdir.

JAVA'NIN YAPISI

NESNE YÖNELİMLİDİR

Gerçek hayattaki nesneler, Java'da temsil edilebilir.
örnek: öğrenci, ders, öğretmen, not
Java'nın sözdizimi C++'ın sözdizimine benzer; ama nesneye dayalı bir programlama dilidir. Nesneye dayalı tasarım ve mimarisi; Smalltalk, Eiffel, Objective C gibi nesne yönelimli dillerden türetilmiştir.

SÖZDİZİMİ

Sözdizimi, C ve C++'ınkine benzer.
C++'ın aşağıdaki özellikleri Java'da yoktur:
İşaretçi (pointer) aritmetiği
Otomatik tip değiştirme
Typedefs, Defines, Pre-processor, Enums
Structures, Unions
Fonksiyonlar
Çoklu kalıtım
Kullanıcının tanımladığı Overloading Operatörler

PLATFORM BAĞIMSIZDIR (MİMARİ OLARAK NÖTRDÜR)

•Java programları platformdan bağımsızdır.
•Java derleyicisi, kaynak kodu byte kod komutlarına çevirir.
•Byte kod komutları, Java Virtual Machine (JVM) tarafından yürütülür. JVM, özel bir yazılımdır. Bilgisayarın işletim sistemi ile kendi üzerinde çalışan Java uygulaması arasında bir katman olarak yerini alır. Bu şekilde, işletim sisteminin hafızasını işgal etmez.

TAŞINABİLİRDİR

•Java, “Bir kere yaz, her yerde çalıştır.” ilkesine %100 uyar. Java’nın işletim sistemine bağımlı değildir:
•Java, bir tamsayının boyut ve aritmetik davranışını kendisi belirler; bu işi çalıştığı makineye/işletim sistemine bırakmaz.

DAĞITIKTIR

•Java TCP/IP ağ yeteneklerine sahiptir. örnek: HTTP, soket, vb.
•RMI (Remote Method Invocation) ve EJBs (Enterprise Java Beans) kullanarak, bir JVM’deki nesneler, diğer bir JVM’deki nesnelerle haberleşebilir.
•CORBA kullanarak, Java nesneleri başka nesnelerle haberleşebilir.

ÇOK KANALLI (MULTI-THREADED) YAPIDADIR

- Java, kanalları yönetme yeteneğine sahiptir.
  - Çeşitli görevler, bir Java uygulamasında eşzamanlı olarak çalışabilir.
- Bazı eşzamanlı erişim kontrolleri dile gömülmüştür.

Java, eşzamanlı erişimi kütüphane seviyesinde değil, dil seviyesinde yapar.

SIKI KURALLI (STRONGLY-TYPED) BİR DİLDİR

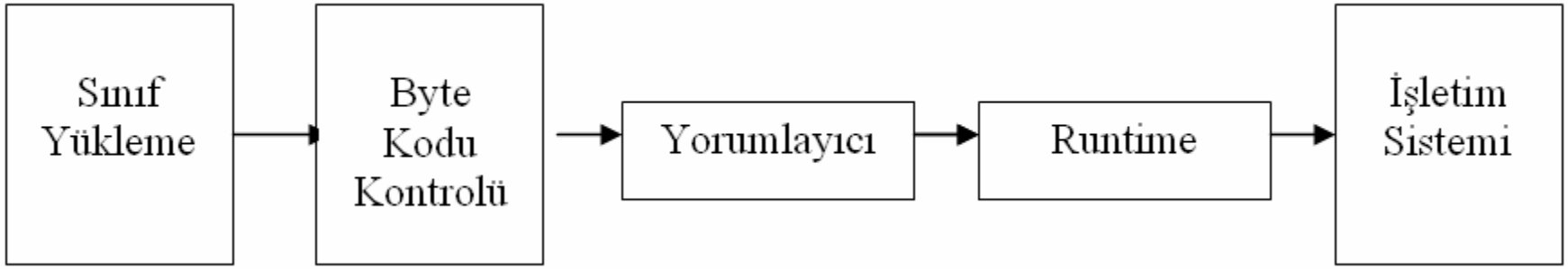
- Bir ifadenin tipi, derleme zamanında belirlenebilir. Bu şekilde, derleyici hataları bildirebilir.
  - Java'da gönderilen ve alınan her parametrenin bir tipi olmalıdır.
  - Tanımlanan her değişkenin bir tipi olmalıdır.

HAFIZA KULLANIMI

- Java, açık olarak işaretçi kullanmaz.
- Java, hafızada yer ayırma ve ayrılan yeri kaldırmayı otomatik olarak yapar.
- Java, otomatik çöp toplama (garbage collection) yapar.

DERLENİR VE YORUMLANIR

- JVM'de özel yorumlayıcı vardır.
  - Runtime'da, sınıflar yüklenir.
  - JVM'deki yorumlayıcı, derlenmiş byte kod komutlarını makine komutlarına çevirir.
  - Kodlar çalıştırılır.
- Java destekli tarayıcılar, Java uygulamaları çalışacak şekilde bir Java yorumlayıcısı da içerirler.



DİNAMİKTİR

- JVM, nesnenin tipine göre runtime'da hangi kodların çalışacağına karar verir.
- Java, runtime'a kadar bir uygulamayı oluşturan modülleri birleştirmez. Bu şekilde, kütüphaneler istenildiğinde yükseltilebilir (upgrade) veya değiştirilebilir.

Java platformu kurduktan sonra, C:\ deki bir dizininde Java klasörü ve bu klasörün altında da Java kaynak kodlarınız için derleme ve çalıştırma yapmanızı sağlayan .exe dosyaları yer alacaktır.

Java kaynak kodlarını Notepad'de (ya da bir kelime-işlemci program) yazacaksınız. Java yorumlayıcısı, main(…) metodunu çalıştırır. Dolayısıyla çalıştırmak istediğiniz Java uygulamanızda bir main(…) metodunuz olmalıdır. main(…) metodu aşağıdaki gibi tanımlanır:

```
public static void main(String[ ] args){
// kodlar buraya yazılır
}
```

Java kaynak kodları, .java uzantılı olarak kaydedilir. Eğer Notepad dışında bir kelime-işlemci kullanıyorsanız, dosyanızı "salt metin (text only)" seçeneği ile kaydetmeniz gerekir.

Java kaynak kodlarını derlemek ve çalıştırmak için DOS (Windows işletim sistemi üzerinde MS-DOS) ortamında çalışacaksınız.

Java kaynak kodlarını derlemek için, bu kodlar için Java derleyicisini çalıştırmanız gerekir:

javac Beykent.java

(Beykent adlı dosyanız için derleme kod satırı)

Derleme sonunda (eğer herhangi bir sözdizimi hatası olmazsa) .class uzantılı bir byte kod dosyası yaratılır. Bu kodları çalıştırmak için Java yorumlayıcısını çalıştırmanız gerekir:

java Beykent

(Yine Beykent adlı dosyanız için çalıştırma kod satırı)

Çalıştırma sonucunda, kodlarınızda belirttiğiniz işlemin sonucu ekrana getirilir.

KOD YAZMA, DERLEME VE ÇLAIŞTIRMA SÜREÇLERİNDE DİKKAT EDİLMESİ GEREKENLER:

1. Tüm Java kaynak kodları .java uzantılı olarak kaydedilmelidir.
2. Eğer Notepad dışında bir kelime-işlemci kullanıyorsanız, dosyanızı “salt metin (text only)” seçeneği ile kaydetmeniz gerekir.
3. Her kaynak kod dosyası için sadece bir “public” sınıf olmalıdır. Dosyanıza vereceğiniz isim, bu sınıftan sonra yazılmalıdır.
4. Eğer derleme sırasında “error: cannot read *.java (hata: *.java dosyası okunamıyor)” hata mesajını alırsanız, derleme kodunu Java kaynak kodu dosyanızın olduğu dizinde çalıştırdığınızı kontrol edin.

5. Eğer derleme sırasında “Java cannot find the class or package, … (Java bu sınıfı ya da paketi bulamadı, …)” hata mesajını alırsanız, CLASSPATH’in (sınıfyolu) doğru tanımlandığını kontrol edin. CLASSPATH, bilgisayarınızın sistem yolunda (system path) aşağıdaki gibi tanımlanmış olmalıdır:
6. *Başlat (Start) > Kontrol Panel (Control Panel) > Sistem (System) > Gelişmiş (Advanced) > Çevre Değişkenleri (Environment Variables) > Yol (Path) > Değişken Değeri: (Variable Value:)* …;dizininiz
7. [varsayılan değişken değerinin sonuna “;” işareti koyduktan sonra Java derleyici dosyanızın (javac.exe) olduğu dizini yazın.]
8. Eğer derleme sırasında “The name specified is not recognized … (Belirtilen isim, … olarak tanınmadı.)” hata mesajını alırsanız, Java derleyici dosyası (javac.exe) işletim sisteminizde yüklü değildir. J2SE platformunun kurulu olduğunu/doğru kurulu olduğunu kontrol edin.
9. Derlenen kodlarınızı çalıştırırken .class uzantısı yazılmaz.
10. Sadece bir main(…) metodu olan sınıflar çalıştırılabilir.
11. Kaynak kodunuzda birden fazla metot tanımlıysa, başlangıç kodunuzu main(…) metodunun olduğu sınıfın içine yazmalısınız. Java yorumlayıcısı bu main(…) metodunu çalıştırır.

Java'da Yorum Satırı

- Java kaynak kodunun içerisine istediğiniz yorumları yazabilmeniz için belli yol izlemeniz gerekmektedir.
- Java'da yorum satırlarını belirtme iki şekilde mümkün olur

1. /* **yorum** */ , slash - yıldızdan , diğer yıldız-slash arasına kadar istediğiniz yorumu yazabilirsiniz . Uzun satırlı yorumlarda bu yöntemi kullanabilirsiniz.
2. // **yorum** , tek satırlık yorum yapmak için idealdir. Kısa yorumlarınız için bu yöntemi kullanabilirsiniz.

Örnek:

Java kaynak kodu (Beykent.java):

```
public class Beykent{
public static void main(String[] args){
System.out.println("Beykent MYO, Bilgisayar Programı...");
}
}
```

VERI TIPLERI

Java dilinde kullanılan veri tipleri, sözdiziminin ve genel yapısının çoğunu aldığı C++ dilinden gelir. C++'ta ve dolayısıyla Java'da, temel veri tipleri ve sınıflar (referans/nesne veri tipleri) birbirinden ayrılmıştır. Temel veri tipleri operatörler tarafından yönetilir. Nesneler ise gönderilen mesajlar tarafından yönetilir. Gerçi operatörlerden birkaçı nesneleri de yönetir. Temel veri tiplerinin nesne olmadığı ve mesaj gönderemediği unutulmamalıdır. Ancak, temel veri tipleri nesneler içinde yer alabilir.

TEMEL VERI TIPLERI

- Mantıksal Veri Tipi (boolean)

boolean	true (doğru) / false (yanlış)	1 bit

- Tamsayı Veri Tipi (integers)

char	Unicode \u0000	16 bit	\u0000 ile \uffff arası
byte	İşaretli tamsayı	8 bit	-128 ile 127 arası
short	İşaretli tamsayı	16 bit	-32768 ile 32767 arası
int	İşaretli tamsayı	32 bit	-2147483648 ile 2147483647 arası
long	İşaretli tamsayı	64 bit	-9223372036854775808 ile 9223372036854775807 arası

- Gerçel Sayı Veri Tipi (float/double)

float	IEEE 754 floating point	32 bit	3.4028235E38 ile 1.4E-45 arası
double	IEEE 754 floating point	64 bit	1.79769313486231 57E308 ile 4.9E-324 arası

C ve C++'tan farklı olarak, Java'da mantıksal veri tipleri diğer veri tiplerinden ayrı olarak değerlendirilir ve aritmetik işlemlere girmez.
char dışındaki tüm tamsayı veri tipleri işaretlidir (eksi ve artı değer alabilir).
Unicode karakterler; sınıf, metot ve değişken ismi olarak kullanılabilir.

Temel *(Primitive)* Tipler

- Temel tipler stack alanında saklanırlar.

Temel tip	Boyut	Minimum	Maximum	Sarmalıyıcı Sınıf Tipi
boolean	–	–	–	Boolean
char	16- bit	Unicode 0	Unicode 2^{16}- 1	Character
byte	8- bit	-128	+127	Byte
short	16- bit	-2^{15}	$+2^{15}-1$	Short
int	32- bit	-2^{31}	$+2^{31}-1$	Integer
long	64- bit	-2^{63}	$+2^{63}-1$	Long
float	32- bit	IEEE754	IEEE754	Float
double	64- bit	IEEE754	IEEE754	Double
void	–	–	–	Void

ÖZEL KARAKTERLER

Genellikle ekrana yazdırılmayan ve özel işleve sahip olan karakterlerdir:

\b	Geri al (Backspace)
\t	Yatay sekme (Horizontal tab)
\n	Yeni satıra geç (Newline)
\r	Satır başı (Carriage Return)
\f	Form besleme (Form feed)
\’	Tek tırnak (Single quote)
\”	Çift tırnak (Double quote)
\\	Ters bölü (Backslash)
\ooo	Octal karakter/rakam
\uxxxx	Unicode (Hexadecimal) karakter/rakam

AYRILMIŞ KELİMELER:

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
super	switch	synchronized	this	throw	throws
transient	try	void	volatile	while	

İSİM VERME KURALLARI:

Belirleyiciler (identifiers) yani sınıf, metot ve değişken isimleri; alfanümerik karakterlerden, altçizgiden (_) ve dolar işaretinden ($) oluşabilir. Aşağıdaki kurallara uyulması gerekir:

1. İlk karakter rakam olamaz.
2. Ayrılmış kelimeler ve özel karakterler kullanılmaz.
3. "true" ve "false" kullanılmaz.
4. Kelimeler arasında boşluk bırakılmaz.

geçerli örnekler:

```
merhabaDunya;
merhaba_dunya;
merhaba_Dunya;
Merhaba;
merhaba;
_merhaba;
$dolar;
merhaba1;
```

geçersiz örnekler:

```
merhaba dunya;
1merhaba;
#diyez;
char;
merhaba!;
```

5. Değişken isimleri, küçük harfle başlar ve isimdeki her yeni kelime büyük harfle başlar.

Değişkenler, bir değer tutar. Tipleri ve isimleri (belirleyicileri) tanımlanır. Yerel değişkenler, kendilerine bir değer atanmadığı sürece tanımlanamazlar. Ancak, örnek isimleri varsayılan bir değer alır.

örnek:

```
double bakiye;
int x;
long hesapNo;
```

Belirleyiciler, Karakter.isJavaLetter() ifadesinin "true" sonucunun veren tüm unicode karakterleri kullanabilir. Böylece, bir Java programı yazan kişi kendi anadilindeki karakterleri (ASCII karakteri olmayan) de kullanabilir.

OPERATÖRLER

Operatörler, değerler/değişkenler üzerinde yapılan işlemleri (operations) temsil eden sembollerdir. Tek değer için (unary), iki değer için (binary) ve üç değer için (ternary) olan operatörler vardır.

örnek:

tek değerli: `4++`

iki değerli: `5 + 3`

üç değerli: `(a>5) ? 1 : 0`

ATAMA OPERATÖRÜ

Atama operatörü: =

Atama operatörü bir değişkene değer atamak için kullanılır.

Atanan değerin tipi ile değişkenin tipi uyumlu olmalıdır. Değişken tanımlama işlemi ile değer atama işlemi tek bir ifade ile gerçekleştirilebilir.

örnek:

```
x = 7;
int x = 7;
```

ARİTMETİK OPERATÖRLER

Toplama operatörü: +
Çıkarma operatörü: -
Çarpma operatörü: *
Bölme operatörü: /
Mod operatörü: %

örnek:

```
x = 7 + 5;                  // x = 12
int x = 33 % 8;             // x = 1
```

Aritmetik operatörler, aritmetik işlemler yapmak için kullanılır.

Değişken tanımlama işlemi ile aritmetik işlemler de tek bir ifade ile gerçekleştirilebilir.

ARTTIRMA VE AZALTMA OPERATÖRLERİ

Arttırma operatörü: ++
Azaltma operatörü: --

Arttırma operatörü, değeri 1 arttırır; azaltma operatörü, değeri 1 azaltır.

örnek:

```
int x = 0;
x++;                    // x = 1
x--;                    // x = 0
```

Bu operatörler, değerlerin/değişkenlerin önüne ya da sonuna gelebilirler. Sonuna geldiğinde önce ifadedeki işlem yapılır, sonra bu operatörlerin işlemi yapılır; önüne geldiğinde önce bu operatörlerin işlemi yapılır, sonra ifadedeki işlem yapılır.

örnek:

```
int x = 0;
int y = x++;                // x = 1, y = 0

int x = 0;
int y = ++x;                // x = 1, y = 1
```

KARŞILAŞTIRMA OPERATÖRLERİ

Küçüktür operatörü:	<	
Büyüktür operatörü:	>	
Küçük-eşittir operatörü:	<=	
Büyük-eşittir operatörü:	>=	
Eşittir operatörü:	==	
Eşit değildir operatörü:	!=	
Karşılaştırma operatörü:	instanceof	// sınıflar için

Bu operatörlerin işlem sonucu mantıksaldır ("true" ya da "false").

örnek:

```
13 < 7;                    // false
int x = 5;
x != 8;                    // true
```

MANTIKSAL OPERATÖRLER (BOOLEANS)

Değil (NOT) operatörü:	!
Ve (AND) operatörü:	&
Veya (OR) operatörü:	|
Sadece Biri (XOR) operatörü:	^
Koşullu Ve (Conditional AND) operatörü:	&&
Koşullu Veya (Conditional OR) operatörü:	||

Mantıksal operatörler sadece mantıksal değerlere uygulanırlar.

Değil operatörü, mantıksal tamamlayıcıdır; etki ettiği değer "true" ise "false", "false" ise "true" sonucunu verir. Ve operatörü her değer de doğruysa "true", en az biri yanlışsa "false" sonucunu verir. Veya operatörü en az bir değer doğruysa "true", her değer de yanlışsa "false" sonucunu verir. Hiçbiri değil operatörü tüm değerlerin yanlış olduğu durumda "true", diğer durumlarda "false" sonucunu verir. Koşullu operatörler, ilk değer için operatörün işlemi sonucu olarak "true" değerini verirse, diğer değerlerin kontrol edilmediği operatörlerdir. Bu özellikleri ile "Ve" ve "Veya" operatörlerinden ayrılırlar.

örnek:

```
boolean t = true;
boolean x = !t;          // x = false
```

BITWISE OPERATÖRLER

Bitwise Değil operatörü:	~
Bitwise Ve operatörü:	&
Bitwise Veya operatörü:	|
Bitwise Sadece Biri operatörü:	^
Bitwise Sola Kaydır operatörü:	<<
Bitwise Sağa Kaydır operatörü:	>>
Bitwise Sağa Kaydır ve Sıfır Ekle operatörü:	>>>

Bitwise operatörler, sadece bitler (1 ve 0) üzerine etki eden operatörlerdir.

Bu operatörler sadece çok özel durumlarda kullanılır. Dolayısıyla bunları bilmek ve kullanmak zorunda değilsiniz.

örnek:

```
3 & 2;                  // 0011 & 0010 sonuç: 0010
4 | 1;                  // 0100 | 0001 sonuç: 0101
3 << 2;                 // 0011 sonuç: 1100
4 >> 2;                 // 0100 sonuç: 0001
4 >>> 1;                // 0100 sonuç: 0010
```

BİLEŞİK ATAMA OPERATÖRLERİ

Aritmetik:	+=	-=	*=	/=	%=
Bitwise Kaydırma:	<<=	>>=	>>>=		
Mantıksal:	&=	|=	^=		

örnek:

```
x += 2;                 // x = x + 2; için kısa yol
x *= z + 3;             // x = x * (z + 3); için kısa yol
```

Atama operatörünün ile diğer operatörlerle birlikte işleme alındığı operatörler, bileşik operatörlerdir. Böylece ifadelerin değerlendirilmesi kısa yoldan yapılır.

ŞART OPERATÖRÜ

Şart operatörü: ? :

Mantıksal bir ifadenin sonucunun doğru olması durumundaki sonuç ile yanlış olması durumundaki sonucun sırayla tek bir ifadede sonuçlandırılmasıdır. if/else (eğer/değilse) deyiminin kısa yoludur.

Üç değere etki eden bir operatördür. İlk değer, mantıksal bir ifade olmalıdır:

```
MantıksalİfadeDoğruİçinSonuç : YanlışİçinSonuç
```

örnek:

```
int a = 5;
int b = (a > 2) ? 8 : 18;      // b = 8
// uzun yolu:          int a = 5, b;
                       if (a > 2){
                            b = 8;
                       else
                            b = 18;
                       }
```

OPERATÖRLERİN ÖNCELİĞİ

Operatörlerin önceliği, bir ifadede yer alan birden fazla sayıdaki operatörün hangi sırayla işleme alınacağını kontrol eder. Aynı öncelik sırasına sahip operatörler için işlem sırası soldan sağa doğrudur. Sadece atama operatörü, bileşik atama operatörleri ve şart operatörü için bu yön sağdan soladır.

Fazla	↑	.	{}	[]	()
		++	--	!	~
		*	/	%	
		+	-		
		<<	>>	>>>	
		<	>	<=	>=
		==	!=		
		&			
		^			
		|			
		&&			
		||			
		? :			
Az		=	*=	+=	vb.

Java'nın Başarılı Olmasındaki Sebepler

- **Nitelikli bir programlama dili olması**
  - C++ da olduğu gibi bellek problemlerinin olmaması .
  - Nesneye yönelik (Object - Oriented) olması
  - C/C++/VB dillerinin aksine dinamik olması .
  - Güvenli olması .
  - Internet uygulamaları için elverişli **(Applet, JSP, Servlet, EJB, Corba, RMI).**
- **Platform bağımsız olması : bir kere yaz her yerde çalıştır**

Çöp Toplayıcı *(Garbage Collector)*

- Bir programın çalışma durumunda ortaya çıkan ve sonradan kullanılmayan (gereksiz) nesneleri bulur ve onları yok eder (*destroy*).
- Bellek yönetiminin (*memory management*) yükü, kodu yazan kişiden Java'ya geçmiş olur
- Diğer dillerde, örneğin C++ da , oluşturulan nesnelerin yok edilme sorumluluğu kodu yazan kişiye aittir.
- Çöp toplayıcısı(*garbage collector*) JVM'in yazılışına (*implementation*) göre değişkenlikler gösterebilir.

Herşey Nesne - 1

- Java'da herşeye nesne olarak davranırız. Herseyin nesne olmasına rağmen nesneleri yönetmek için " referanslar" kullanılır .

Örnek : Diyelim ki elimizde bir maket uçak (nesne olarak düşünün) ve bu maket uçağa ait bir de kumanda (referans) olduğunu düşünelim.

Bu maket uçağı havada sağa sola döndürmek için elimizdeki kumanda cihazını kullanmak zorundayızdır; benzer şekilde havalandırmak veya yere indirmek için de kumanda cihazından faydalanırız. Burada dikkat edilmesi gereken unsur kumanda cihazından çıkan emirlerin maket uçağı tarafından yerine getirilmesidir.

Herşey Nesne - 2

- Elimizde uzaktan kumandanın (referans) olması, maket uçağımızın (nesne) olduğu anlamına gelmez .
- Uzaktan kumandamız (referans) da tek başına hayatı sürdürebilir.

```
String kumanda ; // kumanda referansı şu an için
                 //String bir nesneye bağlı değil.
```

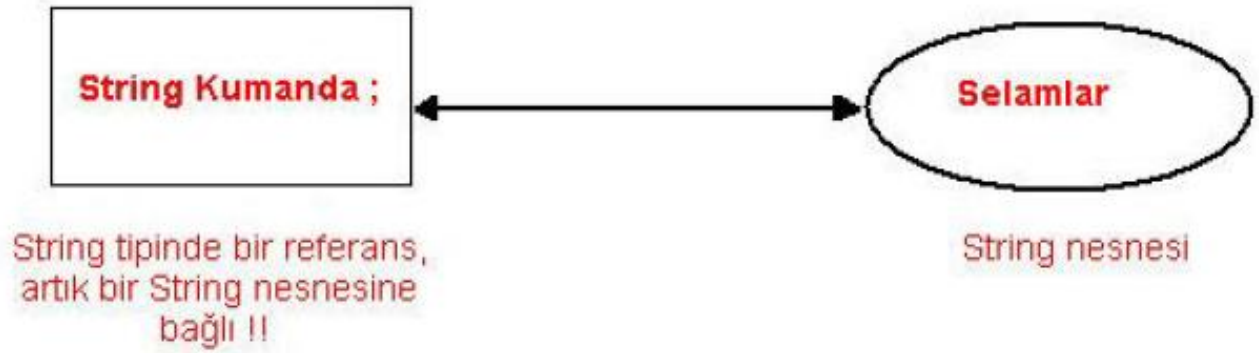
String tipinde bir referans, şu an için herhangi bir String nesnesine bağlı değil

Herşey Nesne - 3

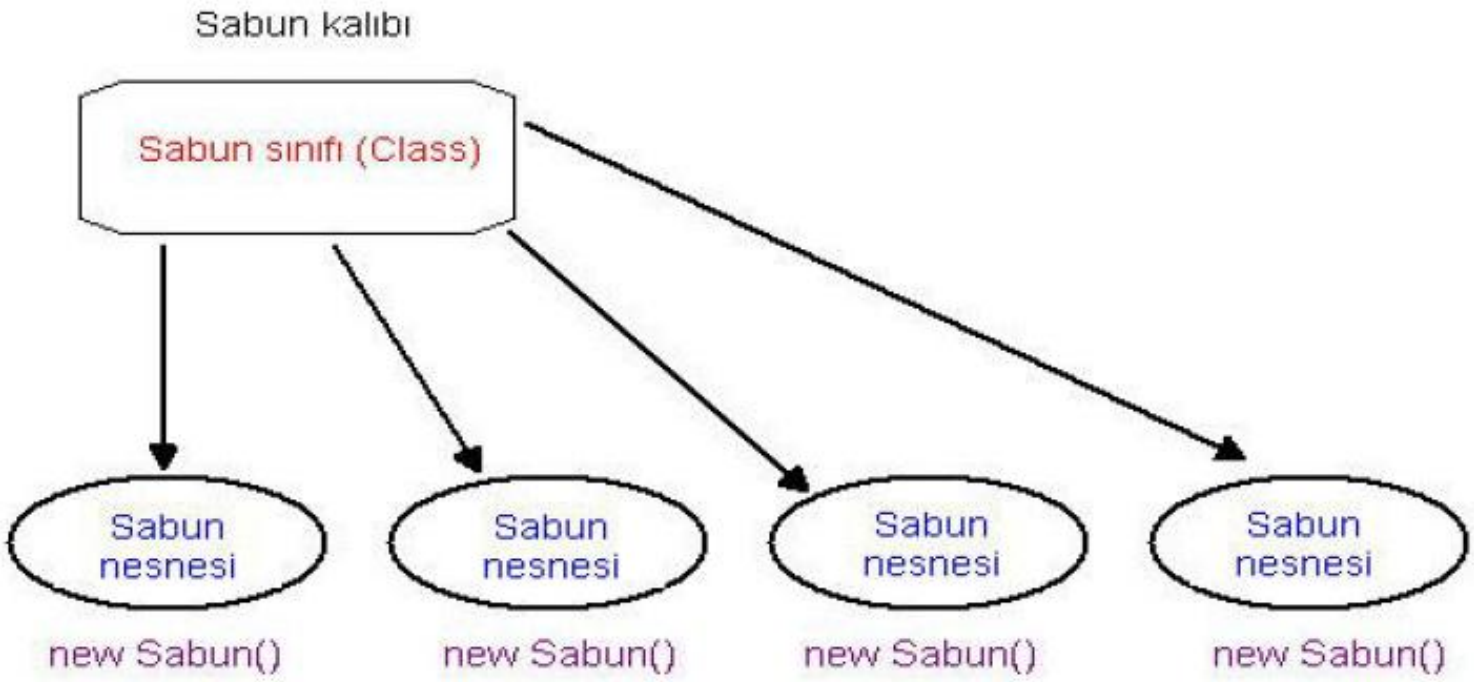
- Bir referansa mesaj göndemek istiyorsak onu bir nesneye bağlamamız gerekir.

```
•String kumanda= new String("Selamlar") ;
```

```
•String kumanda="Selamlar" ;
```



Sınıf Nedir? Nesne Nedir?




```
Sabun   s = new Sabun();
```

Java'da Depolanan Veriler Nerede Durur - 1

- **Stack** : Bulunduğu nokta RAM'dır... Stack üzerinde referansların kendileri bulunur.
- **Heap** : Burası genel amaçlı bir havuzdur . Nesnelerin kendisi bu alanda durur.
- **Statik Alan** : Bu alan RAM'de bulunur. Statik alanda yer alan veriler , programın çalışması süresince orada yaşarlar. Nesnelerin kendileri bu alanda yer almazlar.

Java'da Depolanan Veriler Nerede Durur - 2

- **Non-RAM Bellek :** Bazı durumlarda uygulamaların içerisinde oluşturduğumuz nesnelerin,uygulama sonlandıktan sonra bile varlıklarını sürdürmelerini isteriz.
  1. **Akışkan Nesneler (streamed objects) :** Bu nesneler, genellikle ağ(network) üzerindeki başka bir makineye gönderilmek üzere bayt ırmaklarına dönüştürülürler.
  2. **Kalıcı Nesneler (persistent objects) :** Bu nesneler kendi durumlarını(*state*) saklarlar ve diskimizde saklanırlar. Kendi durumlarını saklamaktan kasıt ise özelliklerinin (*attribute*) değerlerinin korunmasıdır.

Sarmalayıcı *(Wrapper)* Sınıflar

- Temel tiplerin birer adet sarmalıyıcı (*wrapper*) sınıfları bulunur.

```
char c = 'x' ; // temel tip
```

```
Character C = new Character(c); // sarmalayıcı sınıf
```

Geçerlilik Alanı *(Scope)* - 1

```
{
  int a = 177;
  /* sadece a mevcut*/
  {
    int b = 196;
    /*  a ve b mevcut */
  }
  /* sadece a mevcut */
  /* b "geçerlilik alanının dışına çıktı " */
}
```

Geçerlilik Alanı *(Scope)* - 2

- C ve C++ doğru ama Java'da yanlış olan bir ifade

```
{ // dış alan

  int a = 12;

  { // iç alan

    int a = 96; /* java'da yanlış ama C ve C++ doğru */

  } // iç alanın sonu

} //dış alanın sonu
```

Nesneler İçin Geçerlilik Alanı *(Scope of Objects)*

```
if (true){

   String s = new String("Selamlar");

}  /* geçerlilik alanının sonu*/
```

• Geçerlilik alanının sonunda **String** nesnesi "Çöp Toplayıcısı" (*Garbage Collector*) tarafından bellekten silinecektir.

Yeni Sınıf Oluşturma

```
public class YeniBirSinif  {

    .....



}
```

Alanlar - 1

- Alanlar, temel bir tip veya sınıf tipinde olabilir.

```
public class YeniBirSinif  {

   public int i;

   public double d;

   public boolean b;

}
```

Alanlar - 2

Temel (*primitive*) Tip	Mevcut değer (*Default value*)
`boolean`	`false`
`char`	`'\u0000' (null)`
`byte`	`(byte)0`
`short`	`(short)0`
`int`	`0`
`long`	`0L`
`float`	`0.0f`
`double`	`0.0d`

Alanlar - 3

```
public class YeniBirSinif  {
      public int i = 5 ;
      public double d = 3.23;
      public boolean b = true ;
}
```

Alanlar - 4

```
YeniBirSinif ybs = new YeniBirSinif();
```

Alanlara Ulaşım

- **Nesnenin alanlarına ulaşmak için "." (nokta) kullanılır.**
- Bu alanların erişim belirleyicileri
  - public
  - private
  - protected
  - friendly

  olabilir.

```
YeniBirSinif   ybs = new YeniBirSinif();
ybs.i ;
ybs.d ;
ybs.b ;
```

Alanlara Değer Atama

```
YeniBirSinif ybs = new YeniBirSinif();
ybs.i = 5;
ybs.d = 5.3;
ybs.b = false;
```

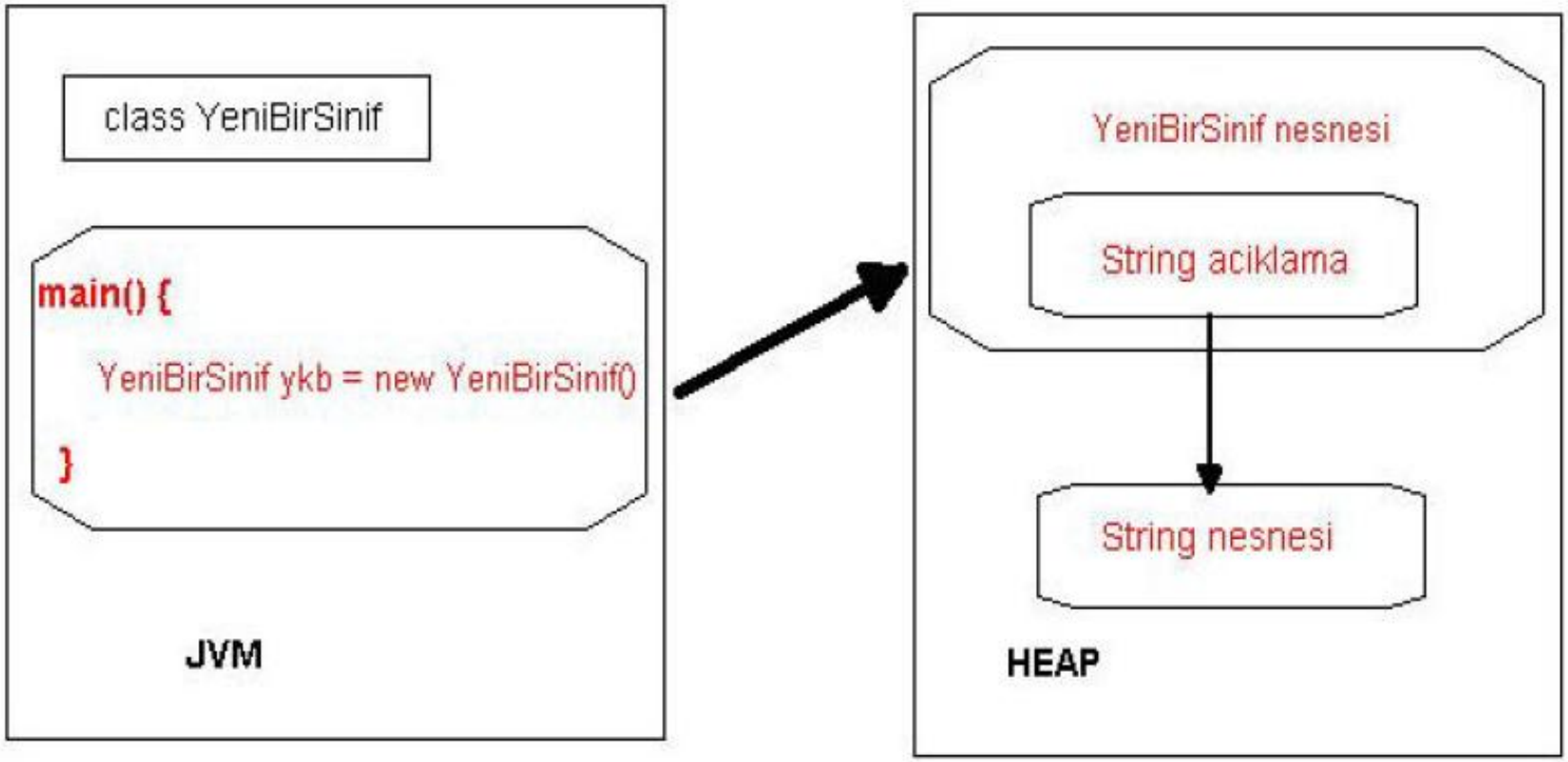
Sınıf Tipindeki Alanlar - 1

```
public class YeniBirSinif  {

  public int i;
  public double d;
  public boolean b;
  public String aciklama = new  String("aciklama");

}
```

Sınıf Tipindeki Alanlar - 2



Yordamlar *(Methods)* - 1

```
dönüşTipi  YordamIsmi( /* parametre listesi */ ) {
  /* Yordamın gövdesi */
}
```

- **dönüşTipi**= Yordamların iki şansı vardır:
  - Değer döndürürler
    - Temel (*primitive*) bir tipde değer (int, double, short vb..)
    - Sınıf tipinde bir değer (String, Double, Short vb...)
  - Değer döndürmezler = **void**

Yordamlar *(Methods)* - 2

- **`yordamIsmi`** = Java'nın kendisine ait olan sözcükler (**if , else , import , class , return.**.vb) ve Türkçe karakterler haricinde istenilen isim kullanılabilir. Ancak, yordamlar bir eylem içerdikleri için, yordam isimlerinin de bir eylemi belirtmesi tercih edilir.

- Örneğin:
  - **`sayiSirala()`**
  - **`enBuyukSayiBul()`**
  - **`sqlCalistir()`**

- **`parametre listesi`**= Yordam içerisinde işlemler yapabilmek için gerekli olan parametreler. Bu parametreler temel tipte veya sınıf tipinde olabilirler.

- **`Yordam gövdesi`** = Bu kısım kodu yazan kişinin yaratıcılığına bağlı olarak değişir.

Yordam *(Method)* Örneği - 1

```
int boyutDondur(String kelime) {
    return kelime.length() ;
}  // yordamın sonu
```

Yordam *(Method)* Örneği - 2

```
String elmaHesapla(int elmasayisi) {
   return  new String("elma sayisi = "
                                + elmasayisi*2);
}  // yordamın sonu
```

Yordam İçindeki Yerel Değişkenlerin İlk Değerlerini Alması

```
void hesapla(String kelime , int kdv ) {
  int sondeger = 0;
  int kelimeboyut = 0 ;
  int toplamboyut ; // Hatalı !!
  toplamboyut++ ;   // Hatalı !!
  kelimeboyut = kelime.length();
  sondeger = kelimeboyut + kdv ;
}
```

İlk Java Programı - 1

```
public class Selam {
   public static void main(String args[]) {
       System.out.println("Selamlar !");
    }
}
```

- **public class Selam :** Bu kısım da yeni bir sınıf oluşturuyor...

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

  - Java'da bir sınıfın tek başına çalışması isteniyorsa (*standalone*) bu yordam yazılmak zorundadır . Bu yordam sınıflar için bir başlagıç noktasıdır.
  - **`static`** yordamlar nesneye bağımlı olmayan yordamlardır. Bu yordamı kullanmak için, ilgili sınıfa ait bir nesne oluşturma zorunluluğu yoktur.

Diziler *(Arrays)*

- **`main()`** yordamı parametre olarak *String* sınıfı tipinde dizi alır, bu *String* sınıfı tipindeki dizinin içerisinde, konsoldan Java uygulamasına gönderilen parametreler bulunur .
  - `args[0]` : konsoldan girilen 1. parametre değerini taşır …
  - `args[1]` : konsoldan girilen 2. parametre değerini taşır …
  - `args[n-1]` : konsoldan girilen n. parametre değerini taşır …
- Java'da diziler sıfır'dan başlarlar. Diziler ilerleyen bölümlerde yoğun bir şekilde incelenecektir.

- `System.out.println("Selamlar !")`
  - Bu komut satırı, bilgileri konsola (ekrana) basmamızı sağlar. Java'nın dokümanlarına bakarsak;
  - ***System*** sınıfı altında static bir alan olan **out** alanının mevcut olduğunu görüyoruz. Bu yüzden ***System*** sınıfını oluşturmak zorunda değiliz (`new System()`).
  - **out** alanı bize ***PrintStream*** nesnesi oluşturur ve ***PrintStream*** nesnesinin `println()` methodu ile bilgileri konsola(ekrana) bastırırız.

Dizin Yapısı

```
JAVA_KURULUM_DIZINI
                    |
                    |
                    |__bin (dizin)
                    |
                    |__demo (dizin)
                    |
                    |__include (dizin)
                    |
                    |__jre
                    |     |
                    |     |__bin(dizin)
                    |     |
                    |     |__lib (dizin)
                    |     |        |
                    |     |        |__rt.jar (class dosyalarının bulunduğu jar dosyası)
                    |     |
                    |     |
                    |__lib (dizin)
                    |     |
                    |     |__tools.jar (Faydalı sınıfların bulunduğu jar dosyası)
                    |     |
                    |     |
                    |     |
                    |__src.jar (Kaynak kodlarin bulunduğu jar dosyası)
```

Hata Mesajları

İdeal bir durumda, yazdığımız program bilgisayarda ilk denememizde hiç hata vermeden tam istediğimiz gibi çalışabilir. Ama bu büyük ihtimalle hiç olmayacağı için karşılaşacağımız hata çeşitlerini bilmemizde yarar vardır. Üç hata çeşidi ile karşılaşabiliriz: Derleme sırasındaki hatalar, çalıştırma sırasındaki hatalar ve mantık (ya da tasarım) hataları.

Derleme hataları program derlenirken ortaya çıkan hatalardır. Bunlar aynı zamanda ***syntax hataları*** olarak da bilinir. Programınızı yazarken yazım hataları yaparsanız (imla hatası gibi), Java'nın kurallarına uymazsanız (örneğin ; ya da { ve } gibi işaretleri yanlış yerlerde kullanmak gibi) derleme hataları alırsınız. ***Çalıştırma hataları*** program derlendikten sonra çalıştırılırken alınan hatalardır. ***Mantık hataları*** da algoritma tasarımı sırasında ya da algoritmayı programa kodlarken yapılan bir hatadan kaynaklanır.

Derleme hataları

Bunlar bulması ve düzeltmesi en kolay hatalardır. Java'da en sık yapılan yazım hatalarından birkaçını sıralayalım.

Java'da komut içeren her ifade ';' işareti ile bitirilmelidir. Bu Java'nın ifadeleri birbirinden ayırabilmesi için gereklidir. Eğer ';' unutulursa Java derleyicisi hata mesajı verecektir.

Açıklamalar mutlaka '//' ile başlamalıdır. Eğer açıklama bir sonraki satırda sürecekse bu satırın başına da '//' konmalıdır.

Programların bölümlerinin başlangıç ve sonlarını belirleyen '{' ve '}' işaretlerini unutmak ya da yanlış kullanmak da en yaygın hatalardır.

Çalıştırma hataları

Programınızı syntax hatalarından temizledikten sonra çalıştırdığınızda programın çıktısı yerine hata mesajı alırsanız ya da programınız hiç sonuç vermeden ekran kilitlenmiş gibi boş kalırsa çalıştırmada bir hata var demektir. Bu tip hataların nedeni daha zor bulunur, çünkü nedenler programın içeriğine göre çok çeşitlidir. Örneğin programınızda bir bölme işlemi vardır ve programın çalışma sırasında bölen değer sıfır oluyordur. İşte bu durumda sıfıra bölme işlemi çalıştırma hatası verecektir. Bu tip hataları programlama yeteneğiniz arttıkça bulmanız ve düzeltmeniz kolaylaşacaktır.

Mantık hataları

Mantık hataları genelde derleme ve çalıştırma hatalarından kurtulduktan sonra ortaya çıkar. Bu aşamada programınız çalışır ve bir çıktı verir. Ancak çıktıyı incelediğinizde bunun sizin istediğiniz çıktı olmadığını görürsünüz. Örneğin çıktılar düzgün bir sırada alt alta çıkmamıştır, ya da rakamlar hatalı hesaplanmıştır ya da program eksik çıktı vermiştir. Bu tip hataları düzeltmek için programın bazı kısımlarını yeniden yazmak hatta bazen algoritmayı değiştirmek gerekebilir.