JAVA NEDİR ?

Java ™ platformu bilgisayar ağının varlığı da göz önüne alınarak uygulamaların/programların farklı işletim sistemleri üzerinde çalıştırılabilmesi düşüncesiyle geliştirilmiş yeni bir teknolojidir. Java teknolojisi kullanılarak aynı uygulama farklı ortamlarda çalıştırılabilir. Örneğin kişisel bilgisayarlarda, *Macintosh* bilgisayarlarda, üstelik cep telefonlarında... (yorum ekle)

Java ™ platformu hem programlama dili, hem de bir ortam olarak düşünülebilir. Programlama dili olarak, açık kodlu, nesneye yönelik (*object-oriented*), güvenli, sağlam, İnternet için elverişli bir teknolojidir denilebilir. Ortam olarak da orta katman(*middleware*) teknolojiler bulmak mümkündür. (yorum ekle)

Gerek Java programlama dili, gerekse bu dile bağlı alt teknolojiler, VB™ veya Borland Delphi™ gibi sadece belirli bir firma tarafından geliştirilmiş ürünler değillerdir. Java ve bu dile bağlı alt teknolojiler, *Sun MicroSystems* tarafından tanımlanmış belirtimlerden (*specifications*) oluşmaktadır. Bu belirtimlere sadık kalan her yazılım firması Java Sanal Makinası, kısaca JVM (*Java Virtual Machine*), veya Java programlama diline bağlı alt teknolojiler yazabilir (örneğin *Application Server* - Uygulama Sunucusu). Eğer bu belirtimlere sadık kalınmayıp standart dışı bir JVM veya Java programlama diline bağlı alt teknolojiler yazılmaya kalkışılırsa hukuki bir suç işlenmiş olur. (yorum ekle)

Peki belirtim (*specifications*) ne demektir? *Sun MicroSystems*, JVM veya Java programlama diline bağlı alt teknolojiler yazmak için belirli kurallar koymuştur; bu kurallar topluluğuna "belirtimler" denir. Örneğin biraz sonra ele alınacak olan çöp toplama sistemi (*garbage collector*)... (yorum ekle)

Çöp toplama sistemi daha önceden oluşturulmuş, ancak şu an için kullanılmayan ve bellekte boşu boşuna yer işgal eden nesneleri belirleyerek otomatik olarak siler. Böylece Java programcısı "*acaba oluşturduğum nesneyi bellekten silmiş miydim?*" sorusunu sormaktan kurtulurlar, ki bu soru C++ programlama dillinde uygulama yazan kişilerin kendilerine sıkça sorması gereken bir sorudur. Şimdi bir yazılım firması hayal edelim, adının ABC yazılım firması olduğunu varsayalım. Bu firma, eğer bir JVM yazmak istiyorsa, bu çöp toplama sistemini, oluşturdukları JVM'in içerisine yerleştirmeleri gereklidir. Çünkü *Sun MicroSystems*'ın belirtimlerinde, çöp toplama sistemi koşuldur! Eğer ABC firması üşenip de çöp toplama sistemini, oluşturdukları JVM'in içerisine yerleştirmezse hukuki bir suç işlemiş olur. (yorum ekle)

Şu anda en yaygın kullanılan JVM'ler, IBM ve *Sun Microsystems*'ın üretilmiş olan JVM'lerdir; ayrıca, HP, *Apple* gibi bir çok firmanın üretmiş oldukları JVM'ler de bulunmaktadır. (yorum ekle)

1.1. **Java ile Neler Yapılabilir?**

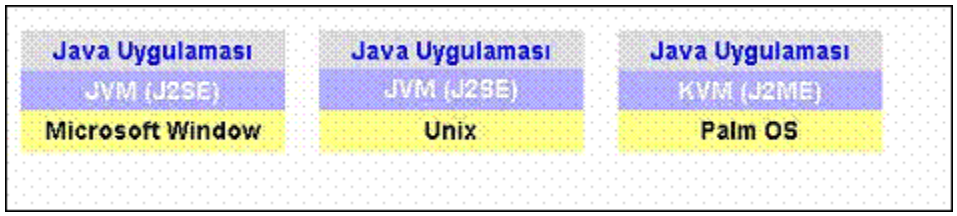
Java diliyle projeler diğer programlama dillerine göre daha kolay, sağlıklı ve esnek şekilde yapılması mümkün olur. Kısaca göz atılırsa Java diliyle,

- GUI (7Grafiksel Kullanıcı Arayüzü) uygulamaları, *Applet*'ler
- Veri tabanına erişimle ilgili uygulamalar
- *Servlet*, Jsp (Web tabanlı uygulamalar).
- Dağınık bileşenler (*Distributed components*) (örneğin EJB, RMI, CORBA).
- Cep telefonları, *Smart* kartlar için uygulamalar.
- Ve daha niceleri…

için uygulamalar yazmamız mümkündür. (yorum ekle)

1.2. **Java Nasıl Çalışır?**

Java uygulamaları JVM tarafından yorumlanır; JVM, işletim sisteminin üstünde bulunur. Bu nedenle, Java uygulamaları farklı işletim sistemlerinde herhangi bir değişiklik yapılmadan çalışır. Böylece Java programlama dilinin felsefesi olan "Bir kere yaz her yerde çalıştır" sözü gerçekleştirilmiş olunur. (yorum ekle)

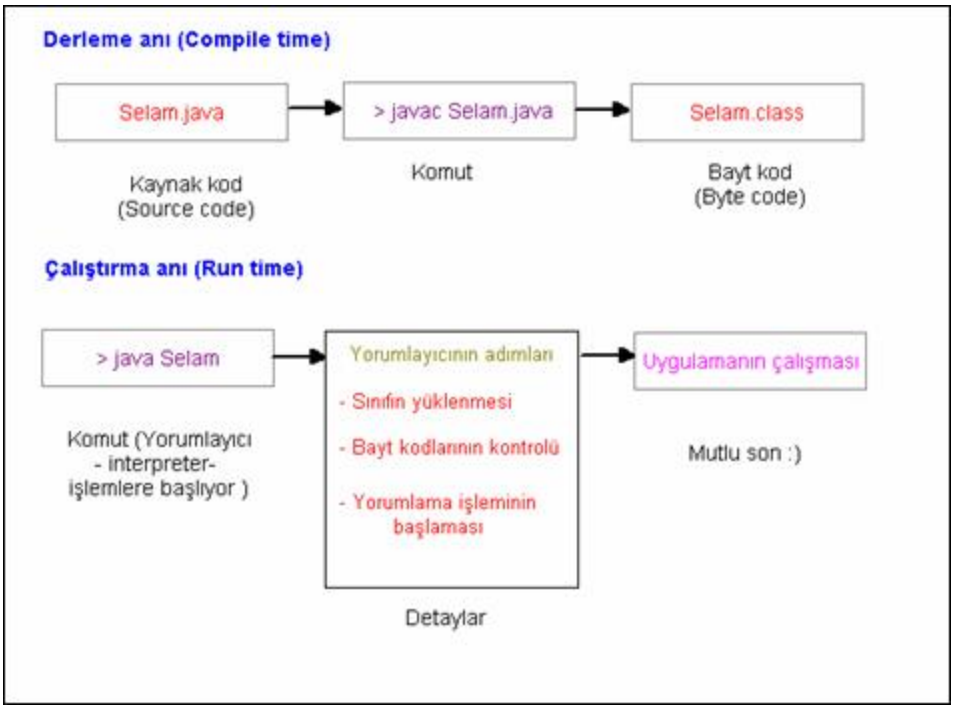


Şekil-1.1. İşletim sistemleri ve JVM'in konumu

Şekil-1.2.'de Java kaynak kodunun nasıl çalıştırıldığı aşamalarıyla gösterilmiştir. *Byte* (sekizli) koduna çevrilen kaynak kod, JVM tarafından yorumlanır ve uygulama çalıştırılmış olur. Kısa bir Java uygulaması üzerinde olayları daha ayrıntılı bir şekilde incelenirse... (yorum ekle)

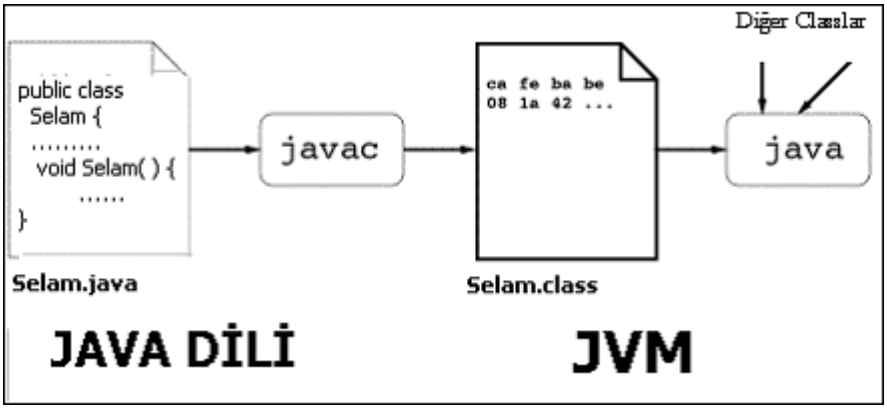
Örnek: *Selam.java* (yorum ekle)

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



Şekil-1.2. JAVA kodunun çalıştırılma aşamaları

Yukarıda yazılan uygulamanın hangi aşamalardan geçtiği şekil üzerinde ilerleyen paragraflarda gösterilmiştir:



Şekil-1.3. Selam.java'nın geçtiği aşamalar

Yazılan kaynak kodları ilk önce **`javac`** komutuyla derlenir; ardından **`java`** komutuyla çalıştırılır. Fiziksel dosyanın içerisindeki her sınıf (*class*) için fiziksel olarak bir ***.class*** dosyası oluşturulur.

1.3. **JAVA Sınıflaması**

Java ortamı 4 ana sınıf altında toplanmıştır:

- Standart Java
- Komple (*Enterprise*) Java
- Gömülü cihazlar için Java (*embedded devices*)
- XML Teknolojileri
- Diğer Teknolojiler (yorum ekle)

1.3.1. **Standart Java**

- J2SE (*Java 2 Standart Edition*)
- J2SE Bileşenleri
  - Yardımcı Teknolojiler (*Assistive Technologies*)
  - Sürükle ve Bırak (*Drag and Drop*)
  - Java Erişim Köprüsü (*Access Bridge*)
  - *JavaBeans* Teknolojisi
  - *JavaBean* Etkinleştirme Sistemi (*JavaBean Activation Framework*) JAF 1.0.2
    - *Javadoc* Aracı
  - Java Altyapı sınıfları (*Java Foundation Classes (JFC) / Swing*)
  - Java *HotSpot* Sanal Makinası (*Virtual Machine*)
  - Java Platformu Ayıklayıcı Mimarisi (*JPDA-Java Platform Debugger Architecture*)
  - Windows XP için Java Uyum-eki (*Java Plug-in for Windows XP*)
  - Java 2D API
  - Java *Web Start*
  - JDBC Teknolojisi
  - Takılıp çıkarabilir (*Pluggable*) *Look and Feel*
  - Uzak Yordam Çağırımı (*Remote Method Invocation*) (RMI)
  - Güvenlik (*Security*)
- J2SE Seçimlik Paketler
  - InfoBus
  - Java Gelişmiş Görüntüleme (*Advanced Imaging*)
  - Java Kimlik Belirleme ve Yetkilendirme Servisi (*Auth. and Auth. S.*) (JAAS)
  - Java İletişim (*Communication*) API (JCA)
  - Java Şifreleme Uzantısı (*Cryptography Extension*) (JCE)
    - Java Veri Nesneleri (*Data Objects*)
  - Java Yardım Teknolojisi (*Help Technology*)
  - Java Ortam (*Media*) API leri
  - Java Ortam Sistemi (*Media Framework*) (JMF)
  - Java İsimlendirme ve Dizin Arabirimi (*JNDI-Java Naming and Directory Interface*)
  - Java Güvenli Soket Uzantısı (*JSSE-Java Secure Socket Extension*)
  - Java Konuşma (*Speech*) API'si

- Java 3D API (yorum ekle)

1.3.2. **Enterprise Java**

- J2EE (*Java 2 Enterprise Edition*)
- CORBA Teknolojisi
- ECperf Teknolojisi
- Komple (*Enterprise*) JavaBeans Teknolojisi
- Kontaynerler için Java Yetkilendirme Kontratı (*Java Authorization Contract for Containers*) (Java ACC)
- Java IDL
- *JavaMail* API
- Java Mesajlaşma Servisi (*Message Service*) (JMS) API
- *JavaServer* Yüzleri (*Faces*)
- *JavaServer* Sayfaları (*Pages*)
- Java *Servlets*
- JDBC Teknolojisi
- J2EE Bağlayıcı Mimarisi (*Connector Architecture*)
- Hareketler (*Transactions*) (yorum ekle)

1.3.3. **Gömülü Cihazlar İçin Java (*Embedded Devices*)**

- Java 2 Platform, *Micro Edition* (J2ME Teknolojisi)
- Bağlı Aygıt Konfigurasyonu (*Connected Device Configuration*) (CDC)
- Sınırlı Bağlanmış Aygıt Konfigurasyonu (CLDC-*Connected Limited Device Configuration*)
- C Sanal Makinası (CVM-*C Virtual Machine*)
- K Sanal Makinası (KVM- *K Virtual Machine*)
- Kişisel Java (*Personal*Java)
- Java Card
- JavaPhone API
- Java TV API
- *Jini Network Technology*
- Gezgin Bilgi Aygıt Profili (MIDP-*Mobile Information Device Profile*) (yorum ekle)

1.3.4. **XML Teknolojileri**

- XML İlişkilendirilmesi için Java Mimarisi (JAXB-*Java Architecture for XML Binding*)
- XML-Tabanlı RPC için JAVA API'si (JAX-RPC-*Java API for XML-Based RPC*)
- XML Mesajlaşması için JAVA API'si (JAXM-*Java API for XML Messaging*)
- XML İşlemleri için JAVA API'si (JAXP-*Java API for XML Processing*)
- XML Kayıtları için JAVA API'si (JAXR-*Java API for XML Registries*) (yorum ekle)

1.3.5. **Diğer Teknolojiler**

- Araç Ürünler
  - MIF *Doclet*
  - Sun ONE Stüdyo (*Studio*)
- AĞ (*NetWork*) Ürünleri
  - Sertifikalı JAIN API Ürünleri (*JAIN API Certified Products*)
  - Java Dinamik Yönetim Seti (*Java Dynamic Management Kit*)
  - Java Yönetim Uzantısı (JMX-*Java Management Extensions*)
  - Java MetaData Arabirimi (JMI-Java Metadata Interface)
  - Java Paylaşılan Veri Araç Takımı (*Java Shared Data Toolkit*)
  - Java *Spaces* Teknolojisi
  - Servis Sağlayıcılar için Java Teknolojisi (*Java Technology for Service Providers*)
  - Jini Ağ Teknolojisi (*Network Technology*)
  - JXTA Projesi
  - J2ME Platformu için JXTA Projesi (*Project JXTA for J2ME Platform*)
  - Sun *Chili!Soft* ASP (yorum ekle)

1.4. **Gelişim Evreleri**

Tablo-1.1. JAVA'nın gelişim evreleri (yorum ekle)

1995	· Java teknolojisinin ilk çıkış yılı; ilk olarak *Applet* teknolojisinin dikkat çektiği yıllar.
1996	· Java Geliştirme Seti (JDK) v1.0 çıkartıldı. Temel seviyeli işlevleri içeren bir versiyon (örneğin soket programlama, Girdi/ Çıktı (Input/Output), GUI (*Graphical User Interface*- Grafik Kullanıcı Arabirimi)
1997	· JDK 1.1 çıkartıldı. Bu sürümde Java GUI, veritabanı erişimi için JDBC, dağınık nesneler için RMI ve daha birçok yeni gelişmeler eklendi.
1998	· JDK 1.2 çıkartıldı. · JFC/*Swing* yayınlandı- aynı yıl içersinde **http://java.sun.com** İnternet adresinden 500,000+ adet indirme (*download*) gerçekleştirildi.
1999	· Java teknolojisi J2SE, J2EE ve J2ME olarak 3'e bölündü. · Java *HotSpot* (performans arttırıcı) yayınlandı. · *JavaServer Pages* (JSP) teknolojisi yayınlandı. · J2EE platformu yayınlandı.

	· Linux üzerinde J2SE platformu yayınlandı.
2000	· JDK v1.3 çıkartıldı. · Java APIs for XML teknolojisi yayınlandı.
2002	· JDK v1.4 versiyonu çıkarıldı (Merlin projesi). · Java API for XML *binding* yayınlandı.
2003	· 2003 yılının sonuna doğru JDK v1.5 versiyonun çıkarılması planlanmaktadır (*Tiger* projesi).

1.5. **Java'nın Başarılı Olmasındaki Anahtar Sözcükler**

Œ. Nitelikli bir programlama dili olması
- · C/C++ da olduğu gibi bellek problemlerinin olmaması.
- · Nesneye yönelik (*Object Oriented*) olması.
- · C/C++/VB dillerinin aksine doğal dinamik olması.
- · Güvenli olması.
- · İnternet uyg. için elverişli olması. (*Applet*, JSP, *Servlet*, EJB, *Corba*, RMI). (yorum ekle)

. Platform bağımsız olması: **Bir kere yaz her yerde çalıştır**! (yorum ekle)

1.6. **Çöp Toplayıcı (*Garbage Collector*)**

Çöp toplayıcı devamlı olarak takip halindedir; Java uygulamasının çalışma süresince ortaya çıkan ve sonradan kullanılmayan gereksiz nesneleri bulur ve onları temizler. Böylece bellek yönetim (*memory management*) yükü tasarımcıdan JVM'e geçmiş olur. Diğer dillerde, örneğin C++ programlama dilinde, oluşturulan nesnelerin yok edilme sorumluluğu tasarımcıya aittir.
Çöp toplayıcının ne zaman ortaya çıkıp temizleme yapacağı belirli değildir; eğer bellekte JVM için ayrılan kısım dolmaya başlamışsa çöp toplayıcı devreye girerek kullanılmayan nesneleri bellekten siler. Çöp toplayıcısı JVM'in gerçeklenmesine göre farklılık gösterebilir; nedeni, her JVM üreticisinin farklı algoritmalar kullanmasından ileri gelmektedir. (yorum ekle)

1.7. **Java'da Açıklama Satırı (*Comment Line*)**

Java kaynak kodunun içerisine kod değeri olmayan açıklama yazılabilmesi için belirli bir yol izlenmesi gerekir. Şunu hemen belirtelim ki, uygulamalarımız içerisinde yorum satırları sık sık kullanılacaktır. Java uygulamaları içerisinde açıklama/yorum satırları koymak için iki farklı yöntem kullanılır: (yorum ekle)

- `/* yorum */`
  Bölme işareti-yıldız ve yıldız-bölme işareti arasına istenilen açıklama yazılabilir. Genel olarak uzun açıklamalarda bu yöntem kullanılır. (yorum ekle)
- `// yorum;`
  Tek satırlık açıklama yapılması için kullanılır. Kısa açıklamalar için bu yöntem kullanılabilir. (yorum ekle)

1.8. **Herşey Nesne**

Her programlama dilinin kendine has veri yönetim şekli bulunur. Java platformunda çalışan bir uygulamada, çalışma sırasında nesneler oluşturulur. Burada ki soru bizim nesnelere doğrudan olarak mı? Yoksa onlara dolaylı bir şekilde mi bağlantı sağlayıp kullandığımızdır. Java programlama dilinde herşeye nesne olarak davranılır. Herşeyin nesne olmasına karşın bu nesnelerin kullanılması için referanslara gereksinim duyulur.
Örneğin, elimizde bir maket uçağı olsun; nesne olarak düşünelim... Bu maket uçağı denetlemek amacıyla bir de kumanda cihazının, yani referansın olduğunu düşünelim. Bu maket uçağı havada sağa veya sola döndürmek için elimizdeki kumanda cihazını kullanmak zorundayız; benzer şekilde havalandırmak veya yere indirmek için kumanda cihazından yararlanırız. Burada dikkat edilmesi gereken unsur kumanda cihazından çıkan emirlerin maket uçağı tarafından yerine getirilmesidir.
Elimizde bir kumanda cihazının bulunması, maket uçağımızda olması anlamına gelmez. Her durumda bir referansı tek başına da tanımlanabilir. İşte kanıtı, (yorum ekle)

Gösterim-1.1:

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

,

Şekil-1.4. Referans tanımı

Burada yapılan olay sadece referans oluşturmaktır. Eğer bu referansa mesajlar göndermeye kalkışılırsa ne olur? Şöyle düşünelim, elimizde fazla para olmadığı için önce kumanda cihazını aldık ve eve getirdik; ama, dikkat ediniz, henüz ortalıkta maket uçağımız filan yok! Bu aşamada, bu kumanda cihazı kullanılarak komutlar gönderilse neler olur? Hiçbir şey... Çünkü bu kumandanın bağlı olduğu bir maket uçak ortalıkta yoktur. Java programlama dilinde de olaylar böyle gelişir. Yalnızca referans tanımlandığı zaman bu pek işe yaramaz; yaraması için bu referansın ilgili nesneye bağlı olması gerekir; aksi durumda, hata ile karşılaşılır. (*Null PointerException*- bkz. Bölüm 8). Şimdi, bu referansımızı ilgili nesneye nasıl bağlanacağını inceleyelim, (yorum ekle)

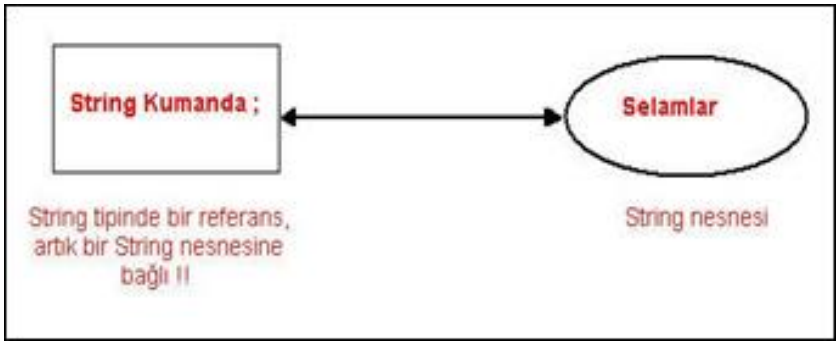
Gösterim-1.2:

```
String kumanda= new String("maket ucak");
```

Gösterim-1.3:

```
String kumanda="maket ucak";
```

Bu gösterimlerin şekil üzerindeki ifadesi aşağıdaki gibi olur:

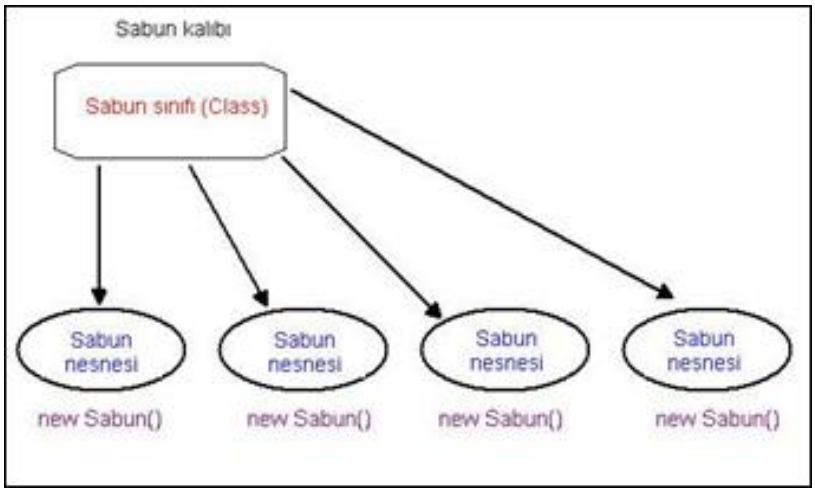


Şekil-1.5. Referans nesne bağlantısı

Verilen gösterimlerde *String* tipindeki referanslara *String* tipindeki nesneler bağlanmıştır. Gösterim-1.2 ile 1.3 arasında herhangi bir fark yoktur. Java'da *String* nesnesinin özel bir yeri vardır. *String* nesneleri çok sık kullanıldıkları için Gösterim-1.2 'deki ifade bir nevi kısaltma gibi düşünülebilir... (yorum ekle)

1.9. Sınıf (*Class*) Nedir? Nesne (*Object*) Nedir?

Sınıf ve nesne kavramı bir benzetme ile açıklanırsa: Sabun fabrikasında yeni bir sabun tasarımı üzerinde çalıştığımızı hayal edelim; ortaya yeni bir kalıp çıkarttık... Artık son aşama olan üretime geçmek istiyoruz. Bu kalıp içerisinde sabun nesnelerinin hangi boyutlarda, hangi renkte olacağı, nasıl kokacağı vs. gibi bilgilerin hepsi bizim tarafımızdan belirlenmiş durumda olacaktır. Üretim aşamasına geçildiğinde hep aynı sabun kalıbını kullanılarak yeni sabun nesneleri üretmemiz mümkün olacaktır. Buradaki önemli nokta, sabun kalıbı tasarımı birkez yapılmış olmasıdır; ancak, bu kalıp ile ***N*** tane sabun nesnesi üretilebilmektedir. Buradan yola çıkılarak sabun kalıbını sınıfa, sabunlarsa nesnelere benzetilebilir. (yorum ekle)



Şekil-1.6. Sınıf ve nesne'nin gösterilmesi

1.10. Depolanan (*Storage*) Veriler Nerede Durmaktadır?

Depo toplam 4 alandan oluşur, bu 4 alan aşağıdaki gibi açıklanabilir:

§ **Yığın (*Stack*):** Bulunduğu nokta bellek içerisindedir; yani RAM üzerinde tutulur. Bu alanda bulunan yığın işaretçisine (*stack pointer*) doğrudan CPU'dan donanım desteği vardır. Yığın işaretçisi aşağıya inince yeni bir bellek alanı oluşturur, yukarı kalkınca ise bellek alanını bırakır (*release*). Java derleyicisi programı oluşturmadan önce yığın üzerinde oluşturulacak olan verilerin boyutlarını ve ömürlerini bilmek zorundadır. Çünkü yığın işaretçisini (*stack pointer*) aşağı ve yukarı hareket ettirecek olan kodu oluşturması gerekmektedir. Yığın üzerinde referansların kendileri bulunur. Maket uçağı örneğini hatırlarsak, bu alanda sadece kumanda cihazları durabilir. (yorum ekle)

§ **Heap:** Genel amaçlı bir bellek havuzudur. Yığın alanının tersine, derleyici burada ne kadarlık bir belleğin pay edileceğini bilmek zorunda değildir. Bu büyük bir rahatlık getirmektedir; çünkü ne zaman bir nesne oluşturmak istersek sadece `new` anahtar kelimesini kullanarak bu alanda bir yer atanır. Bu kadar rahatlığın karşılığında ise

ödenmesi gereken maliyet hızdır. *Heap* alanında yer ayırmak için harcanan zaman, yığın alanında yer ayırmaktan daha fazladır. *Heap* alanında nesnelerin kendisi durur. Maket uçağı örneğini hatırlarsak, bu alanda sadece maket uçaklarının kendileri bulunur, yani bu alanı gökyüzü gibi düşünebiliriz. (yorum ekle)

§ Statik Alan: Bu alan da RAM üzerinde bulunur. Statik alanda yer alan veriler, programın çalışması süresince orada yaşarlar. Tüm nesneler bu statik verileri görebilirler, burayı ortak bir alan gibi düşünebiliriz. Veriyi statik yapmak için `static` kelimesini global değişkenin (*referans*) önüne getirmemiz yeterli olur. Nesnelerin kendileri bu alanda yer almazlar. (yorum ekle)

§ Sabit Disk: Bazı durumlarda uygulamaların içerisinde oluşturduğumuz nesnelerin, uygulama sonlandıktan sonra bile varlıklarını sürdürmelerini isteriz. (yorum ekle)

Akışkan Nesneler (*Streamed Objects*): Bu nesneler genel olarak ağ (*network*) üzerindeki başka bir sisteme gönderilmek üzere *byte* (sekizli) ırmaklarına dönüştürülürler. (yorum ekle)

Kalıcı Nesneler (*Persistent Objects*): Bu nesneler kendi durumlarını saklarlar; saklamaktan kasıt edilen ise özelliklerinin (*attribute*) değerlerinin korunmasıdır. (yorum ekle)

1.11. Temel Tipler

Java programlama dilinde bulunan özel bir grup daha vardır. Bu gruba temel (*primitive*) tipler denir; bunlara uygulama yazılırken çoğu yerde gereksinim duyulur. Bu nedenle bu temel tipleri *heap* alanında `new` anahtar sözcüğüyle oluşturmak pek de avantajlı olmamaktadır. Bunun yerine bu temel tiplerin yığında (*stack*) saklanması çok iyi başarım (*performans*) vermektedir. Yalnız buradaki espri her temel değişkenin bir referans olmamasıdır; yani, temel tipler değerlerini kendi üzerlerinde taşırlar. Tablo-1.2'de Java'nın temel tüpleri listelenmiştir: (yorum ekle)

Tablo-1.2. JAVA programlama dilinin temel tipleri

Temel tip	Boyut	Minimum	Maximum	Sarmalıyıcı sınıf
`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

Bu temel tiplerin birer adet sarmalayıcı (*wrapper*) sınıfı bulunur. Örneğin, temel `int` tipinin sarmalayıcısı *Integer* sınıfıdır; benzer şekilde `double` tipinin sarmalayıcısı *Double* sınıfıdır. Temel tipler ile sarmalayıcıları sınıfları arasındaki farklar ilerleyen bölümlerde ele alınacaktır. (yorum ekle)

Gösterim-1.4:

```
int i = 5; // temel tip
```

Gösterim-1.5:

```
Integer in = new Integer(5);    // sarmalayıcı sınıf
```

1.12. Geçerlilik Alanı (*Scope*)

Her programlama dilinde değişkenlerin geçerlilik alanı kavramı bulunur. Java ile C ve C++ dillerindeki değişkenlerin geçerlilik alanlarının nasıl olduğunu görüp bir karşılaştırma yapalım: (yorum ekle)

Gösterim-1.6:

```
{
    int a = 12;
    /* sadece a mevcut*/
    {
        int b = 96;
        /* a ve b mevcut */
    }

    /* sadece a mevcut */
    /* b geçerlilik alanının dışına çıktı */
}
```

İlk önce, Java programlama dili içerisindeki geçerlilik kavramının nasıl olduğunu inceleyelim. Yukarıdaki gösterimde 2 değişkeninin geçerlilik alanları incelenmektedir. Temel `int` tipinde olan **a** değişkeninin geçerlilik alanı kendisinden daha iç tarafta olan alanlar da bile geçerlidir; ancak, aynı tipte olan **b** değişkeni incelenirse, kendisinden daha dış tarafta olan alanlarda geçerli olmadığı görülür... Şimdi aşağıdaki gösterimi inceleyelim, bu ifade C ve C++ için doğru ama Java programlama dili için yanlış olur.(yorum ekle)

Gösterim-1.7: (yorum ekle)

```
{  // dış alan
     int a = 12;
    {  // iç alan
       int a = 96;  /* java için yanlış, C ve C++ doğru*/  }
       // iç alanın sonu
     }
    //dış alanın sonu
```

1.13. **Nesnelerin Geçerlilik Alanları**

Java programlama dilinde nesnelerin ömürleri, temel tiplere göre daha farklıdır. (yorum ekle)

Gösterim-1.8: (yorum ekle)

```
if (true){
   String s = new String("Selamlar");
}  /* geçerlilik alanının sonu*/
```

Yukarıdaki gösterimde `if` koşuluna kesinlikle girilecektir.; girildiği anda *String* nesnesi *heap* alanında oluşturulacaktır. Bu yeni oluşturulan *String* nesnesi, *String* tipindeki `s` referansı (değişken) ile denetlenmektedir. Peki `if` koşulundan çıkıldığında ne olacaktır? Geçerlilik alanı sona erdiğinden `s` referansı artık kullanılamayacak hale gelecektir; ancak, ya *heap*'deki *String* nesnesi ne olacaktır? Yanıt basittir! Çöp "toplayıcı" devreye girdiği an *heap* alanındaki bu **erişilemez** ve **çöp haline** gelmiş olan*String* nesnesini bellekten silecektir. Bu durum C++ dilinde büyük bir sorundur: Çünkü, C++'da oluşturulan her nesneyi yok etme sorumluluğu yine kodu yazan kişiye aittir; herhangi bir nesneyi yok etmeyi unutursa bellek kaçakları (*memory leak*)başlayacaktır... (yorum ekle)

1.14. **Yeni Sınıf Oluşturma**

Java programlama dilinde kendimize özgü bir sınıf nasıl oluşturabiliriz? Sorusuna yanıt olarak aşağıdaki gösterimi örnek verebiliriz. Aşağıda oluşturulan sınıfın hiç bir fonksiyonu yoktur ama ilerleyen safhalarda bu sınıfımızı geliştireceğiz. (yorum ekle)

Gösterim-1.9: (yorum ekle)

```
public class YeniBirSinif {
  // gerekli tanimlar...
 }
```

1.15. **Alanlar ve Yordamlar**

Bir sınıf (*class*) tanımladığı zaman bu sınıfın iki şey tanımlanabilir: (yorum ekle)

Œ Global **Alanlar** yani global **değişkenler**: temel (*primitive*) bir tip veya bir başka sınıf tipinde olabilirler. (yorum ekle)

Gösterim-1.10: (yorum ekle)

```
public class YeniBirSinif {
   public int i;
   public float f;
   public boolean b;
}
```

Global değişkenlere başlangıç değeri verilmek isteniyorsa,

Gösterim-1.11: (yorum ekle)

```
public class YeniBirSinif {
    public int i = 5;
    public float f = 3.23;
    public boolean b = true;
 }
```

Global değişkenler kullanılmadan önce başlangıç değerlerini almış (*initialize*) olmaları gerekir. Peki, Gösterim-1.11'de biz herhangi bir ilk değer verme işlemi yapmadık ve Java bu konuda bize kızmadı; neden? (yorum ekle)

Tablo-1.3. Java temel tiplerin başlangıç değerleri (yorum ekle)

Temel Tip	Varsayılan (Default) Değer
Boolean	false
Char	'\u0000' (null)
Byte	(byte)0
Short	(short)0
İnt	0
Long	0L
Double	0.0d
Float	0.0f

(Not: Sınıf tipindeki referanslara o tipteki nesne bağlanmamış ise değeri **null**'dır)

Bu sorunun yanıtı yarıda verilen tabloda yatıyor. Eğer bir global değişkene ilk değeri verilmezse, Gösterim-1.11'de yapıldığı gibi, Java bunlara kendi varsayılan (*default*) değerlerini verir. (yorum ekle)
YeniBirSinif sınıfına gelince, bu sınıf içerisinde hala işe yarar bir şeyler yok gibi, sadece 3 adet global değişken tanımlanmıştı... Şimdi bu *YeniBirSinif* sınıfına ait bir nesne oluşturulsun: (yorum ekle)

Gösterim-1.12: (yorum ekle)

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

`ybs` ismini verdiğimiz referansımız, *heap* alanındaki *YeniBirSinif* nesnesine bağlı bulunmaktadır. Eğer biz *heap* alanındaki bu *YeniBirSinif* nesnesiyle temas kurulması istenirse `ybs` referansı kullanılması gerekir. (yorum ekle)
Nesne alanlarına ulaşılması için "**.**" (nokta) kullanılır. (Not: Ulaşmak isteğimiz alan *private* ise o zaman o alana dışarıdan ulaşmanın hiçbir yolu yoktur, **public**, **friendly**, **protected**, **private** ilerideki konularda detaylı bir şekilde anlatılmaktadır) (yorum ekle)

Gösterim-1.13: (yorum ekle)

```
ybs.i;ybs.f;ybs.b;
```

Eğer nesnenin alanlarındaki değerler değiştirilmek isteniyorsa,

Gösterim-1.14: (yorum ekle**)**

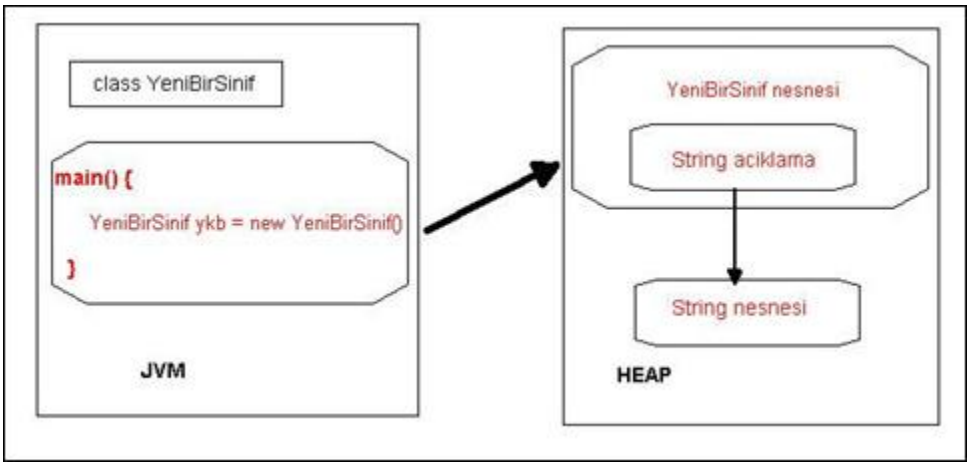
```
ybs.i = 5;ybs.f = 5.3f;ybs.b = false;
```

Sınıflarımıza ait global değişkenlerin tipi temel ise, bu değişkenlere nasıl değer atanacağını ve nasıl değerlerinin alınacağını öğrenmiş olduk... Peki sınıflara ait global değişkenlerin tipleri başka bir sınıf tipinde ise olayların akışı nasıl olacaktır? (yorum ekle)

Örnek: *YeniBirSinif.java* (yorum ekle)

```
class YeniBirSinif {
   public int i;
   public float f;
   public boolean b;
   public String aciklama =
       new String("nesnemizin aciklamasi");
}
```

Örnekte verilen *YeniBirSinif* sınıfının içerisinde temel tipteki global değişkenlerin dışında, başka sınıf tipinde olan `aciklama` değişkeni yer almaktadır. Temel tiplerle sınıf tipindeki değişkenlerin arasındaki fark, aşağıda verilen şekil üzerinden incelenirse,(yorum ekle)



Şekil-1.7. Sınıf Tipindeki değişken

Şekildeki **`main()`** yordamı Java uygulamaları için başlama noktasıdır. *YeniBirSinif* sınıfına ait bir nesne oluştururken görüyoruz ki `aciklama` global değişkenine bağlı olan *String* nesnesi de *heap* bölgesinde yerini alıyor; yani, *heap* bölgesinde 2 adet nesne oluşmuş oluyor. Biri *YeniBirSinif* sınıfına, diğeri ise *String* sınıfına ait nesnelerdir. (yorum ekle)

Yordamlar: Nesnelerin işe yarar hareketler yapmasına olanak veren kısımlar diye bir giriş yapılırsa sanırım yanlış olmaz. Aşağıdaki gösterimde bir yordamın iskeletini incelenmektedir. (yorum ekle)

Gösterim-1.15: (yorum ekle)

```
DönüşTipi yordamınIsmi( /* parametre listesi */ ) {

    /* Yordam gövdesi */
}
```

Yukarıdaki yordam iskeletinde tanımlanmış olan kısımlar birer birer açıklanırsa:

- **dönüşTipi** = Bir yordam ya değer döndürür ya da döndürmez. Bu değer bir temel tip veya bir nesneye bağlı referans olabilir. Hatırlarsanız nesneler *heap* alanında bulunurlardı ve bu nesnelerin yerleri sabittir. Bu yüzden yordam içerisinden döndürüleceği iddia edilen değer, eğer bir sınıf tipinde ise (örneğin *String*) döndürülecek olan, bu sınıf tipindeki nesnenin kendisi değil, bu nesneye bağlı bir referans olacaktır. Eğer bir yordam hiçbir şey

döndürmüyorsa **`void`** sözcüğünü yordamın başına yerleştirilmesi gerekir. (yorum ekle)

§ **yordamınİsmi** = Java'nın kendisine ait olan anahtar sözcükleri (**if, else, import, class, return** vs gibi) ve Türkçe karakter içermeyen herhangi bir isim kullanılabilir; ancak, yordamlar bir eylem içerdikleri için yordam isimlerinin de bir eylemi belirtmesi önerilir. Örneğin, `Sirala()`, `enBuyuguBul()`, `sqlCalistir()` gibi; burada dikkat edilirse, yordam isimlerinin ilk harfleri küçük sonra gelen ek sözcüğün ilk harfi ise büyüktür. Bu ismin anlamını daha kolay görmek içindir. (yorum ekle)

§ **parametre listesi** = Yordam içerisinde işlemler yapabilmek için gerekli olan parametrelerdir. Bu parametreler temel tipte veya sınıf tipinde olabilirler. (yorum ekle)

§ **yordam gövdesi** = Bu kısım kodu yazan kişinin hayal gücüne bağlı olarak değişmektedir. (yorum ekle)

Bu kadar açıklamadan sonra gerçek bir yordam örneği verebilirse,

Gösterim-1.16: (yorum ekle)

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

`uzunlukDondur()` yordamı *String* tipinde parametre alıyor ve *String* nesnesinin uzunluğunu geri döndürüyor. Yordamımızın geri döndürdüğü değer temel `int` tipindedir. Herhangi bir değer geri döndürülebilmesi için `return` anahtar kelimesi kullanılır.(yorum ekle)

Gösterim-1.17: (yorum ekle)

```
String elmaHesapla( int elmasayisi) {
  return new String(" toplam elma sayisi = " +
     elmasayisi*18);
}
```

Gösterim-1.17'de verilen `elmaHesapla()` yordamı tamsayı tipinde parametre alıyor; sonra yeni bir *String* nesnesi oluşturup bu nesnenin bağlı bir referansı geri döndürüyor. Buradaki ilginç olabilecek olan unsur `int` olan bir değişkeni 18 ile çarpılıp sonradan + operatörü ile *String* bir ifadenin sonuna eklenmiş olmasıdır. Diğer dillerde bu işlem için çevirici bir fonksiyona gerek duyulurdu... Örneğin Delphi programlama dilindeki bu işlem için `intToStr()` fonksiyonunu çok kere kullandığımı hatırlarım; ancak, Java dilinde *String* bir ifadeden sonra gelen herhangi bir tipteki değişken otomatik olarak *String* nesnesine dönüştürülür. (yorum ekle)

`" toplam elma sayisi = "` *String* bir ifadedir ve bundan sonra gelen her türlü tip otomatik olarak *String* tipine dönüştürülürler. Eğer Java'nın yardım metinlerinden (http://java.sun.com/javadoc/), *Object* sınıfına ait bilgilere bakılırsa, her nesnenin hali hazırda bir `toString()` yordamının olduğu görülür. Eğer bir nesne otomatik veya

değil *String* nesnesine dönüştürülmek istenirse bu nesnenin `toString()` yordamı çağrılır. Bu konu ilerleyen konularda ayrıntılı olarak ele alınacaktır. (yorum ekle)

Gösterim-1.18: (yorum ekle)

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

`hesapla()` yordamı iki adet parametre almaktadır ve geriye hiçbir şey döndürmeyeceğini `void` anahtar kelimesi belirtmektedir. Bu örnekte dikkat edilmesi gereken ikinci unsur ise yordamların içerisinde tanımlanan yerel değişkenlerine başlangıç değerlerinin kesinlikle programcı tarafından belirtilmesi gerekliliğidir. (yorum ekle)
Sınıflara (*Class*) ait global değişkenlere başlangıcı değerleri verilmediği zaman bu değişkenlere varsayılan değerleri verilir (bkz. Tablo-1.3); ancak, yordam içerisinde tanımlanan yerel değişkenler için aynı durum söz konusu değildir. Bu nedenle`toplamboyut` değişkeninin tanımlanma şekli yanlıştır. (yorum ekle)

Gösterim-1.19: (yorum ekle)

```
void uniteKontrol(int deger) {
   if (deger == 1 ) {
      // eğer değer 1'e eşitse yordamı terk et
      return;
   }else {
    // gerekli işlemler
   }
}
```

Bu yordam örneğindeki anafikir `void` ile `return` anahtar kelimelerinin aynı yordam içinde kullanılmasını göstermektir. `if-else` kontrol yapısı henüz görülmedi; ancak, bu örnek için kullanılmaları gerekliydi... Buradaki `return` parametresi yordamın acilen terk edilmesi gerektiğini belirtir. Yani `return` anahtar kelimesi tek başına kullanıldığında ilgili yordamın içerisinden çıkılır. (yorum ekle)

1.16. İlk Java Programımız

Örnek: *Merhaba.java* (yorum ekle)

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

İlk örneğimizi adım adım açıklanırsa, (Java büyük ve küçük harfe karşı duyarlıdır; yani, `public` yerine `PUBLIC` yazılırsa hata ile karşılaşılır):

public class merhaba: Bu kısımda yeni bir sınıf oluşturuluyor. (yorum ekle)
public static void main(String args[]): Java'da bir sınıfın tek başına çalışması isteniyorsa (*stand alone*) bu yordam yazılmak zorundadır. Bu yordam sınıflar için başlangıç noktası gibi varsayılabilir. Burada iki bilinmedik konuyu ele almak gereklidir: birisi statik yordamlar, diğeriyse dizilerdir. (yorum ekle)
statik yordamlar: Statik yordamlar nesneye bağımlı olmayan yordamlardır. Bunların kullanılması için sınıfa ait nesnenin oluşturulmuş olması gerekmez. (yorum ekle)

Örnek: *TestNormal.java* (yorum ekle)

```
public class TestNormal {
   public void uyariYap() {
     System.out.println("Dikkat Dikkat");
   }

   public static void main(String args[]) {
     TestNormal tn = new TestNormal();    // dikkat
     tn.uyariYap();
   }
}
```

TestNormal.java uygulamamızda `uyariYap()` yordamı statik değildir; bu nedenle bu yordamın çağrılabilmesi için *TestNormal* sınıfına ait bir nesne oluşturulması gerekir. Şimdi diğer örneğimize geçelim... (yorum ekle)

Örnek: *TestStatik.java* (yorum ekle)

```
public class TestStatik {
   public static void uyariYap() {
```

```
      System.out.println("Dikkat Dikkat statik
+"metod");

}
    public static void main(String args[]) {
      TestStatik.uyariYap();
    }
}
```

Bu örnekteki tek fark **`uyariYap()`** yordamının statik olarak değiştirilmesi değildir; çağrılma şekli de değiştirilmiştir. `uyariYap()` yordamı artık *TestStatik* nesnesine bağlı bir yordam değildir, yani `uyariYap()` yordamını çağırabilmemiz için *TestStatik*sınıfına ait bir nesne oluşturulması gerekmez. `main()` yordamında da işler aynıdır, fakat `main()` yordamının Java içerisinde çok farklı bir yeri vardır. `main()` yordamı tek başına çalışabilir uygulamalar için bir başlangıç noktasıdır. (yorum ekle)

Diziler (*Arrays*): `main()` yordamı parametre alarak *String* tipinde dizi alır. Bu *String* dizisinin 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 sıfırdan başlar; ilerleyen bölümlerde ayrıntılı olarak ele alınmaktadır. (yorum ekle)

`System.out.println ("Merhaba Barış!");` Bu satır, bilgilerin ekrana yazılmasınaı sağlar. Java'nın yardım dokümanlarına bakılması önerilir: http://java.sun.com /javadoc *System* sınıfının `static` bir alanı (global değişkeni) olan `out` sayesinde*PrintStream* nesnesine ait bir referans elde edebiliyoruz. *PrintStream* nesnesinin `println()` yordamı ile bilgiler kolayca ekrana yazdırılabilir. (yorum ekle)

1.17. **JAVA Kurulumu, Derleme ve Çalıştırma**

JAVA'nın kurulumu için hem Unix hem de Windows işletim sistemlerinde nasıl gerçekleştirildiğine bakalım; JAVA'nın son versiyonu "http://java.sun.com" adresinden alınabilir. (yorum ekle)

1.17.1. **UNIX/Linux İşletim Sisteminde Kurulumu**

"http://java.sun.com" adresinden sisteminize uygun olan Java versiyonunu seçmeniz gerekmektedir. *Solaris* ve Linux için değişik Java versiyonları bulunmaktadır. Ancak kurulumları aynıdır. İşte yapılması gerekenler: (yorum ekle)

· İndirmiş olduğunuz Java kurulum dosyasını size uygun bir yere açın (örneğin */usr/java*), `gunzip` ve `tar` komutlarının örneği aşağıdaki gibidir; öncelikle sıkıştırılmış dosyayı açıyoruz;

bash# `gunzip j2sdk.tar.gz`

Daha sonra arşivlenmiş dosyanın içeriğini */usr/java* dizinine çıkartıyoruz

bash# `tar xvf j2sdk.tar /usr/java` (yorum ekle)

· *etc/profile* dosyasının içersine bazı eklemeler yapılması gerekmektedir.

Unix'te her kullanıcının kendisine ait profile dosyası bulunur, bu yapılan işlemleri o dosyaların içerisinde de yapabilirsiniz, buradaki fark */etc/* dizinin altındaki *profile* dosyasında yapılan bir değişikliğin tüm kullanıcıları doğrudan etkilemesidir. (yorum ekle)

Öncelikle `PATH` değişkenini değiştirmemiz gerekli ve sonradan `CLASSPATH` değişkenini tanımlamamız gereklidir. *j2sdk.gz* dosyasının içerisindeki dosyaları "/usr/java" dizinine açtığınızı varsayıyorum. (yorum ekle)

`PATH="/usr/bin:/usr/local/bin:/usr/java/bin:."`

Sadece koyu olarak yazılan yeri (*/usr/java/bin*) yeni ekledim; */usr/java/bin* içersinde çalıştırılabilir dosyalar bulunmaktadır. Şimdi sıra `CLASSPATH` değerlerini vermeye geldi.

`CLASSPATH="/ usr/java/lib/tools.jar:."`

tools.jar dosyasının içerisinde yararlı sınıflar bulunur. JAR dosyaları ilerleyen bölümlerde ele alınacaktır. Dikkat edilirse `CLASSPATH` tanımlarken en sona bir nokta koyuldu, bu nokta yaşamsal bir önem taşır; yararı, bulunulan dizindeki .**class** dosyalarının görülebilmesini sağlamaktadır, böylece Java komutunu çalıştırırken saçma hatalar almayız. (yorum ekle)

· Yaptığımız bu değişikliklerin etkili olabilmesi için aşağıdaki komutunun yazılıp yürütülmesi yeterli.

.bash#. `/etc/profile` à en baştaki noktaya dikkat ediniz.

Yaptığımız işlemlerin etkili olup olmadığını öğrenmek için, sırasıyla aşağıdaki komutları deneyebilirsiniz.

bash# `java -version` à

Sisteminizde yüklü olan Java versiyonunu öğrenmenize yarar

bash# `echo PATH` à

PATH değişkeninin hangi değerler taşıdığını söyler.

bash# `echo CLASSPATH` à

`CLASSPATH` değişkeninin hangi değerler taşıdığını söyler. (yorum ekle)

1.17.**2.** JAVA'nın Windows İşletim Sisteminde Kurulumu

Java'nın Windows için hazır bir kurulum dosyası bulunur; bu dosya üzerine çift tıklanarak yapılması gereken işlerin büyük bir kısmı gerçekleştirilmeye başlar; ancak, hala yapılması gereken ufak tefek işler vardır: (yorum ekle)

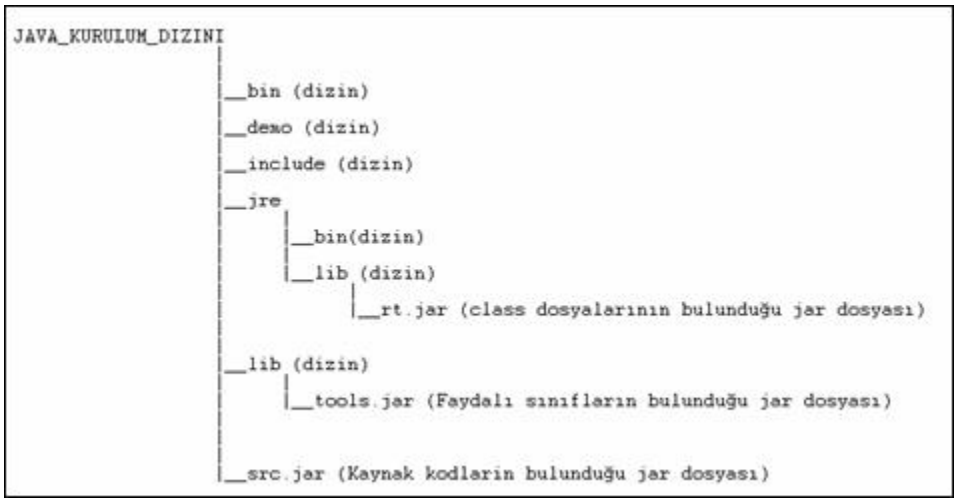
Windows 95-98 için *autoexec.bat* dosyasında aşağıdaki değişlikleri yapmanız yeterli olacaktır. (Not: Aşağıdaki `PATH` ve `CLASSPATH` değişkenleri benim bilgisayarımdan alıntıdır!)

```
SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\JAVA\BIN;
C:\ultraedit;.
SET CLASSPATH=;C:\JAVA\lib\tools.jar;.
```

(**Not**: Tekrar oldu ama olsun çok önemli bir kısım: Dikkat edilirse `CLASSPATH` tanımlanırken en sona bir nokta koyuldu, bu nokta hayati bir önem taşır. Bu noktanın anlamı, bulunulan dizindeki .**class** dosyalarının görülebilmesini sağlamaktadır, böylece javakomutunu çalıştırırken saçma hatalar almayız.) (yorum ekle)

Windows 2000 için `PATH` ve `CLASSPATH` tanımlarını *Environment Variables* kısmına ulaşarak yapabilirsiniz. Bu bölüme ulaşmak için **Control Panel** --> **System** --> **Advanced** --> **Environment Variables** yolunu izlemeniz yeterli olacaktır. (yorum ekle) *System Variables* veya *User Variables* bölümünde, `PATH` değişkenini değiştirmeniz ve "new" diyerek yeni bir `CLASSPATH` değişkeni tanımlamanız gerekmektedir. (`CLASSPATH` değişkeni hiç olmadığı varsayılıyor). `PATH` ve `CLASSPATH` değişkenlerini yukarıdaki gibi Java ile ilgili parametreleri girerek kurulumu tamamlayabilirsiniz. (yorum ekle)

Kurulum ilgili aklınızda daha çokça olabilir veya yolunda gitmeyen işler, bunlar için Java ile beraber gelen kurulum yardım dosyalarını okumanızı öneririm. Şekil-1.8'de Java dosyalarının yapısı görülmektedir. (yorum ekle)



Şekil-1.8. Java dosya düzenlenmesi

1.17.3. **JAVA Kodlarını Derleme ve Çalıştırma**

Elimizdeki *Merhaba.java* dosyasını nasıl derleyeceğiz? Cevabı aşağıdaki gibidir:

```
$ javac Merhaba.java
```

Artık elimizde *Merhaba.class* dosyasının oluşmuş olması gerekir. Şimdi sıra geldi bu dosyayı çalıştırmaya,

```
$ java Merhaba
```

Ekrana çıkan yazın:

```
Merhaba Barış!
```

Ne yapıldığını tekrarlanırsa, elimizde bulunan *Merhaba.java* kaynak dosyasını *javac* uygulamasını kullanarak derledik ve *Merhaba.class* dosyamız oluşmuş oldu. Daha sonradan Java komutunu kullanarak uygulamamızı çalıştırdık. (yorum ekle)

1.18. **Nedir bu args[], Ne İşe Yarar ?**

Tek başına çalışabilir Java uygulamasına, komut satırından (konsoldan) nasıl çalıştığını anladıktan sonra komut satırından Java uygulamamıza parametre göndermeyi öğrenebiliriz. Diğer bir uygulama örneği, (yorum ekle)

Örnek: *ParametreUygulamasi.java* (yorum ekle)

```
public class ParametreUygulamasi {
  public static void main(String[] args) {
     System.out.println("Merhaba Girdiginiz" +
" Parametre = " + args[0]);
  }
}
```

Anımsanırsa Java'da dizilerin indis sıfırdan başlarlar. Şimdi *ParametreUygula.java* kaynak dosyası incelenirse,

```
$ javac ParametreUygulamasi.java
```

Elimizde *ParametreUygulamasi.class* dosyası oluştu; şimdi uygulamamızı çalıştırabiliriz, yalnız farklı bir şekilde,

```
$ java ParametreUygulamasi test
```

Uygulamamızı çalıştırma tarzımız değişmedi, burada ki tek fark, en sona yazdığımız **test** kelimesini uygulamaya parametre olarak gönderilmesidir. İşte programın çıktısı:

```
Merhaba Girdiginiz Parametre = test
```

Tek başına çalışabilir Java uygulamasına konsoldan birden fazla parametreler de gönderebiliriz. (yorum ekle)

Örnek: *ParametreUygulamasi2.java* (yorum ekle)

```
   public class ParametreUygulamasi2 {
     public static void main(String[] args) {
      System.out.println("Merhaba Girdiginiz ilk Parametre =
" + args[0]);
      System.out.println("Merhaba Girdiginiz ikinci"+
" Parametre = " + args[1]);
     }
 }
```

Uygulamamızı öncelikle derleyelim:

```
$ javac ParametreUygulamasi2.java
```

ve şimdi de çalıştıralım:

```
$ java ParametreUygulamasi2 Test1 Test2
```

Ekranda görülen:

```
Merhaba Girdiginiz ilk Parametre = Test1
Merhaba Girdiginiz ikinci Parametre = Test2
```

Peki bu uygulamaya dışarıdan hiç parametre girmeseydik ne olurdu ? Deneyelim:

```
$ java ParametreUygulamasi
```

Sansürsüz şekilde karşılaştığım görüntü:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsExceptionat
ParametreUygulamasi2.main(ParametreUygulamasi2.java:5)
```

Hımm, sorun nedir? Sorun şu, Java uygulamamız bizden kesin olarak iki adet parametre girmemizi bekliyor. Beklediğini aşağıdaki kısımlarda görebiliyoruz: (yorum ekle)

Gösterim-1.20:

```
System.out.println("Merhaba Girdiginiz ilk
Parametre = " +
args[0]);System.out.println("Merhaba Girdiginiz
ikinci Parametre = " + args[1]);
```

`args[0]` ve `args[1]`, ancak biz dışarıdan iki adet parametre girersek bu dizinin ilk iki elemanı veri ile dolacaktır. Eğer dışarıdan parametre girmezsek dizinin birinci ve ikinci elamanları oluşmayacaktır. Oluşmamış bir dizi elemanına ulaşmaya çalıştığımızda ise yukarıdaki hata ile karşılaşılır. Java'nın çalışma anında dizi eleman erişimini kontrol etme özelliği

uygulamanın sağlamlığını arttırmaktadır; fakat, diğer tarafından bu özelliğe karşı ödenmesi gereken fatura ise hızdır. (yorum ekle)

1.19. **Javadoc = Yorum ile Dokümantasyon Oluşturmak**

Uygulamalar yazılırken büyük bir oranla dokümantasyon işi ikinci plana itilir veya unutulur veya en kötüsü hiç yapılmaz. Dokümantasyon kodu yazan kişiler için çok ağır iştir (yazarımızda bu fikri katılıyor...). Java, dokümantasyon hazırlama işini daha kolay ve sevimli bir hale getirmiştir. Bu özelliğe "JavaDoc" denir. *JavaDoc* kodun içersine yazılmış olan yorum satırlarını alarak bunları HTML biçimine dönüştürmektedir. Fakat yorum satırını yazarken bazı kurallara uymamız gerekmektedir. (yorum ekle)

1.19.1. **Sözdizimi Kuralları (*Syntax*)**

Bizim yazdığımız yorum satırlarının *Javadoc* tarafından dikkate alınması için:

/** ile başlaması ve */ ile bitmesi gerekmektedir. *Javadoc* mekanizmasını kullanmanın iki etkili yolu vardır. Bunlardan birincisi gömülü html (*embedded html*), ikincisi ise *doc tags*. (yorum ekle)

Doc tag "@" ile başlarlar. Sınıfa, global değişkenlere ve yordamlara ait üç adet yorum dokümanı tipi vardır. Verilen örnek şablon bunu açıklamaktadır. (yorum ekle)

Gösterim-1.21

```
/** A sınıf ait yorum */
public class DocTest {
   /** i global değişkenin yorumu */
   public int i;
   /** isYap() yordamunun yorumu */
   public void isYap() {}
}
```

Unutulmaması gereken başka bir nokta ise *Javadoc* varsayılan (*default*) `public` ve `protected` olan üyelerin dokümanını üretir. `private` ve `friendly` üyelerin dokümantasyonu yapmaz. Bunların sebeplerine ilerleyen konularda değineceğiz. Eğer*private* üyelere ait bilgilerinde yazılmasına istiyorsak o zaman komut satırından,

```
$ javadoc a -private .....
```

ekini vererek bunu başarabiliriz. (yorum ekle)

1.19.2. Gömülü HTML (*Embedded Html*)

JavaDoc mekanizmasında kendinize ait HTML komutları kullanılması mümkündür:

Gösterim-1.22:

```
/*** <pre>*
 System.out.println("Selamlar");*</pre>*/
```

Gösterim-1.23:

```
/*** Çok güzel <em>hatta</em> liste bile
yerleştirebilirsiniz:*<ol>* <li> madde bir* <li>
madde iki* <li> Madde üç* </ol>*/
```

<h1> <hr> gibi etiketlerini (*tag*) kullanmayın çünkü *Javadoc* bunları sizin yerinize zaten yerleştirmektedir. Gömülü HTML özelliği sınıf, global değişkenler ve yordamlar tarafından desteklenir. (yorum ekle)

1.19.3 Ortak Kullanılan Yorum Ekleri

· **@see:** Başka bir sınıfın, global değişkenin veya yordamın dokümantasyonunu göstermek için kullanabilirsiniz.
(sınıf, global değişken, ve yordam) dokümantasyonlarında, **@see** etiketini (*tag*) kullanabilirsiniz. Bu etiketin anlamı, " başka bir dokümantasyona gönderme yapıyorum" demektir. (yorum ekle)

Gösterim-1.24:

```
@see classismi@see classismi#yordam-ismi
```

JavaDoc, gönderme yaptığınız başka dokümantasyonların var olup olmadığını kontrol etmez. Bunun sorumluluğu size aittir. (yorum ekle)

1.19.3.1. Sınıflara Ait *JavaDoc* Etiketleri

Sınıflara ait etiketler arayüzler içinde kullanılabilir.

· **@version:** Uyarlama numaraları dokümantasyonlamada yaşamsal bir rol oynar. Uyarlama etiketinin görünmesini istiyorsanız:

```
$ javadoc –version
```

Parametresi ile çalıştırmamız gerekmektedir. Versiyon etiketini kodumuza yerleştirmek için; (yorum ekle)

Gösterim-1.25:

```
@version versiyon-bilgisi
```

· **@author:** Sınıfı yazan yazar hakkında bilgi verir. *author* etiketi *Javadoc* tarafından dikkate alınmasını istiyorsanız:

```
$ javadoc –author
```

Parametresini koymanız gerekir. Bir sınıfa ait birden fazla *author* etiketi yerleştirebilirsiniz. *author* etiketini kodumuza yerleştirmek için: (yorum ekle)

Gösterim-1.26:

```
@author author-bilgisi
```

· **@since:** Bir sınıfın belli bir tarihten veya belli bir uyarlamadan itibaren var olduğunu belirtmek için kullanılır. *since* etiketini kodumuza yerleştirmek için: (yorum ekle)

Gösterim-1.27:

```
@since 05.03.2001
```

1.19.3.2. Global Değişkenlere Ait *JavaDoc* Etiketleri

Global değişkenlere ait dokümantasyonlarda sadece gömülü html veya **@see** etiketi kullanılabilir. (yorum ekle)

1.19.3.3. Yordamlara Ait *JavaDoc* Etiketleri

Yordam dokümantasyonunda gömülü html veya @see etiketini kullanabilirsiniz. Ayrıca aşağıdaki etiketleri kullanabilirsiniz. (yorum ekle)

· **@param:** Yordamın aldığı parametrenin açıklamasını belirtir. *param* etiketini kodumuza yerleştirmek için: (yorum ekle)

Gösterim-1.28:

```
@param parametre-ismi açıklaması
```

· **@return:** Yordamın döndürdüğü değerin açıklamasını belirtir. *return* etiketini kodumuza yerleştirmek için: (yorum ekle)

Gösterim-1.29:

```
@return aciklama
```

· **@throws:** İstisnalar (*exception*) konusunu ilerleyen bölümlerde inceleyeceğiz. İstisnalar kısa olarak açıklarsak, yordamlarda hata oluşursa oluşturulan özel nesnelerdir. *throws* etiketini kodumuza yerleştirmek için: (yorum ekle)

Gösterim-1.30:

```
@throws class-ismi açıklaması
```

· **@deprecated:** Önceden yazılmış fakat artık yerine yenisi yazılmış bir yordam *deprecated* olur (tedavülden kaldırılır). *deprecated* yordamı kullandığınız zaman derleyici tarafından uyarı alırsanız. *deprecated* etiketini kodumuza yerleştirmek için: (yorum ekle)

1.19.4. Dokümantasyon Örneği

Örnek: *SelamDoc.java* (yorum ekle)

```
 /** İlk Java Kodumuzun Dokümantasyonu * Ekrana Selamlar diyen bir
uygulama  * @author Altug B. Altintas (altug.altıntaş@koubm.org) *
@version 1.0  * @since 09.01.2002*/

public class SelamDoc {

/**sayıyı artırmak için değişkenler için bir örnek*/
 public int sayac = 0;

/** sınıflarda & uygulamalarda giriş noktası olan yordam * @param args
dişarıdan girilen parametreler dizisi * @return dönen değer yok  *
@exception Hic istisna fırlatılmıyor */
 public static void main(String[] args) {
  System.out.println("Selamlar !");
 }
}
```

Oluşturulan bu kaynak koduna ait doküman çıkartmak için komut satırına aşağıdaki yazılı komutun yürütülmesi yeterlidir:

```
$ javadoc -d javadoc SelamDoc.java
```

Bu komut sayesinde HTML dokümanları bulunduğumuz dizinin altındaki *javadoc* dizininin içinde oluşturulmuş oldu. *Javadoc*'un komutlarıyla daha fazla bilgiyi,

```
$ javadoc –help
```

yazarak ulaşabilirsiniz. (yorum ekle)

1.20. **Sınıf İsimleri ve Yordam İsimleri**

Sınıf isimleri uzun olursa alt çizgi ile ayrılmamalıdır. Sınıf isimleri fiil cümlesi içermemelidir. (yorum ekle)

Gösterim-1.31:

```
public class EnKısaYolBulmaAlgoritması { }
```

Yordamlar fiil cümlesi içermelidirler. İlk kelime küçük harf ile başlamalıdır. (yorum ekle)

Gösterim-1.32:

```
public void yolBul() { }
```

Bu bölümde Java'yı tanıyıp, Java ile nelerin yapılabileceğini gördük. Nesne ile sınıflar arasındaki farktan bahsettik. Nesnelerin ve onlara ait referansların nerelerde durduğu gördük. Java'nın en önemli özelliklerinde biri çöp toplayıcısıdır bu sayede kodu yazan kişiler daha az risk üstlenir. Bu rahatlığında bir bedeli vardır, bu bedelin ne olduğunu ilerleyen bölümlerde göreceğiz. (yorum ekle)

1.21. **Fiziksel Dosya İsimleri**

Önemli olan bir başka husus ise, fiziksel bir dosya içerisinde iki veya daha fazla sayıda sınıf yazabiliyor olmamızdır. Örneğin, yukarıdaki uygulamamızın yer aldığı fiziksel dosyanın ismi neden *SelamDoc.java*'dır? Bunun sebebi, Java programlama dilinde fiziksel dosyaların ismi her zaman *public* sınıf isimleri ile birebir aynı olma gerekliliğidir. Eğer fiziksel dosyanın içerisinde herhangi bir *public* erişim belirliyicisine sahip bir sınıf yoksa, fiziksel dosyanın ismi herhangi bir şey olabilir. Erişim belirliyiciler 4. bölümde incelenmektedir. (yorum ekle)