

# Bölüm -1

## Java Nedir?

Java platform bağımsız, Nesneye Dayalı Programlama'yı (Object Oriented Programming) tamamiyle destekleyen bir yazılım geliştirme dilidir. Java ile geliştirilen bir yazılım uygulaması Java Virtual Machine (Java Sanal Makinesi) yüklü olan herhangi bir bilgisayarda veya cihazda sorunsuz çalışabilir. Java aynı zamanda geliştirme platformunun kendisidir. Yazılım geliştirebilmeye imkan tanıyan ve geliştirilen bu yazılımın çalıştırılmasını sağlayan altyapıya verilen isimdir. Java bir programlama dili olmasının yanı sıra sağladığı bu imkanlarla birlikte platforma da ismini vermiştir. JDK, JRE bahsettiğimiz platformun kendisidir.

Java hızlı, güvenli ve güvenilir özelliklere sahiptir. Masaüstü uygulamalarından, web tabanlı uygulamalara, mobil uygulama geliştirmeden gömülü sistem uygulamalarına kadar çok geniş bir çerçevede kullanılmaktadır. Özel sektörde kurumsal firmalarda, akademik alanlarda bilimsel çalışmalarda da sıkça tercih edilmektedir. Bir kez yaz ve her yerde çalıştır Java'nın en önemli sloganıdır.

### Java'nın Özellikleri

- **Java platform bağımsız bir dildir.** Bu özelliğine yukarıda kısaca değinmiştik. Java'yı platform bağımsız kılan özelliği yazılan kaynak kodlar derlendikten sonra ara bir dil olan byte code'a çevrilmesidir. Ara dile çevrilen bu kod parçaları Java Virtual Machine vasıtasıyla yorumlanır ve çalıştırılır.
- **Java Nesneye Dayalı programlamaya tamamiyle uygun bir dildir.** Java'da her şey nesnelerden ibarettir.
- **Java öğrenmesi basit bir dildir.** Gramer (Syntax) olarak C/C++ tabanlı bir yaklaşıma sahiptir. Java'da pointers(işaretçiler), operator overloading(operator aşırı yükleme), hafıza yönetimi gibi karmaşık olabilecek ve programcının sıkça hata yapabileceği yöntemler, mekanizmalar programcının sorumluluğundan alınıp dilin bünyesine yerleştirilmiştir. Böylece, programcının yapabileceği hatalara karşın bir soyutlama sağlanmıştır. Garbage Collection (Çöp Toplayıcısı) mekanizması ile kullanılmayan nesnelerin hafızadan temizlenmesi işi Java tarafından üstlenilmiştir.
- **Java kararlı bir programlama dilidir.** Java ile yazılan uygulamalar sorunsuz derlendiyse sonsuza dek her platformda çalışabilirler. Güçlü bir hafıza yönetim mekanizması vardır. Hata yakalama sistemi ile kararlılığı artırır.
- **Java, Multi-Thread (Çok Kanallı) programlamayı varsayılan olarak destekler.** Çok kanallı programlama yapabilmek için dilin bünyesinde gerekli altyapı ve imkanlar mevcuttur. Çok kanallı programlama ile geliştirmiş olduğunuz yazılım aynı anda network üzerinden bir dosya indirme işlemi yaparken, bir yanda da veritabanına çeşitli kayıtların yazılabilmesini sağlayabilir. Bu sizlerin bilgisayarınızda müzik dinlerken aynı anda bir Microsoft Word programını kullanabilmeniz gibi bir durumdur. Özetle eş zamanlı işlemleri programlayabilmenize imkan tanır.
- **Dağıtık sistemler ve Web programlama yapabilmenize imkan tanır.** Java ile network (ağlar) üzerinde çalışabilecek dağıtık sistemler programlayabiliriz. TCP/IP,

HTTP ve Socket gibi alt yapılar ile bunları gerçekleştirebiliriz. Remote Method Invocation (RMI) ve REST tabanlı servis özellikleri sayesinde dağıtık bir şekilde haberleşme imkanı sağlamaktadır. Servlet, Java Server Page gibi alt yapılar ile Web programlama da yapılabilmektedir.

- **Gömülü sistemlerde programlama imkanı sağlar.** JavaME platformu ile gömülü sistemlerde Java ile geliştirmeler yapabilirsiniz.

## Java Tarihçesi

1991 yılında Sun Microsystems’de (Şu an bu firma Oracle tarafından satın alınmıştır.) çalışan James Gosling’e ve arkadaşlarına birçok cihazda ve bilgisayarda platform bağımsız şekilde çalışabilecek bir dil ortaya çıkarmaları için görev verildi. Onlar da C++ dilini inceledikten sonra, yeni bir dil ortaya çıkarıp bu hedefi gerçekleştirmeye karar verdiler. Ofislerinin önündeki ağaçtan esinlenerek bu dile Meşe anlamına gelen “Oak” adını verdiler. Ardından, 1 yıl kadar sonra geliştirilen tüm teknolojiler Java 1.0 adı altında toplandı. Bu teknoloji Star7 isimli bir el bilgisayarı ve Time Warner şirketi için ise bir televizyon decoder cihazı geliştirme projelerinde kullanıldı. (Java logosundaki Duke karakteri bu sistemlerde yardımcı sihirbazlığı yapan modülün simgesiydi ☺) Fakat, projelerde istenilen başarı sağlanamadı. Ardından, proje sonucu geliştirilen Java dili kullanılmama durumuna gelme tehlikesiyle karşı karşıya kaldı.

1990’lı yılların başında yaygınlaşmaya başlayan bir teknoloji olan World Wide Web üzerine yoğunlaşmaya başladılar ve dil üzerindeki çalışmalarını bu teknoloji üzerine kurgulamaya başladılar. 23 Mayıs 1995 yılında da Java resmi olarak duyuruldu. Artık, internet tarayıcılarında Java tabanlı uygulamaların çalışması devri başlamıştı. Java’nın 2. sürümü ile birlikte sadece tarayıcılarda çalışan uygulamalar yazmaya imkan veren bir yapıdan, temel nesneye dayalı kurumsal bir programlama diline doğru dönüşmeye başladı. Dilin daha kolay kullanımını sağlayan özellikler ise Java 5 ile geldi. Java 6 ve Java 7 ile dilin olgunlaşması ve kurumsal projelerde sıkça kullanılır hale gelmesi sağlandı. Java 8 ile dile Stream API, Lambda Expressions (Lambda Tanımlamaları) gibi çok önemli yenilikler getirildi.

Java Versiyon Tarihçesi (Detay: [https://en.wikipedia.org/wiki/Java\\_version\\_history](https://en.wikipedia.org/wiki/Java_version_history))

JDK Alpha ve Beta (1995)  
JDK 1.0 (23 Ocak 1996)  
JDK 1.1 (19 Şubat 1997)  
J2SE 1.2 (8 Aralık 1998)  
J2SE 1.3 (8 Mayıs 2000)  
J2SE 1.4 (6 Şubat 2002)  
J2SE 5.0 (30 Eylül 2004)  
Java SE 6 (11 Aralık 2006)  
Java SE 7 (28 Temmuz 2011)  
Java SE 8 (18 Mart 2014)  
Java SE 9 (21 Eylül 2017)  
Java SE 10 (20 Mart 2018)  
Java SE 11 (25 Eylül 2018)  
Java SE 12 (19 Mart 2019)  
Java SE 13 (17 Eylül 2019)

# Java ile Uygulama Geliştirme Yapabileceğiniz Alanlar

Yeryüzünde Java dili geliştirilen yazılımların sayısı gün geçtikçe artmaktadır. Yaklaşık 3 milyar cihazda Java kullanıldığı tahmin edilmektedir. Aynı zamanda Java dili ile birbirinden farklı platformda, farklı amaçlar için çalışan uygulamalar yazabilirsiniz.

- Web uygulamaları geliştirebilirsiniz.
- Masaüstü uygulamaları geliştirebilirsiniz. (Eclipse IDE gibi)
- Kurumsal uygulamalar geliştirebilirsiniz. ([www.sahibinden.com](http://www.sahibinden.com) gibi)
- Mobil uygulamalar geliştirebilirsiniz. (Android ve Java2ME tabanlı uygulamalar gibi)
- Gömülü sistem uygulamaları geliştirebilirsiniz. (Java2ME Embedded altyapısı ile)
- Robotik projelerde kullanabilirsiniz.
- Oyun programlamada kullanabilirsiniz. (Android ile oyun geliştirme gibi)

## Java Uygulama Geliştirme Altyapıları

- 1- **Java SE (Java Standard Edition):** Java programlama diliyle birlikte genel amaçlı, temel düzeyde programlama yapabilmeyi sağlayan platformdur. Java ile ilgili tüm temel kütüphaneleri ve dil özelliklerini içerir. Nesneye dayalı programlama yapabilmek için gerekli olan özellikleri sağlar.
- 2- **Java EE (Java Enterprise Edition):** Java SE'yi kapsamaktadır. Ayrıca, Java ile daha çok web ve kurumsal düzeyde programlama yapabilmeyi sağlar. Servlet, Java Server Page, Web Service'leri, Enterprise Java Bean (EJB) (orta katman yazılım geliştirme altyapısı), JPA (Java Persistence API) (Veritabanı ile etkileşim katmanı)
- 3- **Java ME (Java Micro Edition):** Java ile gömülü sistem uygulamaları geliştirmeyi sağlar. Özellikle mobil cihazlarda geliştirme yapmaya imkan verir.
- 4- **JavaFX:** Java masaüstü uygulamaları geliştirmeyi sağlayan yeni nesil altyapıdır. Öncesinde masaüstü programları geliştirmek için Swing altyapısı tercih edilirdi. Swing'in yerini alması için geliştirilmiştir.









## Java Geliştirme Ortamının Kurulumu

Java ile yazılmış olan uygulamaları çalıştırabilmek için bilgisayarınıza **Java Runtime Environment (JRE)** kurmanız gerekmektedir. JRE uygulamaları çalıştırmak için yeterlidir. Fakat, Java ile yazılım geliştirmeyi amaçlıyorsanız, **Java Development Kit'i (JDK)** kurmanız gerekmektedir. JDK bünyesinde JRE'yi de içermektedir. Ayrıca Java ile geliştirme yapabilmeniz için gerekli kütüphaneleri ve altyapıyı sağlamaktadır. Bu konuya ileride daha derinlemesine bir şekilde ele alacağız. Eğitim boyunca JDK 8 versiyonunu kullanacağız. Şimdi Windows, Mac OS ve Linux ortamları için JDK kurulumundan bahsedelim.

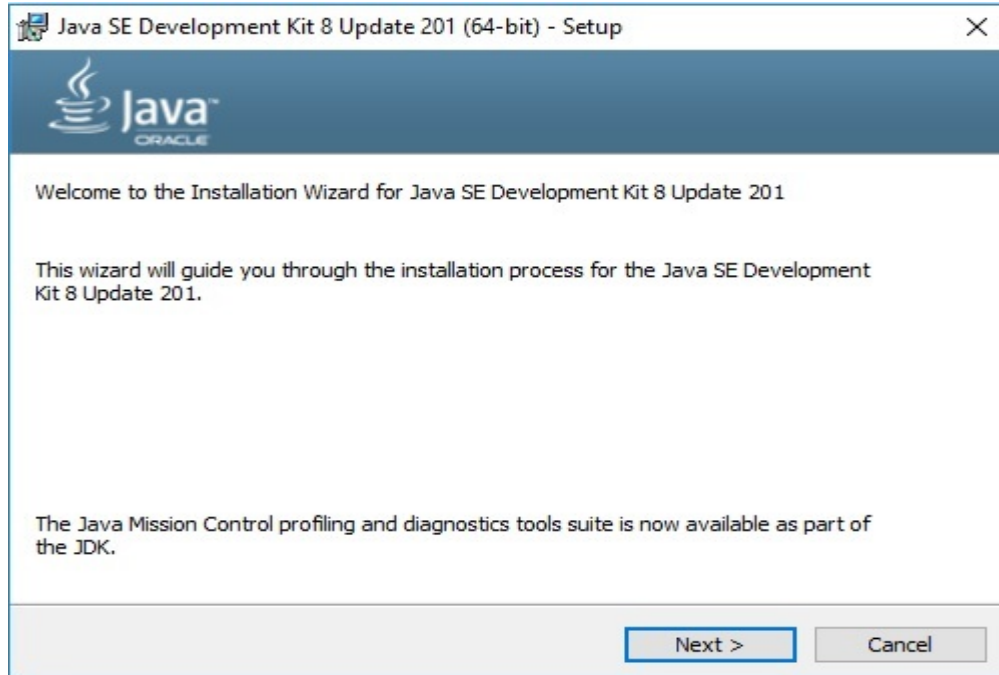
# 1. Windows İşletim Sisteminde JDK 8 Kurulumu

Aşağıdaki linkten Oracle sitesi üzerinden JDK indirme işlemini başlatabilirsiniz.

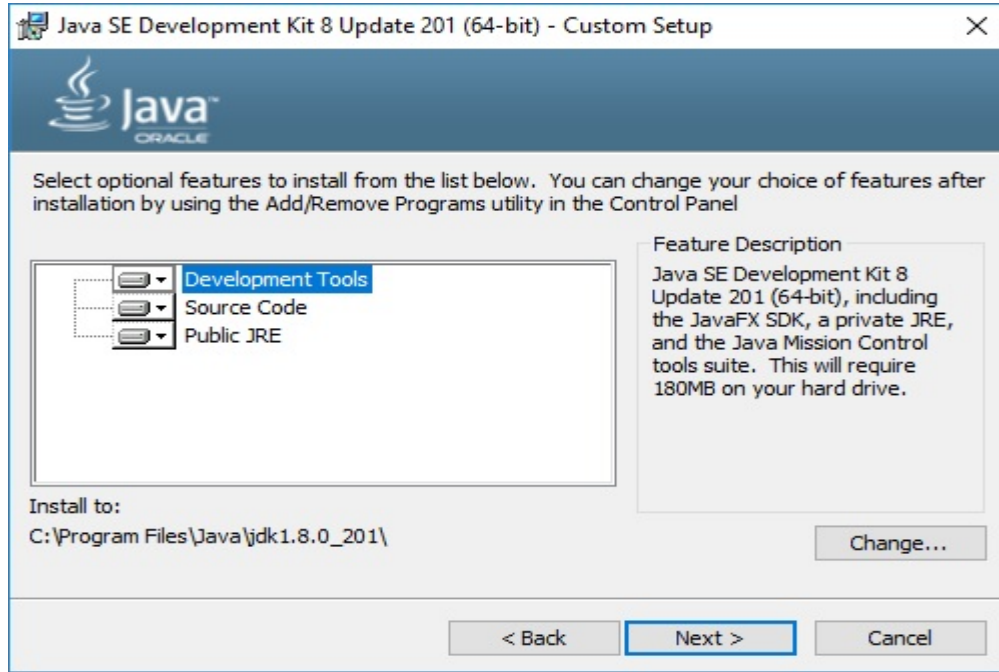
<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>

Linux x64 Compressed Archive	185.53 MB	 <a href="#">jdk-8u241-linux-x64.tar.gz</a>
macOS x64	254.06 MB	 <a href="#">jdk-8u241-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	133.01 MB	 <a href="#">jdk-8u241-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	94.24 MB	 <a href="#">jdk-8u241-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	133.8 MB	 <a href="#">jdk-8u241-solaris-x64.tar.Z</a>
Solaris x64	92.01 MB	 <a href="#">jdk-8u241-solaris-x64.tar.gz</a>
Windows x86	200.86 MB	 <a href="#">jdk-8u241-windows-i586.exe</a>
Windows x64	210.92 MB	 <a href="#">jdk-8u241-windows-x64.exe</a>

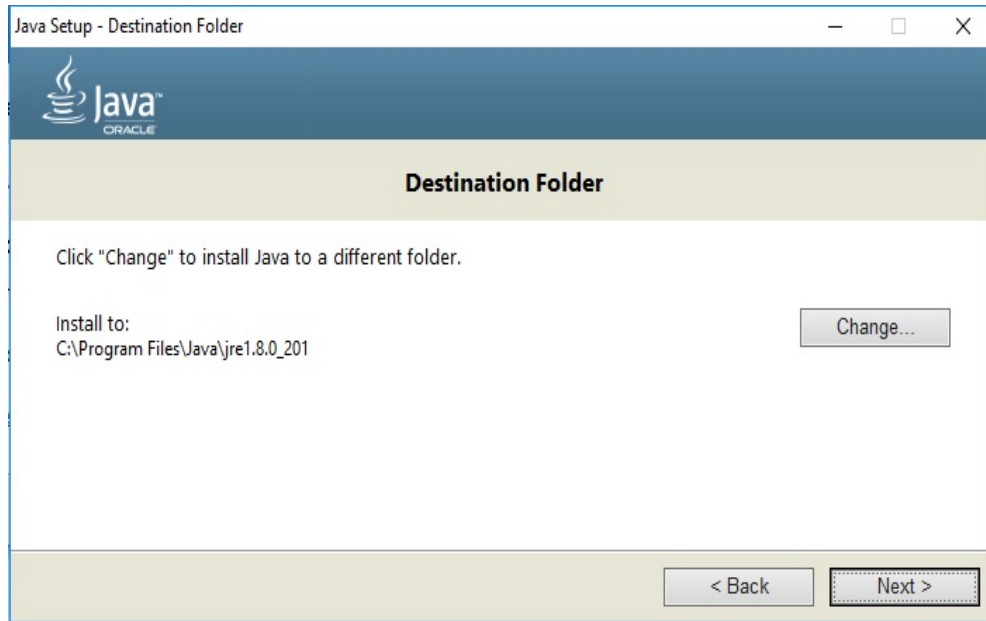
Yukarıda da görüleceği üzere işlemci tipinize göre x86 (32 Bit) / x64 (64 Bit) seçebilirsiniz. Tıklayıp sözleşmeyi kabul edip indirebilirsiniz. İndirilen .exe uzantılı dosyaya çift tıklayarak kurulumu başlatabilirsiniz.



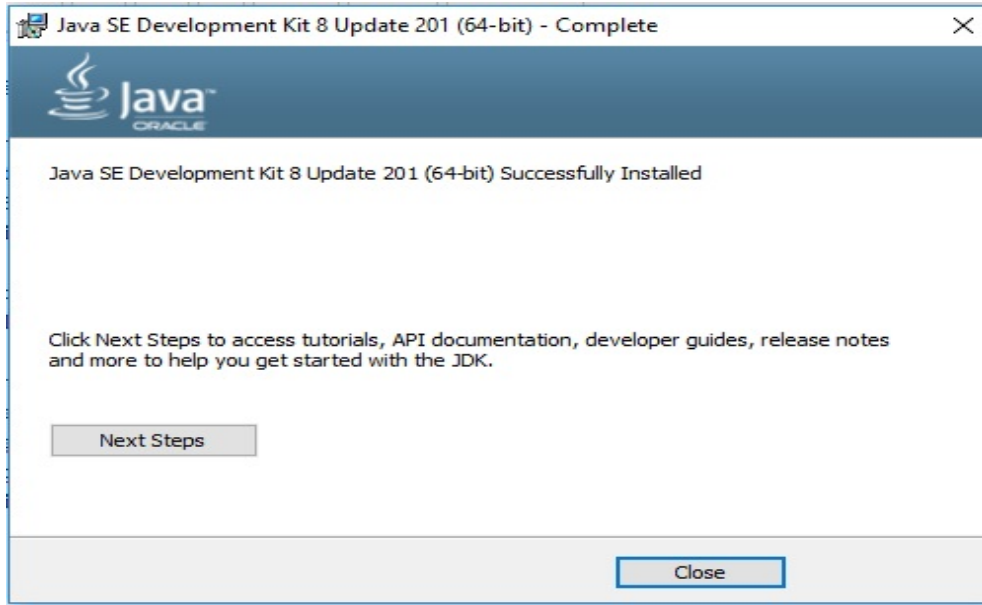
“Next” seçeneği ile kurulumu devam edilir.



Yukarıdaki gibi varsayılan ayarlar bırakılır ve “Next” ile kurulumu devam edilir.



Yukarıda Java’nın kurulacağı dosya yolu belirtilmiştir. “Change” butonuna tıklayarak dilediğiniz dosya yolu verebilirsiniz. “Next” ile kurulumu devam edilir.



JDK 8 kurulumu böylece tamamlanmış olur. Java'nın sisteme doğru bir şekilde kurulduğunu kontrol için "Windows > cmd" yoluyla Komut İstemcisi açılır.

>> java -version

```
Command Prompt
C:\Users\[username]>java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
C:\Users\[username]>
```

Komut satırı üzerinden versiyon sorgulaması yaptığınızda yukarıdaki fotoğrafta da görüldüğü gibi eğer Java başarılı bir şekilde yüklendiyse size kısa bilgiler verecektir.

## PATH ve JAVA\_HOME Tanımlamalarının Yapılması

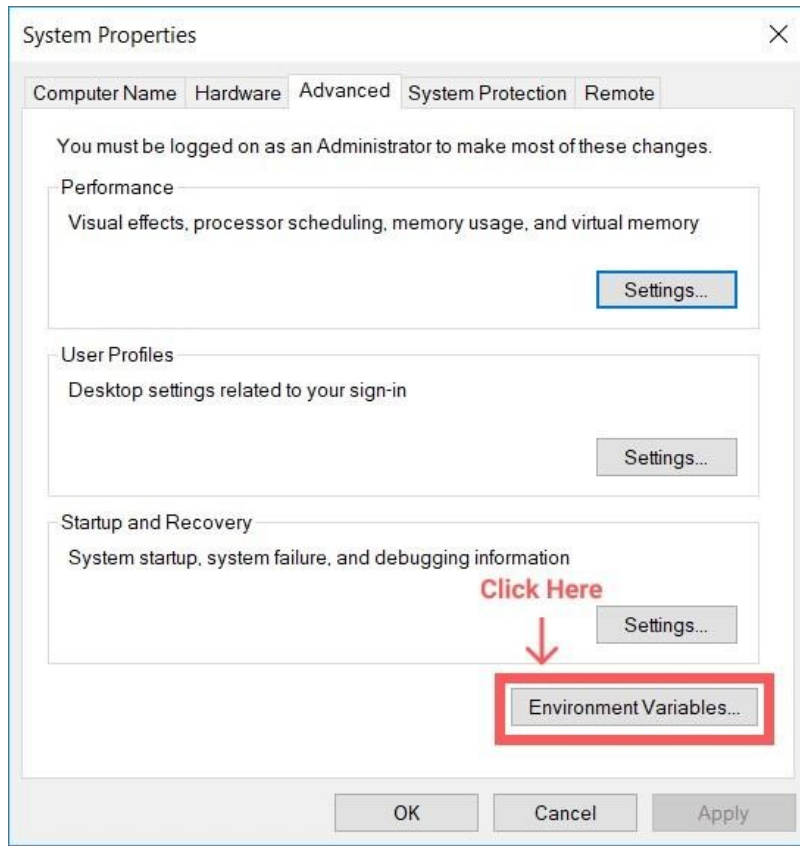
Ortam değişkenleri (Environment Variables) oluşturularak işletim sistemi düzeyinde global tanımlamalar yapılabilmektedir. Java uygulamaları da işletim sistemi tarafından

çalıştırılmaya başlandığında JAVA\_HOME ortam değişkenine ihtiyaç duyarlar. Varsayılan olarak işletim sisteminde tanımlı olan ortam değişkenini kullanmaya çalışırlar. Böylece, tüm Java uygulamalarını hatasız bir şekilde çalıştırma şansını yakalarız.

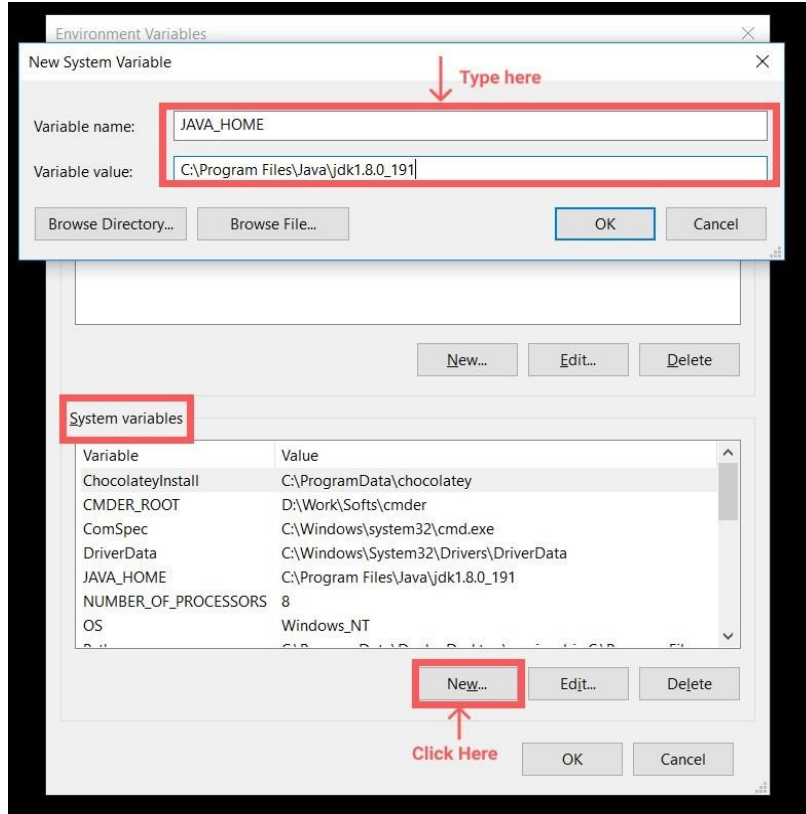
## JAVA\_HOME Ortam Değişkeni Tanımlanması

JAVA\_HOME ortam değişkeni tanımlanırken JDK'nin kurulu olduğu dosya dizinini vermeliyiz. JRE ve JDK'nın dosya dizinleri birbirinden farklıdır.

Windows arama çubuğuna “**Gelişmiş Sistem Ayarları**” veya İngilizcesi ile “**Advanced System Settings**” yazarak erişebilirsiniz. Açılan pencerede “Gelişmiş” (Advanced) tabına gelerek Ortam Değişkenleri’ni (Environment Variables) güncelleyebilirsiniz.



Ardından, “**Sistem Değişkenleri**” (System Variables) altında yer alan “Yeni” (New) butonuna tıklayarak JAVA\_HOME değişkenini, JDK dosya dizinini vererek tanımlayabilirsiniz.



## PATH Tanımının Güncellenmesi

PATH değişkenleri çalıştırabilir yazılım uygulamalarının dosya yolunu işletim sistemine belirtmek için kullanılır. Örneğin: yazılacak olan Java kodlarının derlenebilmesi için “javac” ismindeki yazılım modülüne ihtiyaç olacaktır. JDK kurulumu ile bu uygulama sisteminize yüklenmiş olur. “javac” yazılım modülünü her çalıştırmak istediğinizde tam dosya yolunu da yazmak zorunda kalırsınız.

Örneğin:

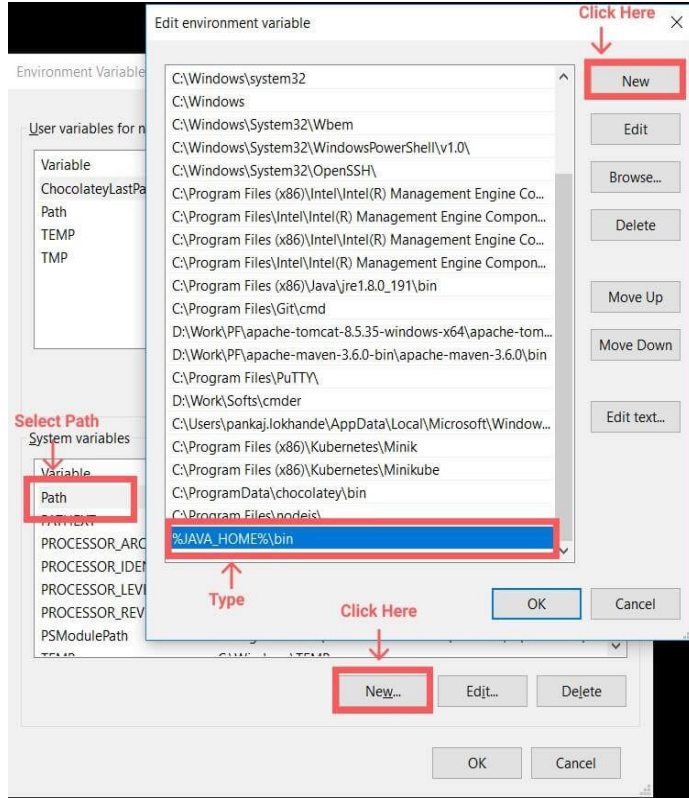
```
>> C:\ProgramFiles\Java\jdk-13.0.1\bin\javac ExampleSourceCode.java
```

Yukarıda da görüldüğü gibi “**ExampleSourceCode.java**” isimli örnek Java uygulamamızı derlemek istediğimizde tam ve kesin bir biçimde dosya yolunu vermek gerekmektedir. PATH tanımlamaları ile bu yükten kurtuluruz. “javac” ile derleme yapmak istediğimizde sadece aşağıdaki gibi bir komut çalıştırmak yeterli olacaktır.

```
>> javac ExampleSourceCode.java
```

Yine “**Sistem Değişkenleri**” (**System Variables**) içinde yer alan “Path” isimli öğeyi seçilir ve “Güncelle” (Edit) seçeneğine tıklanır. Gelen ekranda “Yeni” (New) butonuna tıklayarak yeni PATH tanımı eklenir. Bu PATH tanımında bir önceki bölümde tanımladığımız “JAVA\_HOME” ortam değişkeni kullanılabilir. JAVA\_HOME zaten JDK’nin kurulu olduğu dosya dizinini verir. “**%JAVA\_HOME%\bin%**” şeklinde sonuna “bin” gelecek şekilde ekleme yapılarak, Java işletim sisteminin PATH tanımına eklenmiş olur.





## 2. Linux İşletim Sisteminde JDK 8 Kurulumu

Linux işletim sistemi **Özgür Yazılım Lisansı**'na sahiptir. Bu nedenle Linux'u esas alan birden fazla Linux türevleri (dağıtımları) ortaya çıkmıştır. Ubuntu İşletim Sistemi üzerinde JDK 8 kurulumu açıklanacaktır.

Ubuntu'da yeni bir Terminal açılır. Ve işlemler komut istemcisinden yürütülür. Linux dağıtımlarında yazılım kurulumları, güncellemeleri ve birçok işlem Terminal (Komut İstemcisi) üzerinden halledilir.

```
>> sudo apt update
>> sudo apt install openjdk-8-jdk
```

Yukarıdaki komutlarla birlikte Ubuntu'ya JDK 8 kurulumu tamamlanmış olur. Ubuntu üzerinde yazılım kurulumu için “sudo apt install <yazılımın ismi>” şeklinde komut çalıştırılır.

```
>> java -version
```

Komutu ile Java'nın başarılı şekilde yüklendiği kontrol edilir.

**Ek Bilgi:** İşletim sisteminizde birden fazla JDK versiyonu yüklü ise “sudo update-alternatives --config java” komut ile kontrol edebilirsiniz. Varsayılan JDK versiyonunu da değiştirebilirsiniz.

## JAVA\_HOME Ortam Değişkeni Tanımlanması

Ubuntu'ta ortam değişkenleri “environment” isimli bir dosyada tutulur. Bu dosyayı güncelleyerek JAVA\_HOME tanımlanabilir.

```
>> sudo vim /etc/environment
```

Yukarıdaki komut ile bir Yazı (Text) Editör yardımıyla dosya açılır ve aşağıdaki satır en sona eklenir. Buradaki örnekte “**vim**” adlı yazı editör aracını tercih ettik. “**nano**” gibi araçlar da tercih edilebilir.

```
JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
```

## 3. Mac OS Sistemde JDK 8 Kurulumu

Mac OS üzerinde JDK 8 kurulumu için Oracle websitesinden “.dmg” uzantılı dosyayı indirerek işe başlarız. Aşağıdaki linkten indirme işlemini yapabilirsiniz.

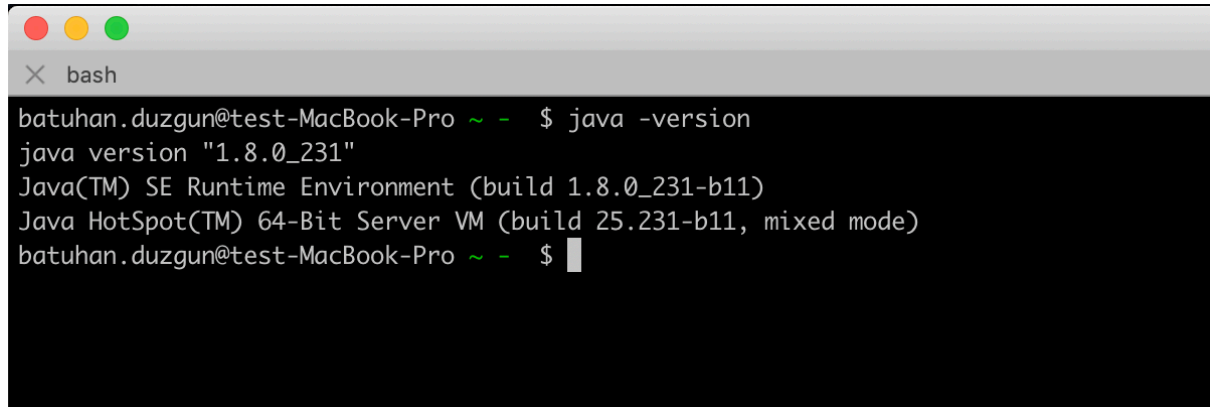
<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html#license-lightbox>

İndirilen dosyaya çift tıklayarak kurulumu başlatabilirsiniz.



Mac OS üzerinde komut istemcisi üzerinden işleri halletmek verimli ve entegrasyon esnekliği açısından faydalı denilebilir. Mac OS için “**iTerm**” Terminali tavsiye edebilirim. “**iTerm**” terminal kurulu olduğunu varsayarak veya var olan Terminali açarak Java kurulumunun başarılı bir şekilde yapıp yapılmadığı kontrol edilmelidir.

```
>> java -version
```

A terminal window with a title bar containing three colored circles (red, yellow, green) and a close button. The title bar text is 'bash'. The terminal content shows the command 'java -version' being executed, resulting in the output: 'java version "1.8.0\_231"', 'Java(TM) SE Runtime Environment (build 1.8.0\_231-b11)', and 'Java HotSpot(TM) 64-Bit Server VM (build 25.231-b11, mixed mode)'. The prompt 'batuhan.duzgun@test-MacBook-Pro ~ - \$' is visible at the end of the output.

```
batuhan.duzgun@test-MacBook-Pro ~ - $ java -version
java version "1.8.0_231"
Java(TM) SE Runtime Environment (build 1.8.0_231-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.231-b11, mixed mode)
batuhan.duzgun@test-MacBook-Pro ~ - $
```

Ardından, Windows ve Ubuntu işletim sistemlerinde yaptığımız gibi Mac OS işletim sisteminin PATH ortam değişkenini, JAVA\_HOME ortam değişkenini belirtmeliyiz.

Komut istemcisinden aşağıdaki komut ile **bash\_profile** dosyası güncellenir.

```
>> vim ~/.bash_profile
```

“vim” ile açtığımız dosyaya aşağıdaki satırları ekliyoruz.

```
export
JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk1.8.0_231.jdk/Contents/Home"
```

Yukarıdaki ifade ile **JAVA\_HOME** ortam değişkenini işletim sistemi düzeyinde tanımladık.

```
export PATH="$PATH:$JAVA_HOME/bin"
```

Yukarıdaki ifade ile Windows işletim sistemindeki gibi **PATH** ortam değişkenine Java tanımını ekliyoruz.

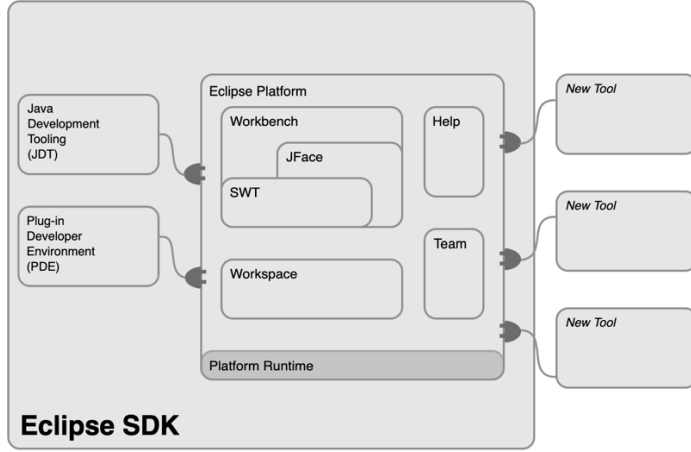
## 4. Eclipse IDE Kurulumu ve Mimarisi

Eclipse IDE, Java projeleri geliştirmenizi sağlayan bir geliştirici araçtır. 2001 yılında IBM Kanada’da hayatına başlamıştır. 2004 yılında kurulan Eclipse Vakfı (Foundation) ile ivme kazanarak Java geliştirme dünyasında en çok tercih edilen geliştirme ortamlarından biri haline geldi. Bizler de eğitim sürecinde Eclipse geliştirme aracını kullanacağız. Eclipse, Eclipse Public License (EPL) isimli açık kaynak kod lisansına sahiptir. Ücretsiz bir dağıtımdır. Oldukça fazla plug-in desteğine sahiptir. Eclipse’in sahip olduğu arayüz (Standard Widget Toolkit) SWT isimli teknoloji ile geliştirilmiştir. Eclipse plug-in’lere dayalı bir mimariye sahiptir.

Not: IDE, bütünleşik geliştirme ortamı anlamına gelmektedir.

## Eclipse Mimarisi

Eclipse temel bir platform üzerinde, tüm yeni işlevleri ve özellikleri bu temel alt yapı üzerine eklenen eklentiler (plug-in) ile sağlanmaktadır. Böylece modüler bir mimariye sahiptir. Eclipse modülerliğini sağlamakta olan altyapı OSGi standartlarındaki “Equinox” yazılım altyapısıdır.



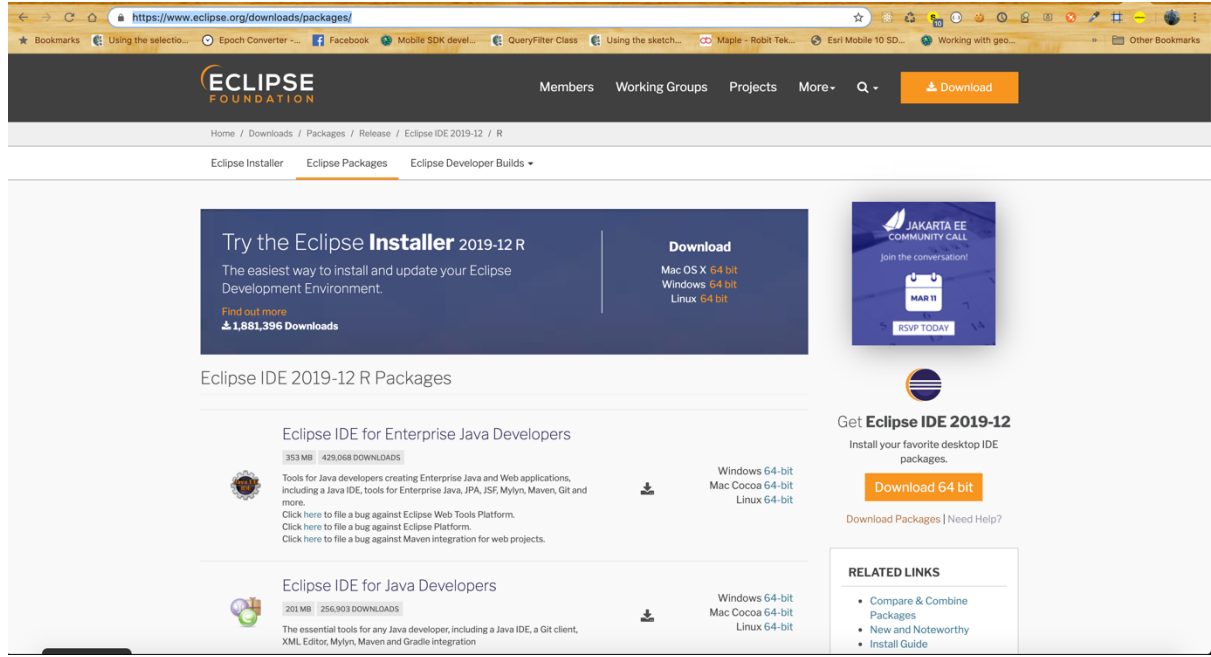
Yukarıdaki mimaride de görüleceği üzere Eclipse Platform adı altında ana bir altyapıdan oluşur. SWT, JFace gibi teknolojiler bu çekirdek yapı içinde yer alır. Platform Runtime katmanını ise OSGi ile var olan Equinox altyapısını sağlar. Equinox ile eklentiler sisteme entegre edilip çalıştırılabilir. Bu da eklenti tabanlı modüler bir mimarinin önünü açar.

Java Development Tooling bizlere kod düzenleme araçları, Java projeleri oluşturmak için gerekli araçları sağlar. Bu eklenti Eclipse çekirdek yapı içinde yer alan önemli bir eklentidir. “New Tool” isminde belirtilen modüller mimariye entegre edilebilen yazılım eklentileridir.

## Eclipse Kurulumu

Aşağıdaki indirme linkinden dilediğiniz bir versiyonu, dilediğiniz işletim sistemi için indirebilirsiniz. Eclipse kurulumu çok basittir. İndirdiğiniz sıkıştırılmış dosya içinden çalıştırılabilir dosyalar çıkacaktır. Direkt çift tıklama ile Eclipse’i çalıştırabilirsiniz.

İndirme linki: <https://www.eclipse.org/downloads/packages/>



# Java ile İlk Uygulama

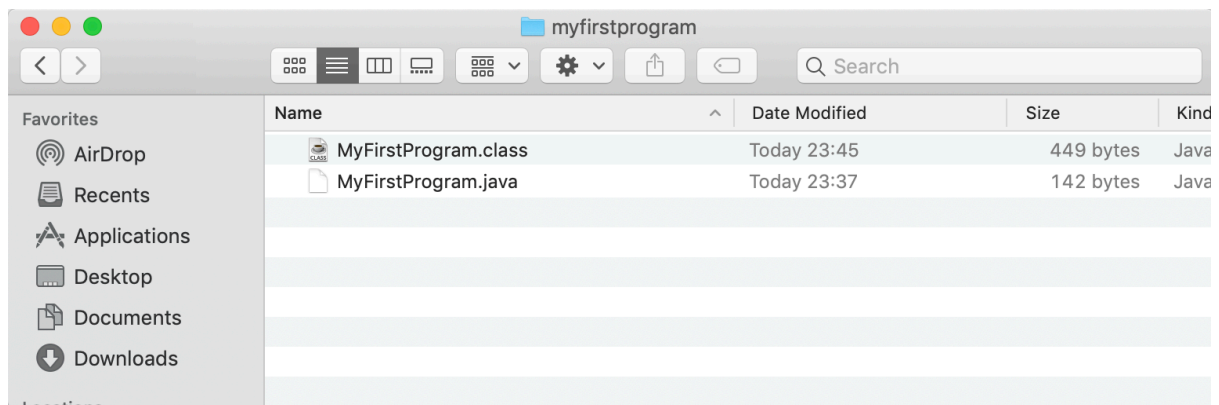
Eclipse IDE'yi indirmiş olmamıza rağmen, herhangi bir IDE olmadan da yazılan Java kodları derlenip, çalıştırılabilmektedir. Bunun için bir yazı editörü ve Java'nın sisteminizde yüklü olması yeterlidir. İlk örneğimizi bu yolla oluşturup, Java altyapısının kodu derleme, byte code'a (ara koda) dönüştürme ve sonrasında JVM vasıtasıyla yorumlayıp çalıştırılabilmesinden bahsedeceğiz. Ayrıca, yazdığımız basit Java programı hakkında kısa bilgiler vereceğiz.

Önce bir yazı editörü açıp içine aşağıdaki kodları yazıyoruz.

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

Dosya “**.java**” olarak kaydedilir. Java’da kaynak kod dosyaları “**.java**” uzantılıdır. Ardından, Java kaynak kodumuzu derlemek (Compile) için JDK içinde yer alan “**javac**” isimli yazılımdan faydalanırız. Derleme işlemi sonrasında byte code’a çevrilmiş “**.class**” uzantılı bir dosya üretilecektir.

```
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $ javac MyFirstProgram.java
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
```



byte code’a çevirme işleminden sonra, yazdığımız uygulamayı çalıştırmak istersek artık “**MyFirstProgram.class**” dosyasını kullanacağız. Java Virtual Machine (JVM), **ClassLoader** vasıtasıyla bu Class dosyasını yükleyecek, ardından byte code’u kontrolden geçirecek ve makine koduna çevirip çalıştırılmasını sağlayacak.

```
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $ java MyFirstProgram
Hello Java! | Merhaba Java!
batuhan.duzgun@test-MacBook-Pro ~/Documents/myfirstprogram - $
```

Derlenmiş olan “**MyFirstProgram.class**” dosyasını JDK içinde var olan “**java**” yazılımı ile çalıştırıyoruz. Kaynak kodun içinde yazdığımız gibi ekrana “**Hello Java! | Merhaba Java!**” ifadesini basmaktadır.

Programdaki kodlara yakından bir bakalım. Bahsettiğimiz kavramlar ileride daha detaylı bir şekilde açıklanacaktır.

“**class**” anahtar sözcüğü ile Java’da bir sınıf tanımı yaparız. Java, Nesneye Dayalı Programlama paradigmasını tamamıyla destekleyen bir dil olduğu için Java’da her şey nesnelerden oluşur. Nesne’leri ise sınıflar vasıtasıyla adeta bir taslak olarak modelleriz. Nesne ise bu taslaktan oluşturulmuş varlığı ifade eder. “class” bir sabun kalıbı ise, kalıptan üretilen her sabun ise o sınıftan üretilmiş nesnelerdir.

“**public**” anahtar sözcüğü Java’da erişim belirteci olarak kullanılır. Aslında, herkes tarafından erişilebilmeyi ifade eder. Eğer, “private” olarak tanımlasaydık bu durumda dışarıdan erişilemez olacaktı.

“**static**” anahtar sözcüğü Java’da sınıftan herhangi bir nesne üretmeden ilgili değişkeni veya metodu çağırabilmeyi ifade eder. Örneğin: `MyFirstProgram.createMagicObject()` şeklinde nesne üretmeden direkt sınıf üzerinden bir çağırım yapabiliyorsak bu static bir metoddur.

Sınıflar içinde değişkenler ve metodlar / fonksiyonlar yer almaktadır. Böylece bir sınıfı şekillendirebiliriz. Değişkenler daha çok nitelikleri ifade ederken, örneğin: arabanın rengi, uzunluğu, genişliği gibi, metodlar / fonksiyonlar ise daha çok eylemleri ifade eder. Örneğin: `startComputerMemoryCheck()` isminde bir fonksiyon bilgisayar ait hafızayı kontrol etme eylemini ifade eder. Eylemler ve nitelikler gerçek hayattaki nesneleri modellememizde de kullanılır. Bu nedenle nesneye dayalı yazılım paradigması gerçek hayat problemlerinin çözümüne doğal olarak çok uygundur.

“**void**” bir metod değer döndürmeyecekse bu anahtar sözcük kullanılır. Fonksiyonlar bünyelerinde belirli aksiyonları icra ettikten sonra eğer ortaya bir sonuç çıkıyorsa bunu çağrıldıkları yere döndürebilirler. Bunlara değer döndüren fonksiyonlar denir.

“**main**” ana programın fonksiyon ismini ifade eder.

“**String[] args**” Java programı komut istemcisinden çalıştırılırken komut istemcisinden programa ekstra parametreler gönderebilirsiniz. Bu ekstra değerler “args” isimli değişken vasıtasıyla gelir.

“**System.out.println()**” JDK içinde tanımlı statik bir fonksiyondur. Konsol ekranına bir şeyler yazdırmak istenirse kullanılır.

## JDK, JRE ve JVM Kavramları

### 1. Java Virtual Machine (JVM)

JVM aslında soyut bir kavramı ifade eder. JVM fiziksel olarak ortada var olan bir yapı değildir. Doğruyu söylemek gerekirse, Java Byte kodunun bilgisayarın en temel komutlarına nasıl dönüştürüleceğini, bu süreç içinde nelerin yapılması gerektiğini kurallar ile taslak olarak ifade eder. Aslında bir sistemin dokümantasyonu gibidir.

JVM, Java Byte koda dönüştürülebilen her yazılım geliştirme dilini çalıştırabilme yeteneğine sahiptir. Örneğin: Scala dilinde yazılmış bir kaynak kod Java Byte dönüştürülebildiği için JVM tarafından Java dilinde yazılmış bir kod parçası gibi işletilebilir.

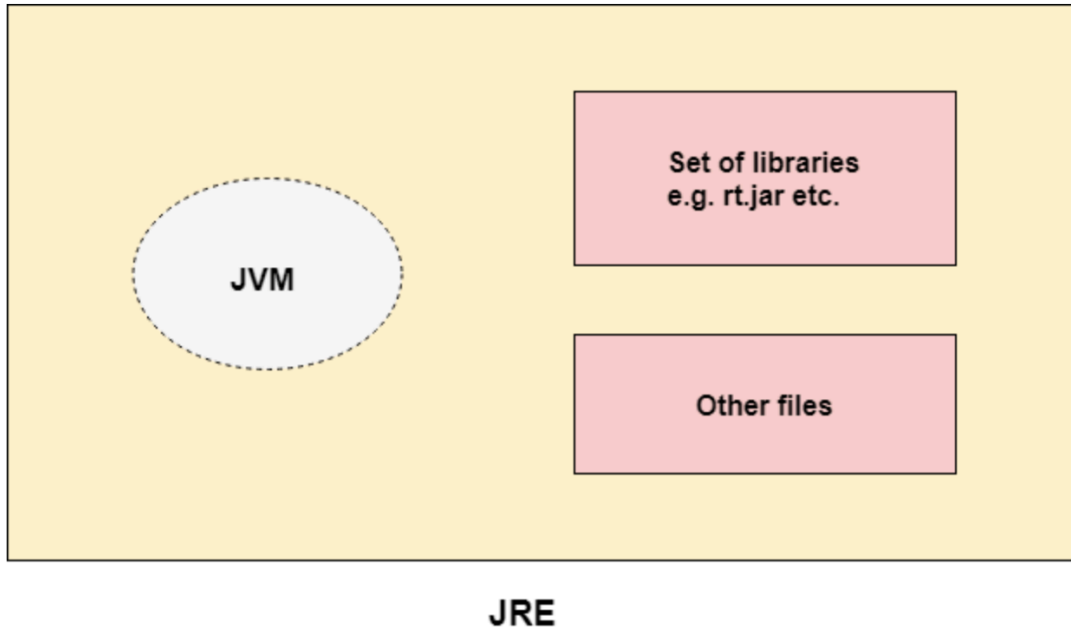
JVM esas olarak aşağıdakileri yerine getirir:

- Kodu yükler, Classloader vasıtasıyla bu işi yapar.
- Hafıza yönetimiyle ilgili olayları yönetir. Heap/Stack, Class hafıza bölgeleri gibi alanların koordinasyonunu sağlar.
- Ara kodu kontrolden geçirip onaylar. Ara kod içinde sorun olabilecek kodları arar.
- Ara kodu alıp bilgisayarın anlayacağı temel komutlara dönüştürür ve programın çalışmasını sağlar.

## 2. Java Runtime Environment (JRE)

JRE, Java ile yazılmış uygulamaların çalıştırılabilmesini sağlayan gerekli araçları ve kütüphaneleri barındırır. Aslında, Java için uygulama çalıştırma ortamı sağlar. JRE içinde halihazırda yazılım modülleri bulunur. Aynı zaman JVM spesifikasyonuna ait bir JVM implemantasyonu da barındırır. JVM'in fiziksel hali bünyesinde yer alır.

JRE, JVM'in fiziksel olarak ortaya çıkmış halidir diyebiliriz.



## 3. Java Development Environment (JDK)

JDK ise biz yazılımcıların Java programlama diliyle uygulamalar geliştirmesini sağlayan tüm altyapıyı sağlar. Bu alt yapı içinde JRE'yi de içinde bulundurur. Buna ek olarak geliştirme yapabilmek için gerekli olan yazılım modüllerini ve kütüphaneleri de JDK içinde yer almaktadır.

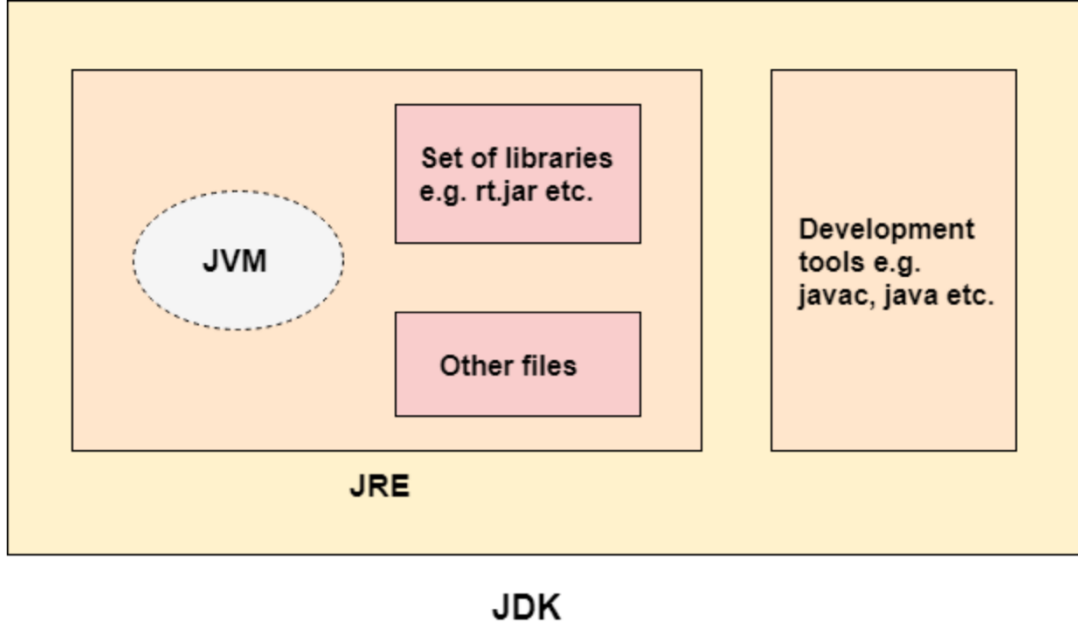


JDK üç tipte olabilir:

Standard Edition Java Platform (Java SE)

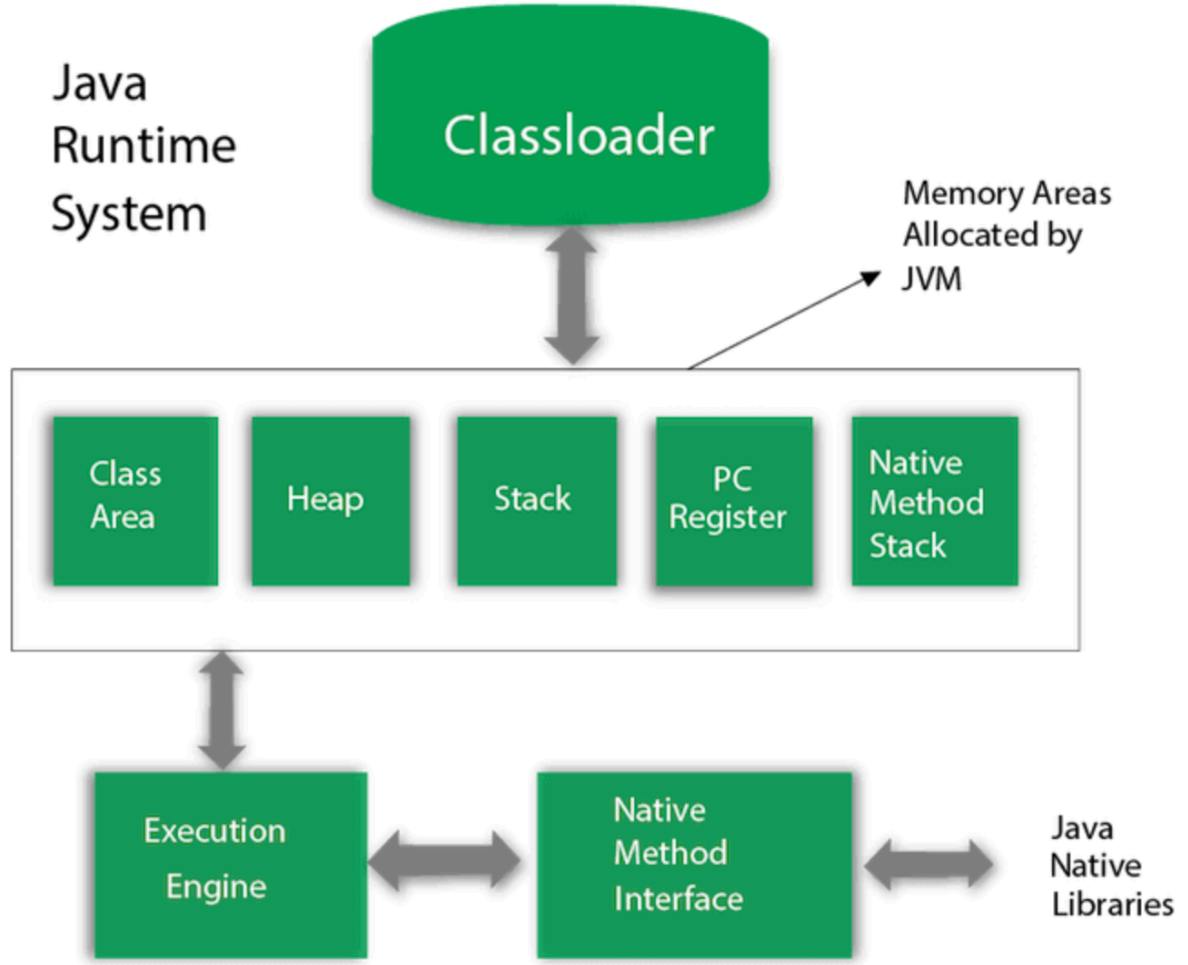
Enterprise Edition Java Platform (Java EE)

Micro Edition Java Platform (Java ME)



## Java Virtual Machine Mimarisi

JVM'in Java Byte kodunu alıp işleyip bilgisayarın temel komutlarına dönüştürerek çalıştıran bir sistem olduğunu öğrenmiştik. Her işletim sistemi tipi için bir JVM implemantasyonu vardır. Windows, Mac OS ve Linux gibi işletim sistemleri için JVM'in çalışan halleri mevcuttur. Bu nedenle platform bağımsız bir özelliğe sahiptir. Windows işletim sistemi üzerinde geliştirilmiş bir Java uygulamaları Java Byte'a çevirilir. Ardından, oluşan Java Byte kodu Linux işletim sistemi üzerinde bir JVM vasıtasıyla çalıştırılabilir.

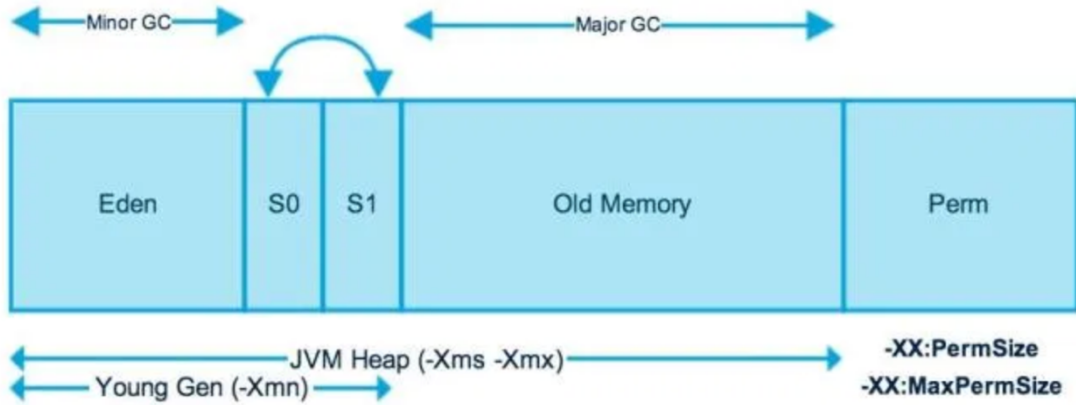
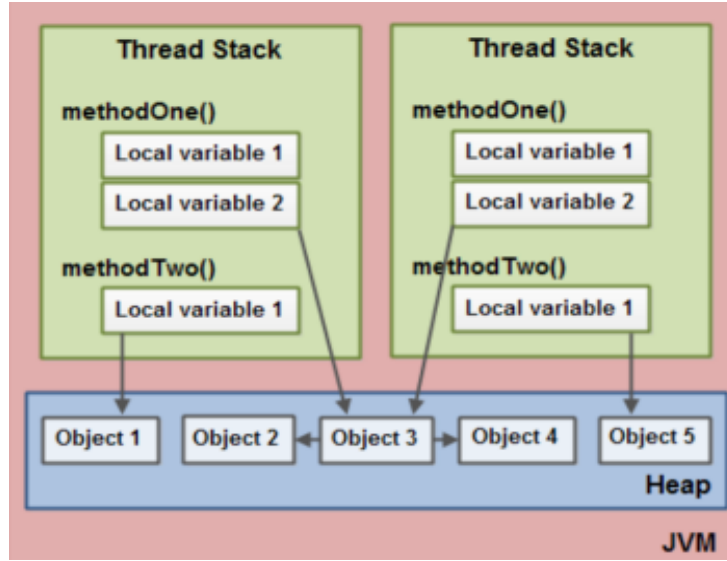


Yukarıda JVM'in kabaca mimarisini özetlemiş oluyoruz.

Classloader ile derlenmiş Java kodları, ki bunlar Java Byte'larıdır, JVM vasıtasıyla yüklenir. Ardından Classloader, JVM Hafıza bölgesine bu kodlar aktarılır.

JVM'in kendine ayırdığı hafıza bölgesi belli alanlardan oluşur. Class Area denilen kısım yüklenen sınıflarla ilgili yapıları depolar. Metoda ait kod bloğunu, kurucu metodu ve metoda ait verileri saklar.

Heap hafıza alanı ise Java'da sınıflardan oluşturulan nesneleri depolar. "new" anahtar kelimesiyle bir nesne yaratıyorsak bu Heap hafızada saklanır. Her metod çağrımında Stack hafıza bölgesinde bir alan oluşturulur. Metod çağrımı tamamlandığında ise metod çağrımına ayrılan bu alan geri iade edilir.



## Java Programlama Dili'nin Genel Özellikleri

Java, dünyada web, mobil ve daha birçok alanda yaygın şekilde kullanılan bir programlama dilidir. Özellikle, kurumsal düzeyde büyük yazılım projeleri gerçekleştirmek için çok uygundur. Java ile **Özgür Yazılım Lisansı**'na versiyonları kullanabilirsiniz. **OpenJDK** buna en güzel örnektir. Ayrıca, Java diliyle yazılım dünyasında ticari ve özgür yazılım lisansına sahip çok fazla geliştirme aracı mevcuttur. Java'yı güçlü kılan da arkasında bu büyük topluluklardır.

Java dilinde Sınıf (Class) kavramı en önemli özelliktir. Java'da her şeyi Sınıf şeklinde tanımlamanız gerekmektedir. Sınıflar bir nesneyi tanımlayan şablonlardır. Örneğin: Kahve Makinesi gerçek hayatta bir nesneyi ifade eder. Yazılım dünyasında bu gerçek hayat

nesnesini tasarlarlarken, “KahveMakinesi” isminde bir şablon tasarlarız. Bu şablon gerçek hayatta var olan nesneye ait tüm özellikleri bünyesinde barındırır. Bu nesneyi ele aldığımızda renk, uzunluk, genişlik, ağırlık gibi nitelikleri vardır. Bu özellikler tasarladığımız “KahveMakinesi” isimli sınıfta birer değişkeni ifade eder. Suyu ısıtma, kahveyi karıştırma, kahveyi pişirme gibi eylemler ise fonksiyonları ifade eder. Gerçek hayat nesnesine ait bu eylemler “KahveMakinesi” isimli sınıfta birer fonksiyon olarak tanımlanır. Böylece, yazılım dünyasında kahve makinesini modellemiş oluruz. Bu modelden veya şablondan üretilenlere de nesne denilmektedir. “KahveMakinesi” isimli sınıftan bir fabrikada üretiliyormuş gibi onlarca nesne oluşturulabilir. Her üretilen yazılımsal açıdan farklı bir nesneyi ifade eder.

**Sınıf:** Gerçek hayatta yer alan bir nesnenin yazılım dünyasındaki model halidir.

**Nesne:** Tasarlanan bu modelden üretilenlere de nesne denilmektedir.

**Metot (Fonksiyon):** Fonksiyonlar eylemleri işaret eder. Örneğin: Printer isimli bir sınıfa ait “print” isimli eylem yazılım dünyasında bir metodu ifade eder. Metotlar belli parametreler alarak veya almadan belli bir üreten veya üretmeyen kod parçalarıdır.

**Değişken:** Değişkenler nesnenin niteliklerini işaret ederler. Örneğin: Printer isimli sınıfa ait “color” isimli nitelik bir değişkeni ifade eder. Eğer rengi beyazsa beyaz bir değer alır.

Java dilinde de sınıflar tasarlanır, bu sınıflardan nesneler üretilir. Bu nesneler birbirine bağımlılıklar kurarlar ve böylece birbiriyle iletişim kurarak bir yazılımı meydana getirirler. Lego parçalarıyla büyük bir uçak yapmak gibi ☺

## 1. Dilin Temel Kuralları

- Java dilinde sınıf, metot, değişken gibi isimlendirme yapabildiğiniz her şey büyük küçük harf duyarlılığına sahiptir. Yani, Java’da “invoicePrice” ile “InvoicePrice” birbirinden farklı değişkenlerdir.
- Sınıf isimlendirmesi yapılırken hep ilk harf büyük olacak şekilde bir tanımlama yapılır. Dilde böyle bir kısıtlama olmasa da okunabilirlik için bu genel kabul görmüş bir yaklaşımdır. Örneğin: “class **Printer**” gibi.
- Metot (Fonksiyon) isimlerinin ilk harfi küçük ile başlayacak şekilde, Camel (Deve) Case stilinde bir tanımlama yapılır. Örneğin: “void **printInvoice()**” gibi.
- Java’da sınıfa ait kaynak kodlar “.java” uzantılı dosyaların içine yazılır. Bu dosya ismi sınıf ismi ile aynı olmalıdır. Örneğin: “Printer.java” ise sınıf tanımı “class Printer” şeklinde olmalıdır.
- Java’da bir programın çalışması için bir başlangıç metoduna ihtiyaç duyar. “public static void main(String args[])” metod başlangıç metodudur.

## 2. Java’da Tanımlayıcı İsimlendirme Kuralları (Identifier)

Yazılım geliştiricinin sınıf, metot ve değişken gibi yapılar için yaptığı isimlendirmelere tanımlayıcı denilmektedir. Örneğin: oluşturulan sınıfa “CustomerService” isminin verilmesi gibi. Java dilinde isimlendirme yapılırken dikkat edilmesi gereken belli başlı hususlar vardır. Bunları aşağıda listelenmiştir.

- Java’da isimlendirme yaparken ilk karakter harf, \$ işareti ve \_ (alt çizgi) ile başlayabilir. Rakam ve sayı isimlendirmenin başında yer alamaz. İlk karakteri belirtilen şekilde kurallara uygun verirsiniz geri kalanında dilediğiniz gibi isimlendirmeye devam edebilirsiniz.
- Java diline özel ayrılmış anahtar kelimeler vardır. Bunları da değişken ismi olarak kullanamazsınız. Örneğin: class, int, for, while gibi anahtar sözcükler tek başlarına bir tanımlayıcı olarak kullanılamazlar.
- Java’da yaptığınız isimlendirme büyük küçük harf duyarlılığına sahiptir. “maxAge” ile “MAXAGE” iki ayrı tanımlayıcıyı ifade etmektedir.

### 3.Java’da Düzenleyiciler (Modifiers)

Java’da tanımladığınız sınıflara, metotlara, değişkenlere başka kod bloklarından erişimi düzenleyen anahtar kelimelere düzenleyiciler (modifier) denilmektedir. Bu belirteçler ikiye ayrılmaktadır.

- Erişim Belirteçleri: **default, public, private, protected** gibi anahtar kelimeler ile ifade edilir. Bu belirteçler yazılan sınıfın, metotun kendi kapsamı dışında bir noktadan erişimini yönetir. Örneğin: “public” anahtar sözcüğü ile erişime açılır. “private” ile dışarıdan erişime kapatılır.
- Erişim Dışındaki Belirteçler: **final, abstract, strictfp** gibi anahtar kelimeleri ifade edilir. Bu sözcükler erişim ile ilgili değildir. Örneğin: “final” sözcüğü sınıfın, metotunun veya değişkenin davranışını etkiler.

### 4.Java’da Değişkenler (Variables)

Java’da bir veri parçasını, ki bu sayı, yazı alanı veya kompleks veri olabilir, hafızada tutmaya yarayan yapılardır. Örneğin: “**int year = 2019;**” gibi tanımlama yaptığımızda hafıza bir alan bu değer için ayrılır. 2019 verisi hafıza bu adrese yazılır. Değişken, hafıza adresini işaret eder. Aslında, hafıza adresinin kendisini belirtir.

Değişkenler üç tiptedir:

- Yerel Değişkenler
- Sınıf Değişkenleri (Statik olanlar)
- Nesne Değişkenleri (Statik olmayanlar)

### 5.Java’da Diziler (Arrays)

Dizi kavramı liste halinde veriler tutmaya yarayan veri yapısıdır. Örneğin: 12 aya ait şirket bilançosu dizi şeklinde tutulabilecek bir veridir. Java’da diziler oluşturulduğunda Heap Hafıza’da tutulurlar. Dizi konusu ileride detaylıca işlenecektir.

## 6.Java’da Enum Yapılar (Enum)

Java’da belirli seçenekleri önceden tanımlanabilen yapılar şeklinde kodlayabilirsiniz.

Örneğin: mağazanın müşteri iki tipte ise bunları ENTERPRISE, INDIVIDUAL şeklinde iki tanımlı değerle ifade edebilirsiniz. Böylece, koddaki okunabilirliği arttırabilirsiniz. Müşteri tiplerini 1 ve 2 gibi sayılarla ifade etseydik, kodun başkası tarafından okunabilirliği azalırdı.

## 7.Java’da Anahtar Sözcükler (Keywords)

Aşağıdaki anahtar sözcükler Java dilinde ayrılmış kelimelerdir. Bu kelimeleri tek başlarına isimlendirmede kullanamazsınız. Örneğin: “**int class = 5;**” şeklinde bir tanımlama yapamayız. Çünkü, class ayrılmış bir anahtar sözcüktür.

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

## 8.Java’da Yorum Alanları (Comments)

Java dilinde kodların arasına açıklayıcı metinler eklenebilir. Kod parçasının ne iş yaptığını anlatan veya bilgi veren yorumlar eklenebilir. Bu yorumlar derleyici tarafından dikkate alınmaz. Tek satırlık veya çok satırlı yorumlar ekleyebilirsiniz.

```
public class MyFirstJavaProgram {  
  
    /* This is my first java program.  
     * This will print 'Hello World' as the output  
     * This is an example of multi-line comments.  
     */  
  
    public static void main(String []args) {  
        // This is an example of single line comment  
        /* This is also an example of single line comment. */  
        System.out.println("Hello World");  
    }  
}
```

Java’da kod yazarken kodlar arasında dilediğiniz kadar boşluk bırakabilirsiniz.

## 9.Java’da Katılım (Inheritance)

Java, nesneye dayalı programlamayı destekleyen bir dil olduğu için katılım mekanizmasına sahiptir. Katılım ile bir sınıf üst sınıftan ortak özellikleri miras yoluyla kendine alabilir. Bu konu ileride detaylıca açıklanacaktır.

## 10.Java’da Ara yüzler (Interfaces)

Nesneye dayalı programlamanın en önemli kavramlarından biri de ara yüz tanıımıdır. Arayüz tanımlarıyla çok biçimliliği (**Polymorphism**) desteklenir. Ara yüz (Interface) kontrat gibi değerlendirilebilir. Örneğin: aracın frenleme sistemi, elektrik sistemi tüm araçlarda bulunması gereken özelliklerdir. Fakat, bunların çalışma biçimi araçtan araca farklılıklar gösterebilir. İşte bu özellik kümesi bir aracın ara yüzü olarak tanımlanabilir. Böylece, yeni bir araç tanımlamak isteyen herkes bu kontrata uygun bir yapı kurmalıdır.

# Java Sınıf ve Nesne Kavramları

Java, önceden de belirttiğimiz gibi Nesneye Dayalı Programlama yöntemini tamamıyla destekleyen bir programlama dilidir. Java bu nedenle aşağıdaki konseptleri destekler ve uygulanmasına olanak tanır. Bu özellikler ileriki konularda tek tek ele alınacaktır.

- Çok Biçimlilik (Polymorphism)
- Kalıtım (Inheritance)
- Kapsülleme (Encapsulation)
- Soyutlama (Abstraction)
- Sınıflar (Classes)
- Nesneler (Objects)
- Metot (Method)
- Mesaj Yoluyla Değer Geçmek (Message Passing)

## Java’da Nesneler (Objects)

Gerçek hayata döndüğümüzde etrafımızda yüzlerce nesne görürüz. Aslında, her nesnenin var olan bir durumu ve davranışı vardır.

Örneğin: bir köpeği ele aldığımızda rengi, ismi, cinsi köpeğe ait durumu ifade eder. Havlaması, koşmak, acıkması ise onun davranışlarını ifade eder.

Aynı şekilde köpeği modelleyen bir nesneyi yazılım dünyasında oluşturduğumuzda renk, isim, cins gibi durumu ifade eden bilgiler değişkenler ile tanımlanır. Ve bu veriler değişkenlerde saklanır. Koşmak, havlamak, acıkmak gibi davranışlar ise yazılım dünyasında metotlar (fonksiyonlar) ile tanımlanır. Metotlar nesneler arası iletişim ve etkileşim için yol sunarlar. Örneğin: bir nesne başka bir nesnenin davranışını o nesnenin metodunu çağırarak etkileşime geçer.

## Java’da Sınıflar (Classes)

Yukarıdaki örneğimizde bir köpeğin yazılım dünyasında nesneler vasıtasıyla bir modelinin olabileceğinden bahsetmiştik. Nesne modelin var olan fiziksel halidir. İşte bu noktada sınıflar bir nesnenin modellenmesi için oluşturulan taslağın kendisidir. Sınıflar bir nesneye ait taslaktır diyebiliriz. Nesne ise o taslaktan üretilmiş bir fiziksel kopyadır. Sınıflardan üretilen yazılımsal nesnelerde bilgisayar hafızasında fiziksel var olan yapılardır.

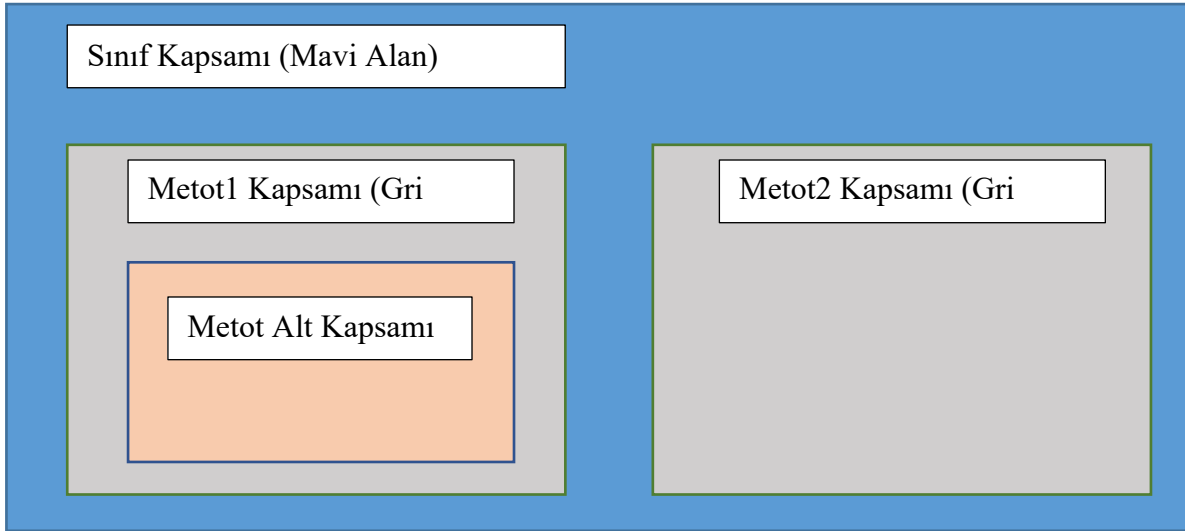
```
public class Dog {  
  
    String breed;  
    int age;  
    String color;  
  
    void barking() {  
    }  
  
    void hungry() {  
    }  
  
    void sleeping() {  
    }  
}
```



Yukarıda örnek bir sınıf tanımı vardır. Java’da sınıflar aşağıdaki yapıda tanımlanır.

```
public class <SINIF_İSMİ>
{
    // Kod bloğu
    // Bu alan sınıfın kapsamını belirler. Bu iki süslü parantez dışında
    kalan kod bloğu sınıfın dışıdır.
}
```

Süslü parantezler o sınıfa ait kod bloğunu temsil eder. Java’da her süslü parantez açıp kapatmak bir kod bloğu oluşturur. İç içe açılan her kod bloğu alt kümeler gibi birbirini kapsar. Kapsayan blok içindeki tüm alt kod bloklarına erişim hakkına sahipken, alt kapsamdaki bir kod bloğu direkt olarak bir üst kapsam bloğuna erişemez. Bunu iç içe kümeler gibi düşünebiliriz.



Değişken Tipleri:

- Yerel Değişken: Bu değişkenler metot veya bir kod kapsamı içinde tanımlanmış olan değişkenlerdir. Tanımlandıkları metodun veya kod bloğunun işi bitince otomatik olarak hafızadan silinirler. Yaşam ömürleri kod kapsamının ömrü kadardır.
- Nesne Değişkenleri: Sınıf tanımı yaparken köpek örneğinde olduğu gibi renk, boy, isim gibi nitelikleri değişkenler vasıtasıyla tanımlarız. Sınıf içinde bunları değişken olarak tanımlarız. Ardından sınıftan üretilen her nesne bu değişkenlere sahiptir. İşte bunlara nesne değişkenleri denilmektedir.
- Sınıf Değişkenleri: Sınıf tanımı yapılırken bir değişken **static** anahtar sözcüğüyle tanımlanıyorsa bu sınıf tipi bir değişkendir. Değişkene değer atamak için bir nesne oluşturmanıza gerek yoktur. Sınıf ismi ve nokta operatörü kullanarak erişilir. Global kapsama sahip değişkenlerdir.

## Java’da Sınıf Kurucuları (Constructors)

Sınıf içinde kurucu metotlar tanımlanabilir. Kurucu metotlar bir nesne oluşturulurken yapılması gerekenlerin tanımlandığı metotlardır. Nesne oluşturma aşamasında bu özel metotlar çağrılır. Her sınıfın mutlaka bir kurucu metodu olur. Eğer, siz kod yazarken kurucu metot tanımlamadıysanız Java derleyicisi boş bir taneyi otomatik olarak tanımlayacaktır.

Kurucu metotların ismi sınıf ismiyle aynı olmak zorundadır. Farklı parametreler alan birden fazla kurucu metot tanımlayabilirsiniz.

```
public class Puppy {  
  
    public Puppy() {  
        // parametresiz bir kurucu metot.  
    }  
  
    public Puppy(String name) {  
        // name isminde bir değişkenle tanımlanmış bir kurucu metot.  
    }  
}
```

Yukarıdaki örnekte Puppy isminde bir sınıfa ait iki tane kurucu metot görülmektedir. Java’da bir sınıftan yeni bir nesne üretmek istediğinizde “new” anahtar kelimesini kullanmanız gerekmektedir.

```
Puppy p = new Puppy("Pamuk");
```

Yukarıdaki şekildeki gibi Puppy sınıfına ait bir nesne oluşturmak istediğinizde dikkat ederseniz nesne oluştururken "Pamuk" şeklinde bir isim bilgisi geçtik. Bu nedenle nesne oluşturma işlemi sırasında `public Puppy(String name)` isimli kurucu metot çağrılır ve çalıştırılır.

```
Puppy p = new Puppy();
```

Yukarıdaki şekildeki gibi parametresiz bir nesne oluşturmuş olsaydık. `public Puppy()` isimli kurucu metot çağrılır ve çalıştırılırdı.

## Java’da Paketler (Packages)

Java’da sınıfları (classes) ve arayüzleri (interfaces) bir araya toplamak için paket kavramı vardır. Birbiriyle mantıksal ilişkiye sahip sınıflar bir paket altına hiyerarşik bir şekilde toplanır. Sizler de kendi Java uygulamalarınızı geliştirirken yeni paketler oluşturup birbiriyle alakalı sınıfları bir araya toplayabilirsiniz.

```
import java.io.*;
```

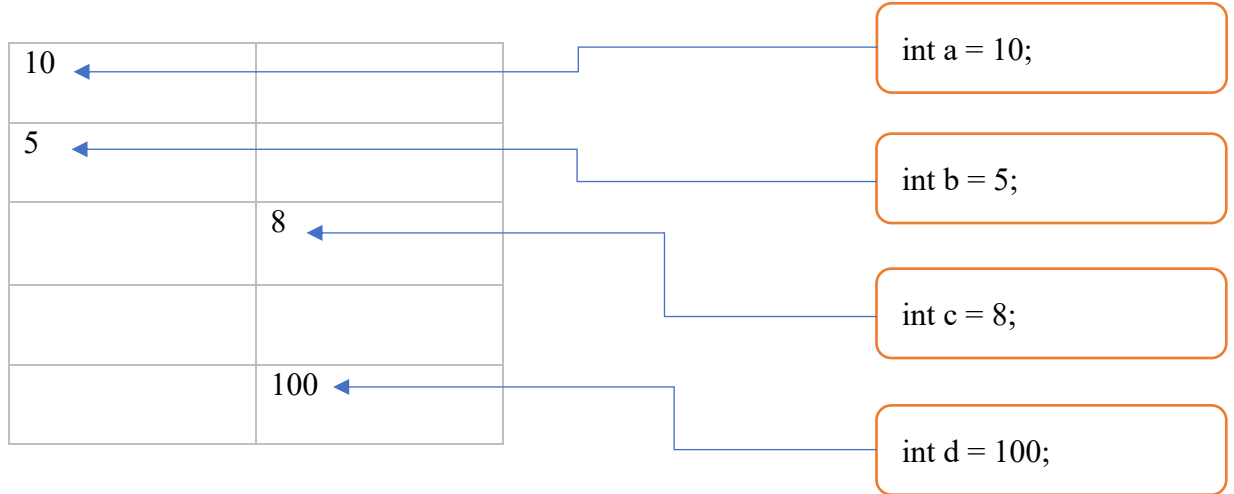
Yukarıdaki Java paketinde giriş/çıkış işlemleriyle ilgili sınıflar **java.io** isimli bir paket altına toplanmıştır. Sondaki \* işareti ise **java.io** paketi altındaki tüm sınıfları projeye dahil et anlamına gelmektedir.

## Java Değişkenler

Değişkenler içinde veri barındıran ve bilgisayarın geçici hafızasında (RAM) fiziksel olan yer kaplayan yapılardır. Değişkenlere değer (veri) ataması yapılabilir. Java'da değişkenleri veri tipleri vardır. Bu tipler Java'da varsayılan olarak tanımlı gelen tipler de olabilir yahut yazılımcıların kendi tanımladığı tipler de olabilir.

```
<veri tipi> <değişken ismi> = veri (değer)
```

Değişken tanımlaması yapıldığında aslında bilgisayar hafızasında bir yeri ayırmış oluyoruz. Bu alan o değişkenin veri tipinin boyutu kadar bir alanı ifade eder. Örneğin: 2 Byte'lık bir veri tipine sahipsek ve bu tipte bir değişken tanımlıyorsak. Her değişken için hafızadan 2 Byte'lık yer ayrılacaktır.



**int** tipinde, yani sayı tipinde tanımlanmış **a,b,c,d** isimli değişkenlerin her biri hafızada bir alanı kaplarlar.

Değişken tanımlama örnekleri:

```
int a, b, c;           // 3 tane değişken virgüller ile ayrılarak tek satırda
                        tanımlanabilir.
int a = 10, b = 10;    // Birden fazla değişken aynı satırda ilk değerleri
                        atanarak tanımlanabilir.
byte b = 22;           // Tek değişkene ilk değer ataması yapılarak
double pi = 3.14159;   // Tek değişkene ilk değer ataması yapılarak
char a = 'a';          // Tek değişkene ilk değer ataması yapılarak
```

```

public class ConnectionPool
{ // Sınıf kapsamının başlangıcı

    int connectionMaximumLimit = 50; // Nesne değişkenidir.

    static int currentActiveConnectionCount = 10; //static değişkendir.
    Sınıf değişkenidir.

    public void acquireConnection()
    { // metot (fonksiyon) kapsamının başlangıcı

        int processId = 90; // Yerel değişkendir.

        // Diğer kodlar ...

    } // metot (fonksiyon) kapsamının sonu

} // Sınıf kapsamının sonu

```

Yukarıdaki örnekte veritabanına bağlantı kurabilmek için bir havuz oluşturduğumuzu hayal edelim. Bu havuzaa belli bir sayıda kullanıcı bağlanıp bir bağlantıyı alıp kullanıp, tekrar sisteme iade ettiğini düşünelim. Bu senaryoda “ConnectionPool” isminde bir sınıf tanımlamak gerekecektir. Bu sınıf havuz nesnesinin taslağıdır. Kapsamı süslü parantezlerle başlayıp bittiği alan kadardır. Bu kısım kod üzerinde açıklama satırlarıyla belirtilmiştir. “ConnectionPool” sınıfı içindeki “connectionMaximumLimit” isimli değişken nesne değişkenidir. Bu sınıftan üretilen her nesnenin kendine ait “connectionMaximumLimit” bir değişkeni olacaktır. “static” olarak isimlendirilen “currentActiveConnectionCount” değişkeni ise sınıf değişkendir. Yani herhangi bir nesne üretmeksizin sınıf üzerinden global olarak erişilebilir.

Yani,

```

ConnectionPool.currentActiveConnectionCount = 1000;

```

Yukarıdaki şekildeki gibi nesne olmadan sınıf tanımı üzerinden erişilebilir.

“acquireConnection” metodu ise havuzdan bağlantı alıp ilgili prosese atamayı sağlayan fonksiyondur. Bu fonksiyon içinde tanımlı olan tüm değişkenler yerel değişkendir. Çünkü, metodun kapsamı içinde tanımlanmıştır. Metodun kapsamı da süslü parantezler arasında kalan alandır. Yorum satırı olarak belirtilmiştir. Metot çalışıp işini bitirince bu yerel değişkenlerde ömürlerini tamamlarlar ve bilgisayar hafızasından silinirler.

```

public class VariableDemo {

    public static void main(String[] args) {

        int firstParameter = 10;
        int secondParameter = 20;

        int summation = firstParameter + secondParameter;

        System.out.println(summation);

    }

}

```

Yukarıdaki örnekte de iki adet değişken tanımlanıp “+” toplama operatörüyle toplanıp sonuç “**summation**” isimli başka bir değişkene atanmıştır. Ardından, “**println**” fonksiyonuyla konsol ekranına sonuç yazdırılmıştır.

### Yerel Değişken

```

public class Test {

    public void popAge() {

        int age = 0;
        age = age + 7;
        System.out.println("Puppy age is : " + age);

    }

    public static void main(String args[]) {

        Test test = new Test();
        test.popAge();

    }

}

```

Yukarıdaki örnekte “**popAge**” metodu içindeki “**age**” isimli değişken yerel tanımlıdır. Dikkat edilecek olunursa Test sınıfından bir nesne oluşturup “**popAge**” metodu çağırılmıştır. Sonuçta ekrana 7 değerini basacaktır. Yerel değişkenlere ilk değer ataması yapılmalıdır. “age” isimli değişkene sıfır değeri ilk değer olarak verilmiştir.

Not: Nesne değişkenlerinin varsayılan değerleri otomatik atanır. Eğer değişken sınıf (referans) tipinde bir değişkense varsayılan değeri “**null**” olacaktır.

Not: Sınıf değişkenleri daha çok sabit değerleri tanımlamada kullanılır.

```

public static final double PI = 3.14;

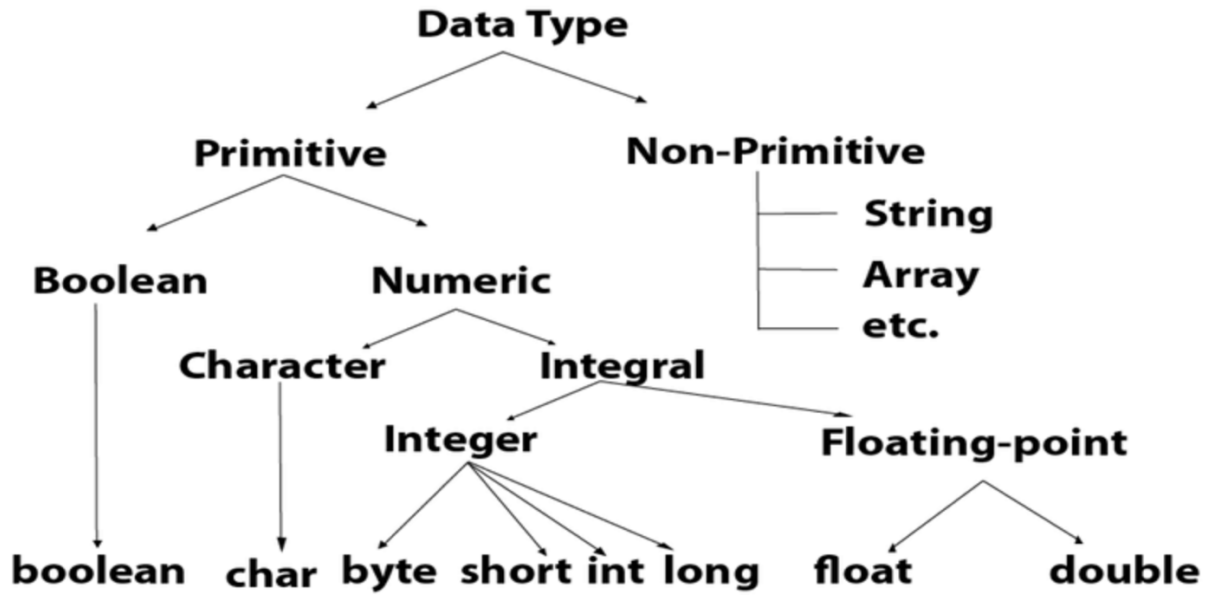
```

# Java Veri Tipleri

Değişkenler, verileri hafızada belli bir alan içinde tutmaya, saklamaya yararlar. Her değişken tanımı yapılırken bir veri tipi belirtilir. Veri tipine göre de değişken tanımlandığı esnada hafızada ne kadar yer kaplayacağı belli olur.

Java'da iki tip değişken grubu vardır:

- İlkel Veri Tipleri (Primitive Data Types)
- Nesne Veri Tipi (Object Data Types)



## İlkel Veri Tipleri

Java'da dille birlikte tanımlı olarak gelen 8 adet ilkel veri tipi vardır.

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

Veri Tipi	Varsayılan Değeri	Veri Boyutu
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

### “boolean” Veri Tipi

Bu veri tipinde sadece iki değer tutabiliriz. “true” veya “false” şeklinde iki değere sahiptir. Hafızada 1 Bit büyüklüğünde yer kaplar.

Örnek Tanımlama:

```
boolean printerEnabled = false;
```

### “byte” Veri Tipi

-128 ile 127 arasında değer alabilen sayısal bir tam sayı tipidir. Varsayılan değeri sıfırdır. Özellikle, hafızada az yer kaplaması nedeniyle kullanılabilir. Eğer, “int” tipine gerek duymuyorsanız, “byte” kullanmak faydalı olacaktır.

Örnek Tanımlama:

```
byte humanAge = 32;
```

### “short” Veri Tipi

16 Bit’lik (yani 2 Byte) veri büyüklüğüne sahip tam sayı veri tipidir. -32,768 ile 32,767 arasında değer alabilir. Varsayılan değeri sıfırdır. Yine “int” veri tipine ihtiyaç duymadığınız zaman “short” tipte değişkenler oluşturarak hafızadan kazanç sağlayabilirsiniz.

Örnek Tanımlama: `short m2OfRegion = 11991;`

## “int” Veri Tipi

32 Bit’lik (yani 4 Byte) veri büyüklüğüne sahip tam sayı veri tipidir. - 2,147,483,648 ile 2,147,483,647 arasında değer alabilir. Varsayılan değeri sıfırdır. “int” tipinde değişken tanımlarken gerçekten o kadar büyüklüğe sahip bir veri tutacak mıyız, iyi kontrol etmek gerekir. Örneğin: insan yaşı bilgisini “int” veri tipinde tutmak hafızada fazladan alan kaplamak demek olacaktır. Zaten insan yaşı “int” değerinden çok küçüktür. Bunun için “byte” tipinde bir değişken tanımlamak hafızayı etkin kullanmayı sağlayacaktır.

Örnek Tanımlama:

```
int bookCountInWorld = 1199221;
```

## “long” Veri Tipi

64 Bit’lik (yani 8 Byte) veri büyüklüğüne sahip tam sayı değeridir. Tam sayı veri tipleri içinde en büyük değer aralığına sahip veri tipidir. Çok büyük basamaklı sayıları tutabilmek için idealdir. Hafızada önemli bir yer kaplar. Kullanırken dikkatli olmak gerekir. -9,223,372,036,854,775,808 ile 9,223,372,036,854,775,807 arasında değer alabilir.

Örnek Tanımlama:

```
long galaxyCountInSpace = 51992212222;
```

## “float” Veri Tipi

32 Bit’lik (yani 4 Byte) veri büyüklüğüne sahip ondalıklı sayı değeridir. Sayı aralığına bir limit getirilmemiştir. Hafızayı etkin kullanmak adına “double” veri tipi yerine tercih edilebilir. Çünkü, “double” veri tipi “float” ‘dan daha büyük bir yer kaplamaktadır. Varsayılan değeri 0.0F şeklindedir. Float tipindeki değişkenlere atanan verilerin sonunda “f” son eki vardır. Değişkene atanan değer “float” tipinde olduğunu belirtir.

Örnek Tanımlama:

```
float freezeRatio = 3.23f;
```

## “double” Veri Tipi

64 Bit’lik (yani 8 Byte) veri büyüklüğüne sahip ondalıklı sayı değeridir. Sayı aralığına bir limit getirilmemiştir. Boyutu büyük olduğu için tanımlama yapılırken gerçekten “float” veri tipinin yetersiz olduğu durumlarda kullanılmalıdır. Varsayılan değeri 0.0d şeklindedir. Atanan verinin sonuna “d” son eki koyularak “double” tipte bir veri olduğu belirtilebilir. Fakat, “d” son ekinin konulmadığı durumlarda ondalıklı sayı verisi varsayılan olarak “double” olarak kabul edilir. Konulması zorunlu değildir.

Örnek Tanımlama: `double freezeRatio = 3.2322;`



## “char” Veri Tipi

16 Bit’lik (yani 2 Byte) büyüklüğüne sahip karakter verilerini tutar. Unicode tipinde karakter verilerini saklar. '\u0000' (0) ile '\uffff' (65535) aralığında değer alır.

Örnek Tanımlama:

```
char letter = 'A';  
char letter = 'B';  
char letter = 'C';
```

## Unicode Karakter Sistemi

Bilgisayar dünyasında karakterler sayısal ifadeler ile temsil edilir. Aslında, her harfin veya sembolün bir sayısal karşılığı vardır. Dünyadaki çoğu dile ait karakterleri tanımlayan sisteme Unicode Karakter Sistemi denir.

Unicode sistemi dışında kullanılan diğer karakter sistemleri:

- **ASCII** (American Standard Code for Information Interchange)
- **ISO 8859-1** (Batı Dilleri için)
- **KOI-8** (Rusça için)
- **GB18030** ve **BIG-5** (Çince ve diğer diller için)

Unicode sistemi ile tüm dillere ait karakterler 2 Byte şeklinde tanımlandığı için uluslararası bir standarda sahiptir. Bu nedenle Java tarafından tercih edilmiştir. Diğer karakter sistemlerinden birbirine dönüşüm yapmak sorunlu ve maliyetli bir iştir.

Notasyon	Temsil Eden Karakter
\n	Yeni satır (0x0a)
\r	Satırbaşı (0x0d)
\b	Geri tuşu (0x08)
\s	Boşluk (0x20)
\t	Bir tab Boşluk
\"	Çift Tırnak Kaçış Karakteri
\'	Tek Tırnak Kaçış Karakteri
\\	Ters slash karakteri
\uxxxx	Hexadecimal UNICODE Karakteri (xxxx)

## Java Temel Operatörler (Basic Operators)

Java dilinde operatörler ile birçok işlemi yapabilmenize olanak tanır. Örneğin: matematiksel operatörlerle birlikte aritmetik işlemler yapabilmenizi, ilişkisel operatörlerle verileri kıyaslayabilmeyi, atama operatörleri ile değişkenlerin değerlerini değiştirmeye fırsat verir.

Java'da operatörler aşağıdaki gibi listelenebilir:

- Aritmetiksel Operatörler
- İlişkisel ve Eşitlik Operatörler
- Bitisel (Bit Düzeyinde) Operatörler
- Mantıksal Operatörler
- Atama Operatörleri

# Aritmetik Operatörleri

Matematiksel işlemler yapabilmeyi sağlayan operatörlerdir.

```
int A = 10;  
int B = 20;
```

şeklinde değişkenlerimiz olduğunu varsayıp matematiksel operatörleri açıklayalım.

Operatör	Açıklama	Örnek
+ (Toplama)	İki değerin toplanıp tek bir değere dönüştürülmesini sağlar. Sayısal değerler yanında yazı tipindeki değerlerde kullanıldığında iki ifadeyi tek bir cümle haline getirmektedir.	$A + B = 30$ "Güzel" + "Kitap" = "GüzelKitap"
- (Çıkarma)	İki değerin çıkarılmasını sağlar.	$A - B = -10$
* (Çarpma)	İki değeri çarpmaya yarar.	$A * B = 200$
/ (Bölme)	Payı paydaya bölmeye yarar.	$B / A = 2$
% (Mod Alma)	Mod alma işlemini sağlar.	$B \% A = 0$
++ (Bir artırım)	Bir artırmayı sağlar.	B++ 21 olacaktır.
-- (Bir azaltım)	Bir azaltmayı sağlar.	B-- 19 olacaktır.

Örnek:

```
public class Test {  
  
    public static void main(String args[]) {  
        int a = 10;  
        int b = 20;  
        int c = 25;  
        int d = 25;  
  
        System.out.println("a + b = " + (a + b) );  
        System.out.println("a - b = " + (a - b) );  
        System.out.println("a * b = " + (a * b) );  
        System.out.println("b / a = " + (b / a) );  
        System.out.println("b % a = " + (b % a) );  
        System.out.println("c % a = " + (c % a) );  
        System.out.println("a++ = " + (a++) );  
        System.out.println("b-- = " + (a--) );  
  
        System.out.println("d++ = " + (d++) );  
        System.out.println("++d = " + (++d) );  
    }  
}
```

## İlişkisel ve Eşitlik Operatörleri

Değişkenlere ait verilerin eşitliğini, büyük-küçük olma durumlarının kıyaslanabildiği operatörlerdir.

```
int A = 10;  
int B = 20;
```

şeklinde değişkenlerimiz olduğunu varsayıp matematiksel operatörleri açıklayalım.

Operatör	Açıklama	Örnek
== (eşitlik)	İki değişkenin değerlerini birbiriyle kıyaslayıp, eğer eşitse true döndürür. Değilse false değer döndürür.	(A == B) false
!= (eşit değil)	İki değişkenin değerini kıyaslar ve eşit değilse true, eşitse false döndürür.	(A != B) true

> (büyüktür)	> operatörünün solundaki değer, sağındaki değerden büyükse true döndürür. Değilse false döner.	(A > B) false
< (küçüktür)	< operatörünün solundaki değer, sağındaki değerden küçükse false döndürür. Değilse true döner.	(A < B) true
>= (büyük eşit)	>= operatörünün solundaki değer, sağındaki değerden büyük veya eşitse true döndürür. Değilse false döner.	(A >= B) false
<= (küçük eşit)	<= operatörünün solundaki değer, sağındaki değerden küçük ve eşitse false döndürür. Değilse true döner.	(A <= B) true

Örnek:

```
public class Test {

    public static void main(String args[]) {

        int a = 10;
        int b = 20;

        System.out.println("a == b = " + (a == b) );
        System.out.println("a != b = " + (a != b) );
        System.out.println("a > b = " + (a > b) );
        System.out.println("a < b = " + (a < b) );
        System.out.println("b >= a = " + (b >= a) );
        System.out.println("b <= a = " + (b <= a) );

    }
}
```

## Bitsel Operatörler

Java'da bitsel operatörler bit düzeyinde işlemler yapabilmeyi sağlar. Bit düzeyinde VE (AND), VEYA (OR) ve DEĞİL (NOT) gibi işlemler yapabilmeyi sağlar. Ayrıca, bit düzeyinde sağa veya sola kaydırma işlemleri yapılabilir. Bitsel operatörler long, int, short, byte, char gibi veri tiplerine uygulanabilir.

Operatör	Açıklama	Örnek
& (ve)	Bit düzeyinde tek tek VE işlemini uygular. 1 & 0 ise 0 olur. 1 & 1 ise 1 olur. 0 & 0 ise 0 olur.	(A & B)
(veya)	Bit düzeyinde tek tek VEYA işlemini uygular. 1   0 ise 1 olur. 1   1 ise 1 olur. 0   0 ise 0 olur.	(A   B)
^ (XOR)	Bit düzeyinde tek tek XOR işlemi uygular. 1 ^ 0 ise 1 olur. 1 ^ 1 ise 0 olur. 0 ^ 0 ise 0 olur.	(A ^ B)
~ (tersini alma)	Bit düzeyinde her bitin tersi alınır. DEĞİL işlemi uygulanır. 1100 ise 0011 olur.	(~A)
<< (sola kaydırma)	<< operatörü ile bitset olarak sola doğru belirlenen miktarda kayma yapılır. Boş kalanları sıfır ile doldurur. Ör: A<<2 yapılırsa, 0011 1100 → 1111 0000	(A << 2)
>> (sağa kaydırma)	>> operatörü ile bitset olarak sağa kaydırma işlemi yapılır. Boş kalan alanlar sıfır ile doldurulmaz. Ör: A>>2 yapılırsa, 0011 1100 → 1111	(A >> 2)
>>> (sıfırlı sağa kaydırma)	>>> operatörü ile bitset olarak sağa kaydırma işlemi yapılır. Boş kalan alanlar sıfır ile doldurulur. Ör: A>>>2 yapılırsa, 0011 1100 → 0000 1111	(A>>>2)

Örnek:

```
public class Test {

    public static void main(String args[]) {
        int a = 60;          /* 60 = 0011 1100 */
        int b = 13;          /* 13 = 0000 1101 */
        int c = 0;

        c = a & b;            /* 12 = 0000 1100 */
        System.out.println("a & b = " + c );

        c = a | b;           /* 61 = 0011 1101 */
        System.out.println("a | b = " + c );

        c = a ^ b;           /* 49 = 0011 0001 */
    }
}
```

```

System.out.println("a ^ b = " + c );

c = ~a;          /*-61 = 1100 0011 */
System.out.println("~a = " + c );

c = a << 2;       /* 240 = 1111 0000 */
System.out.println("a << 2 = " + c );

c = a >> 2;       /* 15 = 1111 */
System.out.println("a >> 2 = " + c );

c = a >>> 2;      /* 15 = 0000 1111 */
System.out.println("a >>> 2 = " + c );
}
}

```

## Mantıksal Operatörler

true / false döndüren ifadeler veya boolean tipinde değerler tutan değişkenler üzerinde uygulanabilir.

Operator	Description	Example
&& (mantıksal ve)	Eğer A ve B değeri true ise true olur.	(A && B) false
(mantıksal veya)	Eğer A veya B değerinden biri true ise true olur.	(A    B) true
! (mantıksal değil)	() parantez içinde yer alan ifade true ise değerini alıp false yapar. Eğer ifade false ise true yapar.	!(A && B) true

Örnek:

```

public class Test {

    public static void main(String args[]) {

        boolean a = true;
        boolean b = false;

        System.out.println("a && b = " + (a&&b));
        System.out.println("a || b = " + (a||b) );
        System.out.println("!(a && b) = " + !(a && b)); } }

```

## Soru İşareti Operatörü

? işareti operatörü ile Java’da mantıksal kıyaslama yapılabilir. ? ifadesi Java’daki “if-else” yapısı yerine kullanılabilir. Tek satırda bunu yapabilmemizi sağlar.

```
public class Test {  
  
    public static void main(String args[]) {  
        int a, b;  
        a = 10;  
        b = (a == 1) ? 20: 30;  
        System.out.println( "Value of b is : " + b );  
  
        b = (a == 10) ? 20: 30;  
        System.out.println( "Value of b is : " + b );  
    }  
}
```

## “instanceof” Operatörü

Nesne tipinde değişkenler için kullanılabilir. İlkel veri tipleri için bu operatör kullanılamaz. “instanceof” operatörü ile değişkenin barındırdığı veri aradığımız tipe sahip mi değil mi anlamamızı sağlar.

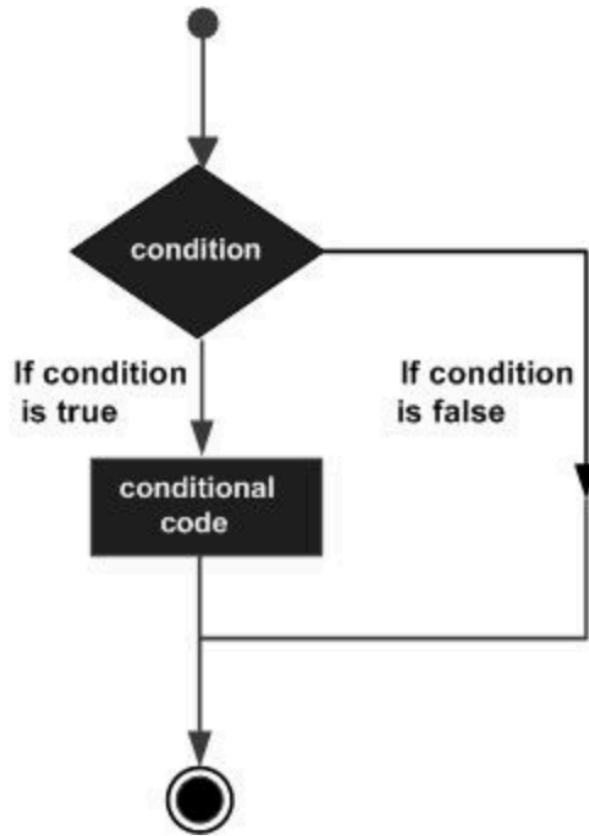
Örnek:

```
public class Test {  
  
    public static void main(String args[]) {  
  
        String name = "James";  
  
        boolean result = name instanceof String;  
        System.out.println( result );  
    }  
}
```

## Karar Mekanizmaları (if-else)

Koşula göre program içinde farklı işlemleri yerine getirmek gerekebilir. Bu tarz durumları Java’da kodlayabilmek için if-else, switch-case gibi yapılar bulunmaktadır. Örneğin: “yaşı 50’den küçük olanların personel kayıtlarını getir” gibi bir ifadede yaşı 50’den küçük olanları tespit etmek için karar mekanizmaları kullanılır. Eğer koşul sağlanmıyorsa başka bir kod bloğu işletilir.





Yukarıdaki ifadenin yazılımsal olarak Java’da karşılığı aşağıdaki gibidir.

```
if (age < 50) {  
    // personel kayıtlarını getir  
}
```

Koşulun gerçekleşip gerçekleşmediği “if” cümlesi içindeki ifadeye bağlıdır. Eğer mantıksal ifade true ise “if” kod bloğu çalıştırılır. Tabi ifadenin false döndüğü durumda da programın bir şeyler yapmasını isteyebiliriz. Bu durumda ise “else” ifadesi kullanılır. Aşağıdaki gibi bir örnekle açıklayabiliriz.

Eğer, faiz oranı %70’den büyükse “Kurumsal Müşteri” tipinde kredi ver, değilse “Standart Müşteri” tipinde kredi ver şeklinde bir karar mekanizması aşağıdaki gibi tasarlanabilir.

```
if (creditRatio > 0.7) {  
    System.out.println("Kurumsal müşteri tipinde kredi");  
}  
else {  
    System.out.println("Standart müşteri tipinde kredi");  
}
```

Koşullar birden fazla olabilir ve hiçbir koşula uymuyorsa en sonunda varsayılan bir duruma girilir ve o kod bloğunu çalıştırmak gerekir. Bu tarz durumlarda ise “if - else if - else” gibi yapılar kullanılır.

Örneğin tuz oranı %80 ve üzerinde ise “yüksek derecede tuzlu”, %80 ile %50 arasında ise “orta derecede tuzlu”, bunların dışında bir durumda ise “düşük derecede tuzlu” şeklinde ekrana bilgiler yazan bir program yazmak istediğimizde if-else if-else yapısını kullanabiliriz.

```
float saltRatio = 0.9f;
if(saltRatio >= 0.8) {
    System.out.println("yüksek derecede tuzlu");
}
else if(0.5 < saltRatio && saltRatio < 0.8 ) {
    System.out.println("orta derecede tuzlu");
}
else {
    System.out.println("düşük derecede tuzlu");
}
```

“if-else” yapılarını iç içe de kullanma şansına sahibiz.

Örneğin: 18 yaşından küçük olanlar kan bağıışı yapamazlar, fakat, 18 yaşına eşit ve büyük olan bir kişi eğer kilosu 48’den büyükse kan verebilir, kilosu 48’den küçükse kan veremez gibi basit bir kuralı Java’da kodlayalım.

```
int age=25;
int weight=48;

if(age>=18){

    if(weight>=48){
        System.out.println("Kan verebilirsiniz");
    }
    else{
        System.out.println("Kan veremezsiniz");
    }

}
else{
    System.out.println("Kan verebilmek için yaşıınız 18'den büyük olmalıdır.");
}
```

## Java’da “switch-case” Yapıları

Programlama yaparken birden fazla koşula sahip durumlarla karşılaşabiliriz. Örneğin: eğer 1’e basarsanız “vize işlemleri”, eğer 2’ye basarsanız “kredi kartı işlemleri”, eğer 3’e basarsanız “ev kredisi işlemleri”, eğer 4’e basarsanız “müşteri temsilcisine bağlanmak”, sıfıra basarsanız “diğer işlemler menüsüne gitmek” gibi çoklu koşullara göre programlama yapmak gerekebilir. Bunu çözmek için “if-else if” yapılarını ya da “switch-case” yapısını kullanırız.

Not: “switch-case” yapısında eğer her case’in sonuna “break” ifadesi koymazsak ise aradığı koşulu bulana kadar tüm case’lere girip o kod bloklarını çalıştıracaktır.

```
Scanner scanner = new Scanner(System.in);
int operationChoice = scanner.nextInt();

System.out.println("0-Diğer işlemler");
System.out.println("1-Vize işlemler");
System.out.println("2-Kredi kartı işlemler");
System.out.println("3-Ev kredisi işlemler");
System.out.println("4-Müşteri temsilcisi işlemler");
System.out.println("Your choice is : " + operationChoice);

switch (operationChoice) {
    case 0:
        System.out.println("Diğer işlemler menüsü");
        break;
    case 1:
        System.out.println("Vize işlemleri");
        break;
    case 2:
        System.out.println("Kredi kartı işlemleri");
        break;
    case 3:
        System.out.println("Ev kredisi işlemleri");
        break;
    case 4:
        System.out.println("Müşteri temsilcisi işlemleri");
        break;
    default:
        System.out.println("Lütfen geçerli bir işlem tipi
seçiniz");
}
```

# Java’da Döngü Mekanizmaları

Döngüler bilgisayara tekrarlı işleri yaptırabilmek için gereklidir. Örneğin, 1000’e kadar olan tüm tek sayıları bulmak tekrarlı bir işlemdir. Müşteri şifresini tekrar tekrar girmeye çalışmakta tekrarlı bir işlemdir.

Döngüler 4 ana bileşenden oluşur:

- Döngünün iterasyon sayısını tutacak değişkene ilk değer atanır.
- Döngünün sonlandırılması veya devam etmesi için bir koşul cümlesi belirtilir.
- Döngünün her iterasyonda ne kadar artıp ne kadar azalacağı belirtilir.

## “for” Döngüsü

Bu döngü tipinde iterasyon sayısı belli bir miktardadır. Döngünün maksimum kaç kez işletileceği önceden bellidir.

```
for( initialization; condition; incr/decr) {  
    //statement or code to be executed  
}  
  
for( int i=-100; i < 1000; i++) {  
    // tek sayıları bul  
    if(i % 2 == -1 || i % 2 == 1) {  
        System.out.println("Tek sayı: " + i);  
    }  
}
```

Döngüler iç içe tanımlanabilir. Bunlara nested loops denir. Örneğin: çarpım tablosunu ekrana yazdırmak için iç içe 2 tane döngü tanımlamak gerekir.

```
for(int i = 1; i <= 9; i++) {  
    for(int j = 1; j <= 9; j++) {  
        int result = i * j;  
        String formattedOutput = String.format("(%d x  
%d)=%d", i, j, result);  
        System.out.println(formattedOutput);  
    }  
}
```

## “while” Döngüsü

“for” döngüsüne benzerdir. “while” döngüsü genellikle işlemin kaç kez tekrar edeceği bilinmediği durumda kullanılır. Örneğin: müşterinin hesabına giriş şifresini kaç kez yanlış gireceğini bilemeyiz. Bu nedenle bu tarz durumlarda “while” döngüsü tercih edilir.

```
Scanner scanner = new Scanner(System.in);
String customerPassword = "12345";
boolean passwordSuccessfull = false;

while(!passwordSuccessfull) {

    System.out.println("Hesap şifrenizi giriniz:");
    String typedPassword = scanner.next();
    if(customerPassword.contentEquals(typedPassword)) {
        passwordSuccessfull = true;
        System.out.println("Sisteme başarılı giriş yaptınız!");
    }

}
```

## “do-while” Döngüsü

“while” döngüsünün çok benzeridir. Bu döngüde koşul en sonda yer aldığı için en az bir kez tur çalıştırılır.

```
int i = 0;
do {
    System.out.println("Sadece bir kez çalıştım :) ");
    i++;
} while(i==1);
```

## “break” ve “continue” Komutu ile Döngüyü Kontrol Etmek

İşletilen bir döngüyü bir koşul sonucu sonlandırmak için “break” komutu kullanılır. “continue” kelimesi ile döngünün devam etmesi sağlanır. Aşağıdaki örnekte döngü içinde ilk 50 değeri continue ile atlayacağız. Dizinin ortasına geldiğimizde ise break komutuyla döngüyü kıracağız.

```
for(int i=1; i <= 100; i++) {

    if(i < 50) {
        System.out.println("Daha yarısına gelmedim!");
        continue;
    }

}
```

```
if(i == 50) {  
    System.out.println("Dizinin ortasındayım!");  
    break; } }
```

## Java Sayı ve Yazı İşlemleri (Numbers & Strings)

Java’da ilkel (Primitive) veri tiplerinden bahsetmiştik. Bu veri tiplerinin bir de referans özellikte olanları da vardır. Bunlar sınıf tabanlı veri tipleri diyoruz. Örneğin: tamsayı olan ve ilkel bir veri tipi olan “int” tipinin bir de “Integer” şeklinde bir sınıf ile temsil edildiği veri tiplerine referans veri tipi diyoruz.

Bunun yanında matematiksel işlemlere yardımcı olmak ve matematiksek fonksiyonları hazır kullanmak için Java’da Math isminde bir hazır tanımlanmış sınıf vardır. Aşağıdaki örnekler kodların içinde gerekli açıklamalar verilmiştir.

Örnek:

```
// Veri dönüşümü  
Long personelId = new Long(100);  
  
int personalAsId = personelId.intValue();  
byte personalAsByte = personelId.byteValue();  
short personalAsShort = personelId.shortValue();  
double personalAsDouble = personelId.doubleValue();  
double personalAsFloat = personelId.floatValue();  
String personalAsText = personelId.toString();  
  
System.out.println(personalAsId);  
System.out.println(personalAsByte);  
System.out.println(personalAsShort);  
System.out.println(personalAsDouble);  
System.out.println(personalAsFloat);  
System.out.println(personalAsText);
```

```
// Kıyaslama metodu ==> compareTo  
Long personelId = new Long(100);  
  
System.out.println(personelId.compareTo(50L));  
System.out.println(personelId.compareTo(100L));  
System.out.println(personelId.compareTo(150L));
```

```
// değerlerin eşitliğini karşılaştırma metodu ==> equals  
Long personelId = new Long(100);  
  
System.out.println(personelId.equals(50L));  
System.out.println(personelId.equals(100L));
```

```
// String değerlerden sayılara dönüşüm metodu ==> valueOf
long number1 = Long.valueOf("100");
int number2 = Integer.valueOf("5");
short number3 = Short.valueOf("1");
```

```
// String değerlerden sayılara dönüşüm metodu ==> parseX
long number1 = Long.parseLong("100");
int number2 = Integer.parseInt("5");
short number3 = Short.parseShort("1");
```

```
// mutlak değer alma fonksiyonu ==> abs
Integer a = -8;
double d = -100;
float f = -90;
```

```
System.out.println(Math.abs(a));
System.out.println(Math.abs(d));
System.out.println(Math.abs(f));
```

```
// ceil metodu ile yukarı yuvarlama, floor ile ise aşağı yönlü yuvarlama yapılır.
```

```
double d = 100.675;
float f = 90.15f;
```

```
System.out.println(Math.ceil(d));
System.out.println(Math.ceil(f));
```

```
System.out.println(Math.floor(d));
System.out.println(Math.floor(f));
```

```
// rint metodu ile ondalıklı kısım 0.5'den büyükse yukarı doğru
// eğer 0.5'e eşit ve küçük ise aşağı doğru yuvarlama yapar. rint
fonksiyonu sonuç olarak sadece int tipinde değer verir.
```

```
double d = 100.675;
double e = 100.500;
double f = 100.200;
```

```
System.out.println(Math.rint(d));
System.out.println(Math.rint(e));
System.out.println(Math.rint(f));
```

```
// rint metodu ile ondalıklı kısım 0.5'e eşit ve büyükse yukarı doğru
// eğer 0.5'den küçük ise aşağı doğru yuvarlama yapar.
// round fonksiyonu long veya int döndürür.
```

```
double d = 100.675;
double e = 100.500;
float f = 100;
float g = 90f;

System.out.println(Math.round(d));
System.out.println(Math.round(e));
System.out.println(Math.round(f));
System.out.println(Math.round(g));

// max verilen iki değerden en büyüğünü döndürür.
// min verilen iki değerden en küçüğünü döndürür.
System.out.println(Math.min(12.123, 12.456));
System.out.println(Math.max(12.123, 12.456));

// e tabanında log alma fonksiyonudur.
double x = 2.76;
System.out.printf("log(%.3f) is %.3f\n", x, Math.log(x));

// üs alma fonksiyonudur. 2 üzeri 3 gibi.
double x = 2;
double y = 3;
System.out.printf("pow(%.f, %.f) is %.f", x, y, Math.pow(x, y));

// karekök alma fonksiyonudur.
double x = 4;
System.out.printf("sqrt(%.3f) is %.3f\n", x, Math.sqrt(x));

// 0-1 arasında rastgele sayı üretme fonksiyonudur.
System.out.println(Math.random());
```

## Java’da String İşlemleri

Java’da varsayılan dil içerisinde gelen String işleme kütüphaneleri mevcuttur. String sınıfı içinde yer alan statik fonksiyonlarda yazılımcılara yardımcı olmaktadır.

String veri tipi Java’da ilkel (primitive) bir veri tipi değildir. Bu nedenle “new” anahtar sözcüğü kullanılarak nesne şeklinde oluşturulabilir.

Java’da String veri tipinde bir değişken tanımı aşağıdaki gibi yapılabilir. Buna Literal tanımlama diyoruz. Sıklıkla kullanılan bir tanımlama biçimidir. Değişkene direkt olarak veriyi “=” operatörüyle atama yapıyoruz.



```
String greeting = "Hello world!";
```

Literal tanımlama dışında “new” anahtar sözcüğüyle bir nesne olarak oluşturabilirsiniz. Bu yöntemle Heap hafıza bölgesinde bir alan kaplamış olursunuz. Literal tanımlama ile Heap hafızadan kazanç sağlanır.

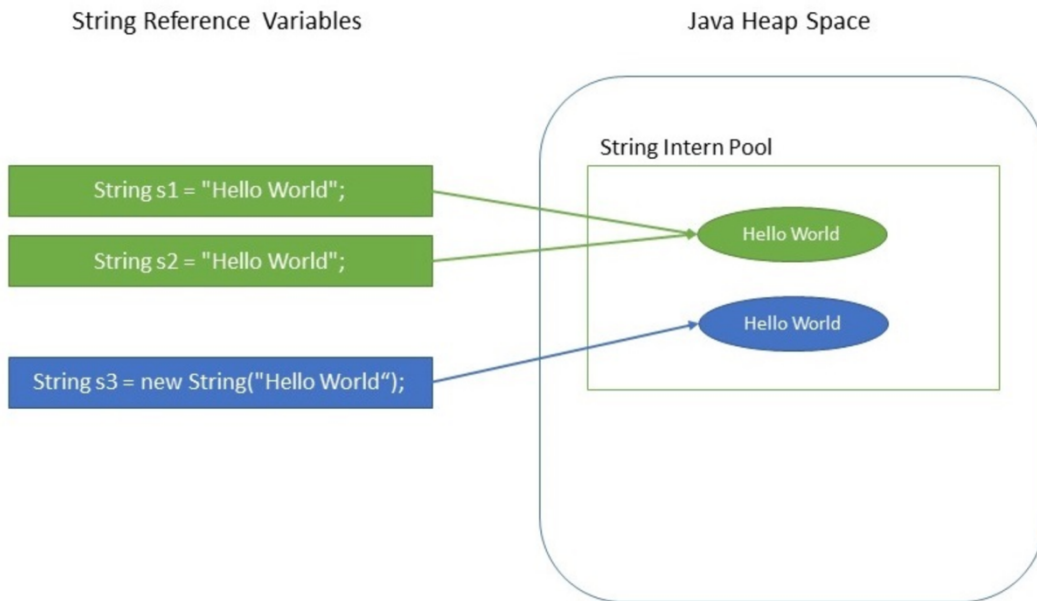
Örnek:

```
String helloString = new String("Merhaba Dünya!");
```

String tipinde tanımlanan değişkenler veya nesneler değiştirilemez duruma sahiptir. Buna Immutable Object denir. String içindeki değeri değiştirmek isterseniz. Yeni bir değişken veya nesne tanımlama yapmak gerekecektir.

String veri tipi niçin Immurable (Değiştirilemez) tasarlandı diye düşünebilirsiniz. Bunun birden fazla sebebi vardır. Öncelikle, yazılım geliştirilirken String veri tipi çok sık kullanılır. String tipinde sürekli nesne üretilmesi Heap hafıza bölgesini çok kötü kullanmaya ve performans problemlerine yol açar. Bu nedenle Literal tanımlarla ve özel String Pool (Havuz) ile birlikte performans kazancı hedeflenmiştir. String’ler için Java’da özel bir hafıza bölgesi vardır. Bu bölgede bir String havuzu vardır. Gerçekten bu havuz içinde belli miktarda tanımlı String nesneleri bulunur. Bu nesneler Heap hafızada belli bir bölgeyi kaplarlar. Fakat, sınırlı sayıdadırlar. Aşağıda, Literal tanımlama ile “new” ile String oluşturma arasındaki fark gösterilmektedir. İki tane Literal tanımlı s1 ve s2 isminde değişkenimiz olduğunu düşünelim.

```
String s1 = "Hello World";  
String s2 = "Hello World";  
  
s1 == s2
```



Yukarıdaki örnekte bizler iki tane s1 ve s2 tipinde iki tane Literal tanımlı değişkenler oluşturduk. Aslında, bu iki değişken aynı değere sahip oldukları için String Pool'dan (Havuzdan) önceden oluşturulmuş bir nesneyi işaret ederek iki tane ayrı hafıza bölgesini doldurmaktan kurtuldular ve hafıza kazancı sağladılar. Yukarıdaki şekilde de bu durum izah edilmektedir. S3 değişkeni s1 ve s2 ile aynı değere sahip olsa da “new” ile yeni bir nesne olarak oluşturulduğu için hafızada başka bir bölgede oluşturulur.

Örnek:

```
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Merhaba";
String s4 = new String("Merhaba");

System.out.println("s1 == s2 : " + (s1 == s2));
System.out.println("s2 == s3 : " + (s2 == s3));
System.out.println("s3 == s4 : " + (s3 == s4));
```

Yukarıdaki örnekte de bu durum geliştirilmiştir. s1 ve s2 değişkenleri aynı değere sahip Literal tanımlı String değişkenleri oldukları için aynı hafıza bölgesini işaret ederler. s3 değişkeni Literal tanımlı olmasına rağmen başka bir değere sahip olduğu için String Pool'dan (Havuzdan) başka bir nesneyi alıp onu işaret etmektedir. s4 değişkeni ise “new” ile oluşturulduğu için Heap hafızadan bambaşka bir alanı almıştır. Yukarıdaki programın çıktısı aşağıdaki gibidir. “==” operatörü String’lerde hafıza adresi kıyaslaması yapar. s1 ile s2 aynı hafıza adresini gösterir. s2 ve s3 Literal tanımlı olsalar da havuzdaki farklı nesneleri işaret ettikleri için adresleri eşit değildir. s3 ve s4 aynı değerlere sahip olsa da biri Literal tanımlıdır ve havuzdaki bir nesneyi işaret eder. Diğeri ise “new” ile tanımlandığı için Heap hafızada başka bir adresi işaret eder.

```
s1 == s2 : true
s2 == s3 : false
s3 == s4 : false
```

String’ler değiştirilemez olduğu için Güvenlik ile ilgili konularda da varsayılan olan korumacı bir özelliğe sahiptir. Ayrıca, String değişkenler Immutable (Değiştirilemez) olduklarından dolayı Çok Kanallı (MultiThread) programlamada Thread-Safe özelliğe sahiptirler. String Pool, String Literal ve String new arasındaki fark size mülakatlarda sorulabilir.

Not: String veri tiplerinde verinin karakter uzunluğunu bulmak için “length()” metodundan faydalanılır. Örnek:

```
String s1 = "Hello";
int lengthOfs1 = s1.length();
```

Not: String ifadeleri birbiriyle birleştirmek için “+” operatörü veya “concat” metodu kullanılır. Örnek:

```
String namePrefix = "My name is ";
String greetingMessage = namePrefix.concat("Zara");
```

# Formatlı String İfadeler Oluşturmak

Java’da “String.format” metoduyla formatlı veriler oluşturabilirsiniz. “format” metodu String sınıfının static fonksiyonudur. Nesne üretmeksizin direkt sınıf ismiyle çağırabilirsiniz.

Örnek:

```
int speed = 50;
String departureCityName = "Akhisar";
String arrivalCityName = "İstanbul";
String fullText = String.format("Aracın ortalama hızı: %d kilometredir." +
                                "Araç %s şehrinde kalkıp %s şehrine varacaktır.",
                                speed, departureCityName, arrivalCityName);
System.out.println(fullText);
```

Yukarıdaki örnekte bir metin oluşturulmaya çalışılıyor. Metin içerisinde tanım sayıları ifade eden %d ve String veri tipini ifade eden %s alanları vardır. Bu alanlar dinamiktir. Gelen değerleri cümle içinde gösterilmesini sağlarlar.

```
String s = "Strings are immutable";

// s isimli String değişkenindeki ifadenin 8. indeksindeki karakteri alır.
// Burada dikkat edilmesi gereken şey indeksler sıfırdan başlar. 0 yüzden
// 9. karakteri okuyoruz.
char result = s.charAt(8);

-----

String str1 = "Strings are immutable";
String str2 = "Strings ARE immutable";

// s1 ve s2 değişkenleri içinde yer alan değerleri büyük küçük harf
// duyarlılığı olmaksızın kıyaslar.
int result = str1.compareToIgnoreCase(str2);
System.out.println(result);

-----

String str = new String("This is really not immutable!!");

// String ifadenin sonu verilen ifadeyle bitiyor mu kontrol eder. true veya
// false döner.
// Cümlelerin sonu "immutable!!" ile bitiyor mu kontrol ediyoruz. true döner.
boolean retVal = str.endsWith( "immutable!!" );
System.out.println("Returned Value = " + retVal );

-----
```

```
String str1 = new String("This is really not immutable!!");
String str2 = new String("This is really not immutable!!");
// equals metodu iki String değişkenin aynı değere sahip olup olmadığını
kıyaslar.
// == operatörü ile iki String'leri kıyaslasaydık, hafıza adreslerini
kıyaslamış olacaktır. O da false dönecekti.
boolean retVal = str1.equals( str2 );
System.out.println("Returned Value = " + retVal );
```

```
-----

String str = new String("Welcome to kodluyoruz.org");
String subStr1 = new String("Tutorials");
// indexOf metodu verilen ifadenin cümlede nerede hangi indeksten itibaren
başladığını belirtir.
// Eğer ifadeyi cümle içinde bulamazsa -1 döner.
System.out.println("Found Index :" + str.indexOf( subStr1 ));
```

```
-----

String str = new String("Welcome to kodluyoruz.com");
// replace metoduyla bir cümle içindeki istediğimiz ifadeyi bir başka ifade
ile değiştirebiliriz.
// Örneğin: Welcome ifadesini Merhaba ile değiştiriyoruz. Değişiklik
sonucunda değişmiş halini yeni bir String olarak döner
str = str.replace("Welcome", "Merhaba");
str = str.replace("to", "");
System.out.println(str);
```

```
-----

String str = new String("Welcome-to-kodluyoruz.org");
// split fonksiyonu cümleyi ayırmak için bir karakter alır. Sonra o
karaktere göre cümleyi parçalara böler.
// bu örnekte - işaretiyle ayırma işlemi uygulanmıştır.
String[] items = str.split("-");
System.out.println("Return Value :" );
```

```
-----

String str = new String("Welcome to kodluyoruz.com");
// startsWith metoduyla cümle belirtilen ifadeyle başlıyor mu diye kontrol
edilebilir.
System.out.println(str.startsWith("Welcome") );
```

```
-----
```

```
String str = new String("Welcome to kodluyoruz.com");  
// substring fonksiyonu verilen başlangıç indeksinden itibaren verilen  
// bitiş indeksine kadar olan bölümü kırpıp ve yeni bir string olarak  
// döndürür.  
String subStringPart = str.substring(10, 15);
```

## StringBuilder Sınıfının Kullanımı

Java’da performanslı String birleştirme işlemleri için “StringBuilder” sınıfı kullanılabilir. “+” operatörü ve “concat” fonksiyonuna göre daha performanslı bir yöntemdir.

Örnek:

```
StringBuilder builder = new StringBuilder();  
builder.append("İlk cümle");  
builder.append("İkinci cümle");  
builder.append("Üçüncü cümle");  
  
System.out.println(builder.toString());
```

“append” metoduyla String ifadeler eklenir. Ardından, “toString” metoduyla birleştirilmiş tüm String ifade alınır.

## Java’da Dizi (Array)

Dizi (Array) kavramı programlama dillerinde bir veri tipini ifade eder. Bu veri tipi liste halindeki ardışık verileri bir arada tutan yapıya denilir. Bu ardışık yapıya ait elemanlara indeks yoluyla erişim sağlanabilir. Diziler sabit boyutludur. Örneğin: 10 elemanlık dizi. Dizilerde aynı tipten veri tutulur. Örneğin: tüm elemanları “int” olan bir dizi.

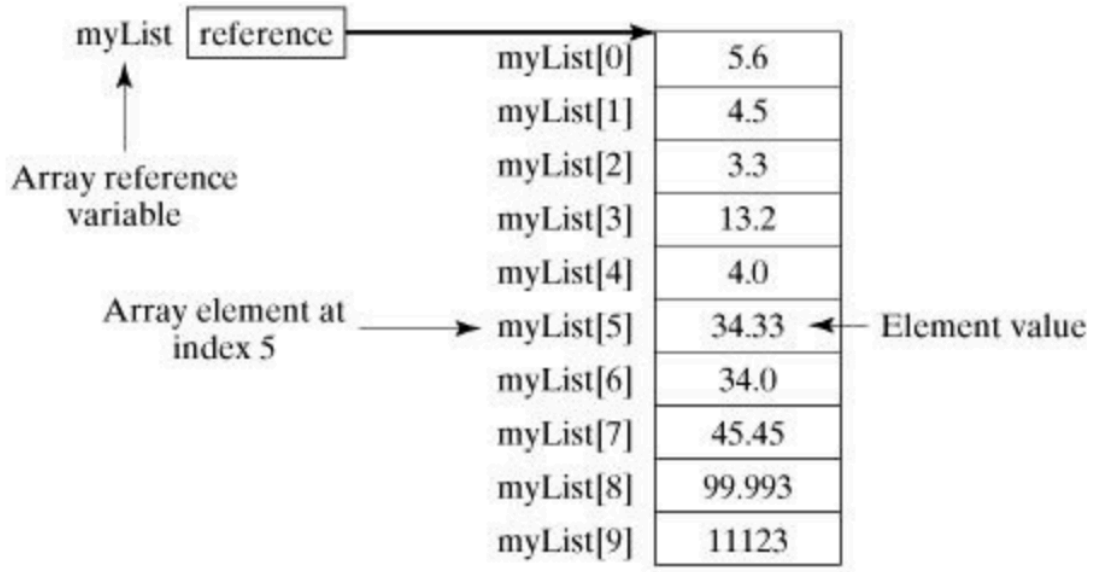
Dizi’nin hafızada bir başlangıç adresi olur ve ardışık olan diğer elemanlar sırayla hafızaya yerleştirilir. Dizi’ler “new” anahtar sözcüğüyle oluşturulur. Böylece, Heap Hafıza bölgesinde yer kaplarlar.

```
double[] myList;    // tercih edilen yol  
  
veya  
  
double myList[];    // başka türlü tanımlama biçimi
```

Diziler veri tipi ve [] parantezler ile belirtilir. Yukarıda iki farklı tanımlama görülmektedir. Hafızadan yer alıp diziye alan ayırabilmek için “new” anahtar kelimesi kullanılır.

```
double[] myList = new double[10];
```

Yukarıda maksimum 10 eleman alabilen “double” veri tipinde olan bir dizi oluşturulmuştur.



Yukarıda “myList” isminde bir dizi tanımlamıştık. Tanımlanan dizi hafıza yukarıdaki şekildeki gibi tutulur. “myList” değişkeni dizinin başlangıç adresini tutar. Dizilerde ardışık bir sıra olduğu için ilk elemandan sonra gelen elemanların hafıza adresleri de birer birer artar. Dizi blok halinde yer kaplar. Diziye erişmek için indeks numarası kullanılır. Örneğin: “myList[0]” demek dizinin 1. Elemanını verecektir. Java’da dizilerin indeksleri sıfırdan başlar. Örneğin: “myList[5]”, yani 5 nolu indeksteki dizi elemanını ver dediğimizde aslında dizinin 6. Elemanına erişmiş oluruz.

Örnekler:

```
// Java'da diziye ilk değerler süslü parantezler arasında verilir.
double[] myList = { 1.9, 2.9, 3.4, 3.5 };

// tüm dizi elemanlarını arada boşuk bırakarak sırayla ekrana yazdırır.
for (int i = 0; i < myList.length; i++)
{
    System.out.println(myList[i] + " ");
}
```

Yukarıdaki örnekte myList isimli diziye ilk değerleri hemen verilmiştir. Süslü parantezler arasında kalan değerler dizi elemanlarıdır.

Ardından, bir “for” döngüsü ile dizi elemanları ekrana yazdırılır.

## Dizileri fonksiyonlara parametre olarak göndermek

Tanımladığınız dizileri fonksiyonlara parametre olarak gönderebilirsiniz.

Örneğin elimizde static bir fonksiyon olsun. Bu fonksiyona int tipinde verileri olan bir diziyi girdi (input) olarak vermek istersek aşağıdaki gibi olur.

```
public static void printArray(int[] array)
{
    for (int i = 0; i < array.length; i++)
    {
        System.out.print(array[i] + " ");
    }
}
```

printArray ( **int[] array** ) kırmızı olarak işaretlediğimiz yer diziyi yerel değişken olarak fonksiyona gönderdiğimiz noktadır. Java’da tüm değişkenler “Pass by Value” yöntemiyle geçerilir. Bu şu demektir. Sizin tanımladığınız değişkenin, nesnenin veya dizinin birebir kopyası oluşturulur. Bu kopya değer fonksiyona yerel değişken olarak gider. Bu Java mülakatlarında size sorulabilecek bir detaydır.

## Dizileri fonksiyonlardan geri döndürmek

Fonksiyonlar belli bir işi yapıp sonucunda değer dönebilen veya dönmeyen kod bloklarıdır. Fonksiyonlar için altın kural, her fonksiyonun tek bir işi olmalıdır. Örneğin: dizi ortalaması alma işi yapan bir fonksiyon dizileri ekrana yazdırma işini yapmamalıdır. Veya dizilerin ortalamasını alma işi ile dizileri toplama işlemi aynı fonksiyon içinde olmamalıdır. Her biri ayrı fonksiyonlar olmalıdır.

```
public static int[] reverse(int[] list)
{
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--)
    {
        result[j] = list[i];
    }

    return result;
}
```

Yukarıda dizinin tersine çevrilmiş halini döndüren bir fonksiyon vardır. public static **int[]** reverse(...) kırmızı ile işaretlenen alan dizi döndüreceğimizi ve bu dizinin veri tipini söylüyoruz. Burada veri tipimiz “int” 😊