Java Sanal Makinesi (JVM) Nedir?

**JVM**(Java Sanal Makinesi) bir soyut makinedir. Java Sanal Makinesi (JVM), Java bytecode’nu makine diline çevirir. JVM, Java Çalışma Ortamının (JRE) bir parçasıdır. Başka programlama dillerinde, derleyici, belirli bir sistem için makine kodu üretir. Ancak, Java derleyicisi,**Java Sanal Makinesi** olarak bilinen bir **Sanal Makine** için kod üretir.

Java Sanal Makineleri pek çok donanım ve yazılım platformu için uygundur.

### **Java Sanal Makinesi Nasıl Çalışır?**

İlk olarak Java kodunu bytecode’a derler. Bu bayt kodu (bytecode) farklı makineler üzerinde yorumlanır. Ana makine ve Java kaynağı arasında, Bytecode aracı bir dildir.

Java Sanal Makinesi (JVM), hafıza alanı (memory space) bölüştürmek için sorumludur.

# JVM (Java Sanal Makinesi) Mimarisi

[*Harun Çetin*](https://blog.haruncetin.com.tr/author/Harun%20%C3%87etin/)

*[Ağustos 26, 2021](https://blog.haruncetin.com.tr/2021/08/)*

*[Yorum Bırak](https://blog.haruncetin.com.tr/2021/08/27/jvm-java-sanal-makinesi-mimarisi/" \l "respond)*

JVM (Java Virtual Machine), Java uygulamalarını çalıştırmak için çalışma zamanı motoru (runtime engine) görevi yapan soyut bir bilgi işlem makinesidir. Java bayt kodunun yürütülebileceği çalışma zamanı ortamını sağlar. JVM için üç kavram vardır:

* **Specification (Tanımlama):**JVM bir tanımlamadır. Bu tanımlama içerisinde Java Sanal Makinesi’nin çalışma mekanizması tarif edilir. Ancak bu tarifin uygulamasını yapacak üretici, algoritma seçimi konusunda serbesttir.
* **Implementation (Uygulama**): JVM’in uygulaması JRE (Java Runtime Environment) olarak bilinir. Günümüzde en büyük uygulama sağlayıcısı Java’nın da sahibi olan Oracle’dır (Hotspot VM) ancak diğer şirketler tarafından da bağımsız JVM uygulamaları geliştirilmektedir (örn. GraalVM, Zing JVM, Eclipse OpenJ9, vs).
* **Instance (Örnek):**Bir Java sınıfını çalıştırmak için konsola “java” komutu yazılıp entera basıldığında bellekte JVM’in bir örneği oluşturulur.

Genelde Java geliştiricileri, geliştirdikleri yazılımın bayt kodunun JRE (Java Runtime Environment) tarafından yürütüleceğini bilir, ancak birçoğu, JRE’nin bayt kodunu analiz eden, kodu yorumlayan ve yürüten Java Sanal Makinesi’nin (JVM) bir uygulaması olduğunu bilmez. Aslında bir Java geliştiricisi için, JVM mimarisini bilmek daha verimli kod yazmayı sağlaması açısından çok önemlidir.

Bu yazıda, Java’nın JVM mimarisini ve bileşenlerini detaylı olarak ele alacağız.

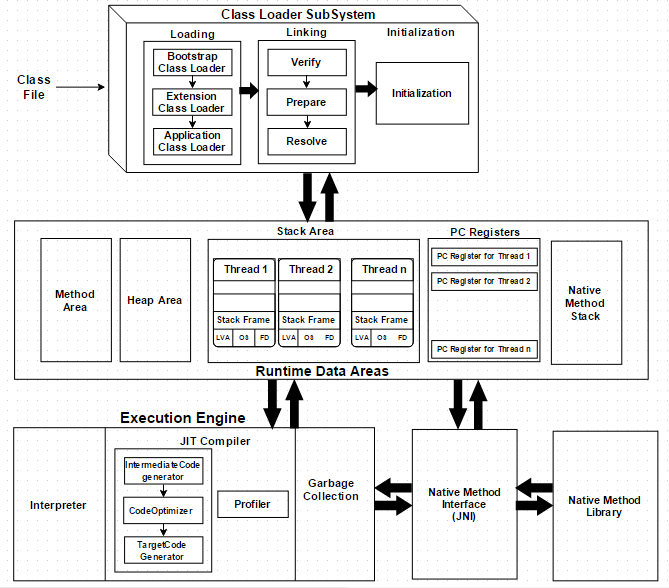
## **JVM Nasıl Çalışır?**

Sanal Makine, fiziksel bir makinenin yazılımsal uygulaması olarak tanımlanabilir. Java programları da böyle bir sanal makine üzerinde çalışır. JVM, WORA (Write Once Run Anywhere) mantığıyla geliştirilmiştir. Bunun anlamı, bir programcı bir sistemde geliştirdiği Java kodunu, herhangi bir ayarlama yapmadan başka herhangi bir JRE yüklü sistemde çalıştırabilir. Bu JVM sayesinde mümkün olmaktadır. Bir Java kaynak dosyası, Java derleyicisi (javac) tarafından derlendiğinde, bu kaynak dosyasından .class uzantılı çalıştırılabilir bayt kodu (bytecode) dosyası oluşturulur, ardından bu dosya, JVM’ne girdi olarak verilerek belleğe yüklenmesi ve çalıştırılması sağlanır.

JVM Mimari Şeması (kaynak: dzone.com)

Yukarıdaki şemada gösterildiği üzere JVM üç ana alt sisteme bölünmüştür:

1. ClassLoader Alt Sistemi
2. Runtime Data Areas (Çalışma Zamanı Veri Alanları)
3. Execution Engine (Yürütme Motoru)



# Java 8 ve JVM Dilleri

Java Sanal Makinesi (JVM), Java 7 ile başlayan [Da Vinci Machine](http://openjdk.java.net/projects/mlvm/) projesiyle, özellikle dinamik tipli dilleri JVM üzerinde çalışabilir kılmaktadır.

[JVM Dilleri Tablosu](https://kodedu.com/2014/10/java-8-ve-jvm-dilleri/#jvm-dilleri-tablosu)nda listelenen programlama dilleri JVM bünyesinde koşturulabilmektedir. Bazı diller yorumlama usülüyle koşturulurken, bazıları ise bayt koda dönüştürüldükten sonra koşturulmaktadır. Fakat, JavaScript haricindeki dillere karşılık bir uygulayıcı kütüphaneyi projenize eklemeniz gerekmektedir.

Örneğin JVM üzerinde Ruby dili ile uygulama geliştirmek istiyoranız, JRuby bağımlılığını Java projenize eklemelisiniz.

JRuby Maven Dependency

<dependency>

<groupId>org.jruby</groupId>

<artifactId>jruby</artifactId>

<version>1.7.16</version>

</dependency>

Diğer listeli diller için de benzer biçimde gereken bağımlılık Java projenize eklenmelidir.

Fakat, JavaScript programlama dili için olay biraz farklı bir durumda. Çünkü, Java 7 Rhino, Java 8 ise Nashorn isimli JavaScript motorlarını gömülü olarak JVM içerisinde bulundurmaktadır. Bu Java ekosisteminin JavaScript diline ne kadar önem verdiğini ayrıca göstermektedir.

### Java Scripting API

Java programlama dili, tüm bu listeli dilleri koşturabilmek için ortak arayüzlerin bulunduğu bir API sunmaktadır. Java Scripting API bileşenleri [javax.script](http://docs.oracle.com/javase/8/docs/api/javax/script/package-summary.html) paketi içerisinde bulunmaktadır.

javax.script paketi oldukça basit arayüz ve sınıflar içermektedir.Bunlardan en önemlisi ScriptEngine arayüzüdür.

### ScriptEngine

ScriptEngine türünden nesneler, ScriptEngineFactory#getEngine metodu üzerinden eşsiz bir takma isim ile elde edilmektedir. Bu nesneler ile, String türünden kod blokları koşturulabilmekte, ayrıca Java ile iletişim kurulabilmektedir. Örneğin, Nashorn JavaScript motoru için "nashorn" veya "rhino" takma adları, Ruby için ise "jruby" takma adı kullanılmaktadır.

Örneğin;

...

ScriptEngine engine = ScriptEngineFactory.getEngine("nashorn"); **(1)**

ScriptEngine engine = ScriptEngineFactory.getEngine("rhino"); **(2)**

ScriptEngine engine = ScriptEngineFactory.getEngine("jruby"); **(3)**

ScriptEngine engine = ScriptEngineFactory.getEngine("jython"); **(4)**

...

1. Java 8 için JavaScript motoru
2. Java 7 için JavaScript motoru
3. Ruby için JRuby motoru
4. Python için Jython motoru

### Nashorn JavaScript Motoru

Nashorn, Java 8 için özel olarak sıfırdan geliştirilen bir JavaScript motorudur. Nashorn, Rhino JavaScript motoruna göre 5 kat daha fazla performans sunmaktadır.

Nashorn JavaScript motoru EcmaScript 5 standardını desteklemekte ve tüm testlerini başarıyla geçmiş bulunmaktadır.

JVM dillerinden Java Scripting API destekleyenler, ScriptEngine#eval metodu ile kod bloklarını koşturma imkanı elde etmektedir. Bu sayede ortak arayüz bileşenleri üzerinden Java harici diller JVM üzerinde koşturulabilmektedir.

Nashorn Engine Örneği

ScriptEngine engine = ScriptEngineFactory.getEngine("nashorn"); **(1)**

engine.eval("function topla(a,b){ return a + b; }"); **(2)**

String sonuc=(String)engine.eval("topla(3,5)"); **(3)**

System.out.println(sonuc); // 8

1. Nashorn Engine elde ediliyor.
2. topla isimli JavaScript fonksiyonu tanımlanıyor.
3. topla fonksiyonu Nashorn ile koşturuluyor, ve sonucu elde ediliyor.

Siz de Java Scripting API destekleyen diğer dilleri JVM ekosisteminde koşturabilirsiniz.