volkanulutas@gmail.com

Java

Java Temelleri

__Java Temelleri Ders Notu

---

1. Java Tarihçesi

1.1. Gizli Bir Proje: "The Green Project" (1991)

Java'nın doğuşu, sanılanın aksine internet için değil, **tüketici elektroniği** (akıllı TV'ler, set-top box'lar) için başladı.

- **Sun Microsystems:** Proje burada, **James Gosling** ve ekibi (Green Team) tarafından başlatıldı.
- **James Gosling:** Java'nın babası olarak bilinir. Hedefi; donanımdan bağımsız, güvenli ve esnek bir dil yaratmaktı.
- **Oak İsmi:** İlk başlarda dile, Gosling'in ofis penceresinin tam karşısındaki meşe ağacından esinlenerek **"Oak"** (Meşe) adı verildi. Ancak bu ismin telif hakları başka bir şirkete ait olduğu için değiştirilmesi gerekti.

Ek Araştırma: “Java The Green Project” diye YouTube’de izleme yapılabilir.

The Evolution of the Java Language: From Green Project to Global Power -> *https://www.youtube.com/watch?v=HsnsiuqUn24*

1.2. Java İsminin Doğuşu (1995)

Ekip bir kafede isim ararken, Endonezya'daki Java adasından gelen kahveyi çok tükettikleri için dilin adını **"Java"** koymaya karar verdiler. Java logosundaki kahve fincanı da buradan gelir.

- **İnternetin Yükselişi:** 1995'te internet patlaması yaşanınca, Java'nın "bir kez yaz her yerde çalıştır" yapısının web tarayıcıları için mükemmel olduğu fark edildi.
- **Netscape Desteği:** O dönemin en yaygın kullanılan web tarayıcısı Netscape'in (günümüzün Chrome’unu düşünebilriiz) Java'yı desteklemesiyle Java bir anda dünya çapında popüler oldu.

1.3. Oracle Dönemi (2010)

Yazılım dünyasının en büyük satın almalarından biri gerçekleşti: **Oracle**, Sun Microsystems'i 7.4 milyar dolara satın aldı. Bu durum Java'nın daha kurumsal bir rotaya girmesini ve ticari desteğinin artmasını sağladı.

1.4. Modern Çağ ve LTS Sürümleri

Java eskiden çok nadir güncellenirdi. Artık her **6 ayda bir** yeni sürüm çıkıyor. Ancak şirketler her sürümü kullanmak yerine **LTS (Long Term Support)** yani "Uzun Süreli Destek" sürümlerini tercih ederler.

Sürüm	Neden Önemli?
Java 8	Devrimsel bir sürümdür. Lambda ifadeleri geldi. Hala sektörde en çok kullanılan sürümdür.
Java 11	Oracle'ın yeni lisanslama ve modüler yapıya geçtiği kararlı sürüm.
Java 17	Modern özelliklerin (Record, Sealed Classes) standartlaştığı performans canavarı.
Java 21	En güncel LTS. **Virtual Threads** gibi devasa performans yenilikleri içeriyor.

Eğlenceli Bilgi: Java'nın maskotu, Gosling'in ekibinden Joe Palrang tarafından tasarlanan **Duke** adındaki kırmızı burunlu, üçgen gövdeli simgedir.

---

2. Neden Java?

Platform Bağımsız (Platform Independent): "Write Once, Run Anywhere" (Bir kere yaz, her yerde çalıştır) felsefesi. En büyük "satış noktası" budur. Diğer dillerde yazdığın kod genellikle derlendiği işletim sistemine mahkumdur. Java'da ise yazdığın kod her yerde çalışır.

Popülerlik ve Kullanım Alanları: Android uygulamalarından kurumsal web sistemlerine, büyük veri (Big Data) çözümlerinden gömülü sistemlere kadar Java'nın gücü.

- Hadoop, Spark, Kafka gibi devasa sistemlerin kalbi Java'dır.

Nesne Yönelimli (Object-Oriented): Java'da her şey bir "nesne" etrafında döner. Bu, gerçek dünyadaki karmaşık sistemleri (örneğin bir banka sistemi veya bir oyun) kod dünyasına modellerken işleri kolaylaştırır. Kodun tekrar kullanılabilirliğini ve sürdürülebilirliğini sağlar.

Güvenli (Secure): Java, "sandbox" (kum havuzu) denilen güvenli bir alanda çalışır. Bellek yönetimi (C++'daki gibi pointer'lar yoktur) Java tarafından otomatik yapılır, bu da sistemin çökmesini veya dışarıdan müdahaleleri zorlaştırır.

Performanslı: Java, saf yorumlanan (interpreted) dillerden (Python gibi) daha hızlıdır. **JIT (Just-In-Time)** derleyici sayesinde çalışma anında kodu optimize ederek makine diline çok yakın bir hızda çalışır. Ek olarak, **Garbage Collector** Java için performans ve güvenlik konusunda muhteşem bir icattır.

Eğlenceli Bilgi: "Java 30 yaşında olmasına rağmen, her 6 ayda bir kendini yenileyerek hala dünyanın en modern ve en çok aranan dillerinden biri kalmayı başarıyor."

Ek Araştırma: Garbage Collector konusunda YouTube'da minik videolar izlenebilir.

How Garbage Collection Works -> https://www.youtube.com/watch?v=Z7egyIzL89Y

---

3. Java Nasıl Çalışır? Arkasındaki mantık nedir?

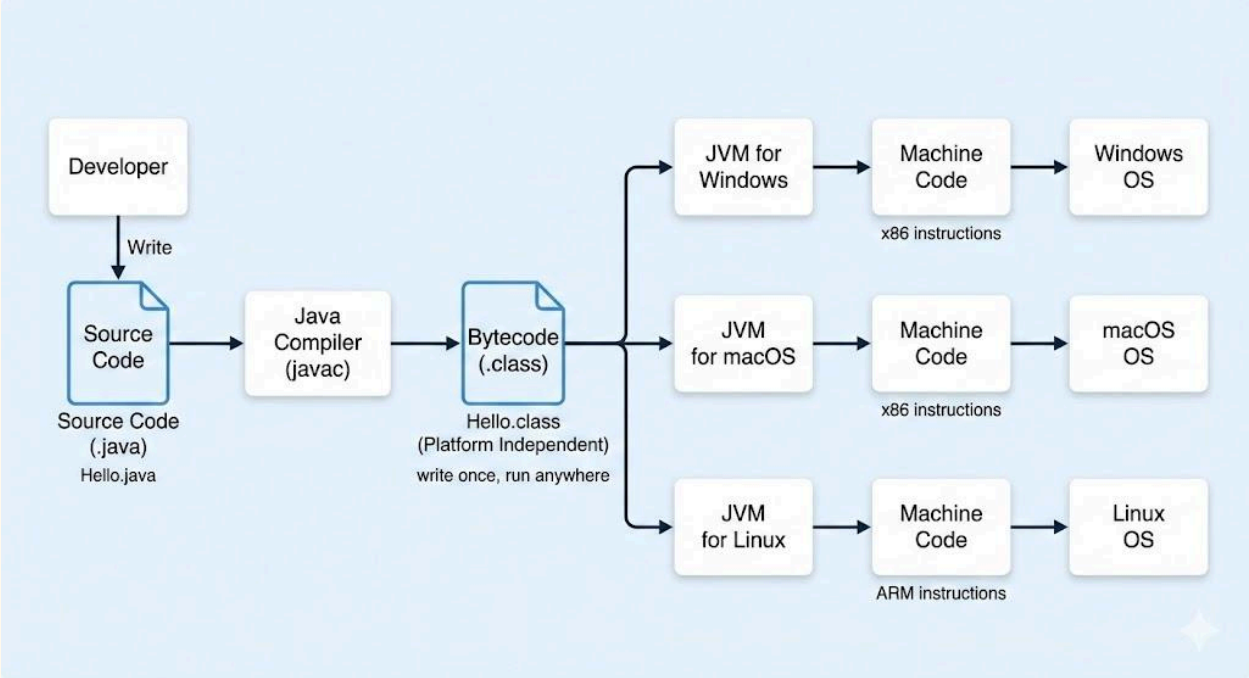
JDK, JRE ve JVM arasındaki farkı netleştirmemiz gerekiyor.

- **JVM (Java Virtual Machine):** Kodun işletim sisteminden bağımsız çalışmasını sağlayan "sanal makine".
- **JRE (Java Runtime Environment):** Programı çalıştırmak için gereken kütüphaneler.
- **JDK (Java Development Kit):** Bizim (yazılımcıların) kod yazmak için ihtiyaç duyduğumuz araç seti.

Ek Araştırma: JRE vs. JDK -> https://www.youtube.com/watch?v=fDXMo3lX-Ug

---

3.1. Java'nın Sihri



Yukarıdaki akış Java'nın en büyük sihridir. Bir "çeviri" süreci gibi düşünebiliriz . Java'nın platform bağımsız olmasını sağlayan şey, kodun direkt işlemciye değil, aradaki bir sanal makineye (JVM) hitap etmesidir. Bu sanal makine bir şekilde ihtiyaçlar doğrultusunda kullanıcıların bilgisayarlara kendi istekleri veya kurulan başka uygulamalar tarafından yüklenir. Şu an dünyada JVM çalıştıran bilgisayar veya cihaz sayısı 3 milyar olarak tahmin edilmektedir. Bu sayının içerisindeki cihazlar; sunucular ve bulut, mobil cihazlar, akıllı kartlar (sim kartlar, pasaport çipleri gibi.), masaüstü bilgisayarlar ve IOT cihazlar.

3.2. Adım Adım Akış

1. **Java Kodu (`.java`):** Bizim yazdığımız, insanın okuyabildiği kaynak kodudur. (Örn: `Hello.java`)
2. **Derleme (Compiler - `javac`):** Java derleyicisi bu kodu alır ve **Bytecode**'a dönüştürür.
3. **Bytecode (`.class`):** Bu dosya artık işlemcinin değil, sadece **JVM**'in anlayabileceği bir dildir. Bu dosya evrenseldir; Windows'ta da aynıdır, Linux'ta da.
4. **JVM (Java Virtual Machine):** Her işletim sistemi için özel bir JVM vardır. JVM, bu Bytecode'u o an üzerinde çalıştığı işletim sisteminin (Windows, MacOS, Android) diline anlık olarak çevirir.

> **Önemli Not:** "Java platform bağımsızdır ama JVM platforma özeldir." Yani Windows için ayrı bir JVM, Mac için ayrı bir JVM indirirsin; ama aynı `.class` dosyasını ikisinde de çalıştırabilirsin.

Eğlenceli Bir Örnek:

Java'yı **Esperanto** diline (yapay evrensel dil) benzetebiliriz.

- Bir kitap yazıyorsunuz (Java Kodu).
- Bunu evrensel bir şifreye çeviriyorsunuz (Bytecode).
- Her ülkenin (İşletim Sistemi) girişinde bir tercüman (JVM) bekliyor.
- Tercüman, bu evrensel şifreyi alıp o ülkenin yerel halkına kendi dillerinde anlatıyor.

Ek Araştırma: Why java is platform independent ? -> https://www.youtube.com/watch?v=iA0dizdqta8

Java'nın bu yapısı sayesinde bugün bankacılık sistemlerinden Mars'taki robotlara kadar her yerde Java kullanılıyor.

4. Maven

4.1. Maven Öncesi "Karanlık Çağ" (The Jar Hell)

Maven yokken bir Java projesi geliştirmek, kod yazmaktan çok "lojistik" yönetmek gibiydi. İşte o dönemdeki temel sorunlar:

- **Manuel Kütüphane Yönetimi:** Bir projede JSON işlemek istiyorsan, internete girer, ilgili .jar dosyasını bulur, indirir ve projenin içine (genelde lib klasörüne) elle kopyalardın.
- **Bağımlılık Zinciri (Transitive Dependencies):** En büyük kabus buydu. Diyelim ki A kütüphanesini indirdin. Ama A kütüphanesi çalışmak için B ve C'ye ihtiyaç duyuyor. B ise D'ye... Hangi kütüphanenin neye ihtiyaç duyduğunu manuel olarak bulup hepsini tek tek indirmek zorundaydın. Eksik bir .jar dosyası demek, meşhur NoClassDefFoundError hatası demekti.
- **Standart Olmayan Proje Yapısı:** Ahmet projesini farklı bir klasör düzeninde tutardı, Mehmet farklı. Bir işe girdiğinde projenin neresinde ne var çözmek günler alırdı.
- **Derleme ve Paketleme Çilesi:** Kodu çalıştırmak için önce derlemen, sonra test etmen, sonra .war veya .jar haline getirmen gerekirdi. Bunların hepsi manuel komutlarla veya karmaşık Ant scriptleriyle yapılırdı.

4.2. Maven Nedir ve Neyi Çözdü?

Maven, en temel tanımıyla bir **Proje Yönetim ve İnşa Otomasyon (Build Automation)** aracıdır. "Convention over Configuration" (Yapılandırma yerine standartlar) prensibiyle çalışır.

Maven'in Getirdiği Devrimler:

1. **Merkezi Depo (Central Repository):** Artık internette .jar aramıyoruz. Maven'a "Bana Spring'in 5.3 versiyonunu getir" diyoruz, o gidip merkezi sunucudan otomatik indiriyor.
2. **POM.xml (Project Object Model):** Projenin "kimlik kartı". Proje ismi, versiyonu ve ihtiyaç duyduğu tüm kütüphaneler bu tek dosyada tanımlanır.
3. **Otomatik Bağımlılık Yönetimi:** Sen sadece A kütüphanesini istersin; Maven, A'nın çalışması için gereken B, C ve D'yi senin yerine tespit eder ve sessizce indirir.
4. **Standart Dizini Yapısı:** Maven der ki: "Kodların src/main/java içinde, testlerin src/test/java içinde olacak." Bu sayede bir Maven projesini açan her yazılımcı, dosyaların nerede olduğunu anında bilir.

4.3. Maven Build Lifecycle (Yaşam Döngüsü)

Maven, bir projenin doğumundan (kod yazımı) paketlenip dağıtılmasına kadar olan süreci standart fazlara böler. Bir komutla hepsini sırayla çalıştırabilirsin:

- **Validate:** Proje bilgilerinin doğruluğunu kontrol eder.
- **Compile:** Kaynak kodları derler.
- **Test:** Birim testleri (Unit Tests) koşturur.
- **Package:** Derlenen kodu dağıtılabilir formata (JAR/WAR) sokar.
- **Install:** Paketi yerel deponuza (Local Repository) atar (diğer projelerinizde kullanabilmeniz için).

4.4. Karşılaştırma Özeti

Özellik	**Maven Öncesi**	**Maven Sonrası**
Kütüphane Yönetimi	Manuel (İndir-Kopyala)	Otomatik (pom.xml)
Klasör Yapısı	Keyfi / Dağınık	Standart ve Düzenli
Versiyon Çakışması	Çok yaygın ve çözmesi zor	Maven tarafından otomatik yönetilir

Yeni Yazılımcı Adaptasyonu	Günler sürer	Dakikalar içinde projeyi ayağa kaldırır

Eğlenceli bir örnek: Maven öncesi yemek yapmak için tarlaya gidip buğday topluyor, un yapıyor ve odun kesiyordun. Maven sonrası ise sadece bir tarif (pom.xml) veriyorsun ve mutfak robotu her şeyi senin için hazırlıyor.

Not: Bu bölümden sonrasını derste birlikte yapabiliriz.

5. Geliştirme Ortamının Hazırlanması

- **JDK Kurulumu:** Güncel versiyon seçimi.
- **IDE Seçimi:** IntelliJ IDEA (tavsiyemdir), Eclipse veya NetBeans tanıtımı.
- **İlk "Hello World" Projesini Oluşturma.**

5.1. Bir Java Programının Anatomisi

İlk kodu ekrana yazdığınızda, oradaki her kelimenin ne anlama geldiğini (yüzeysel de olsa) açıklamalısın.

Java

```
public class Main {

   public static void main(String[] args) {

      System.out.println("Merhaba Java!");

   }

}
```

5.2. Temel Sözdizimi (Syntax) Kuralları

Java'nın katı ama güvenli kuralları:

- **Büyük/Küçük Harf Duyarlılığı (Case Sensitivity):** Main ile main aynı şey değildir.
- **Noktalı Virgül (;):** Her cümlenin sonundaki nokta gibi.
- **Süslü Parantezler ({ }):** Kod bloklarını (kapsamları) belirleme.

5.3. Değişkenler ve Temel Veri Tipleri

Verileri bellekte nasıl saklarız?

Veri Tipi	Açıklama	Örnek
int	Tam sayılar	int yas = 25;
double	Ondalıklı sayılar	double fiyat = 19.99;
char	Tek bir karakter	char harf = 'A';
boolean	Doğru/Yanlış	boolean aktif Mi = true;
String	Metin ifadeleri	String isim = "Ali"

Küçük bir not: Java "Strongly Typed" bir dildir. Yani bir değişkenin tipini başta söylerseniz, sonradan değiştiremezsiniz. Bu, hata payını azaltır!