

# Web Programlama 2

## Ders 01: Giriş

---

# Gerekli Araçlar

JDK 11

Git

Maven

IDE (IntelliJ IDEA'yı öneriyorum)

Docker

Komut Satırı Terminali

- Mac/Linux: Dahili olarak geleni kullanabilirsiniz
- Windows: GitBash'i öneriyorum

# Hedefler



**Test** ve **güvenlik** aşamalarına dikkat ederek enterprise uygulama geliştirme hakkında bilgi edinme



Veritabanı erişimi



Geliştirilmiş bean yapısı ile iş katmanı



HTML template ile *Server-side* rendering



Sadece teorik değil aynı zamanda build ve deploy işlemleri

# Enterprise Uygulamalar

- **Büyük seviyede** uygulamalar
  - Onlarca/Yüzlerce insan (geliştirici/yönetici/vs.) dahil olmaktadır
  - Pek çok farklı süreç çeşitli sunucularda gerçekleşmektedir.
  - Google, Amazon, Facebook, Netflix, vs.
- Bir web uygulaması yalnızca “*frontend*” kısmından oluşur ve çok da büyük uygulamalar değildir.
- *Backend*: veritabanları, (yüzlerce) web servisler, load balancer, gateways, vs.



# Web Programlama 1 vs Web Programlama 2

Bu derste, SQL veritabanı kullanan bir web uygulaması oluşturacağız.

Web Programlama 2 dersinde, RESTful API'lar ve mikroservislerden oluşan backend uygulamaları oluşturacağız.

Bu derste frontend'ten ziyade backend uygulamalarına odaklanıyor olsak da full-stack bir uygulama geliştireceğiz.

# WP 1 vs Web Uygulama Geliştirme

- “*Web Uygulama Geliştirme*” (WUG) dersinde JS ile yazılmış modern SPA uygulamaları geliştireceğiz.
    - Ayrıca REST ve GraphQL’e de giriş yapılacaktır.
  - WP 1 JS olmadan server-side rendering kullanarak (yalnızca HTML/CSS) daha eski bir bakış açısı ile bakacağız
  - WUG dersi WP 1 ve WP 2 dersinin ortasında bir yerdedir.
-

# Teknolojiler



Java Enterprise  
Edition (**JEE**)

Data layer (JPA/JTA)  
Business logic layer  
(EJB)  
Front-end layer (JSF)



**SpringBoot** Framework



Test: **Selenium**



Deployment: **Docker**



# Neden Java?

- En popüler programlama dillerinden biridir.
  - Büyük çaplı backend uygulamalarının temel dilidir.
  - *C#'taki .Net* şu anda gayet iyi bir alternatiftir.
    - Bu dersteki öğreneceğimiz pek çok konsept aynı zamanda C#/.Net'de de farklı kütüphane/framework'lerle uygulanabilir.
  - İyi bir *strongly-type dile ihtiyaç olduğundan*
    - JavaScript, Python, Ruby, vs gibi değil.
-



# Spring vs JEE

---

- **JEE**, Java Enterprise uygulamaları için oluşturulmuş “resmi” frameworktü ancak şu anda değildir (2017’den beri)
  - Şu anda sadece diğerleri gibi bir framework
- **Spring** JEE üzerine inşaa edilmiş ve Pivotal tarafından geliştirilmiş bir başka framework’tür.
- Şimdi olduğu gibi, Spring daha fazla kullanılmaktadır ve daha hoş bir yapıdadır.
- Ancak WildFly container’da deploy ederken Arquillian ile EJB testi yapmanın zorluğunu yaşamadan SpringBoot’u gerçekten takdir edemezsiniz.

# Uygulamalar

- Her hafta bir uygulama yapacağız
- Veritabanı erişimi
- Server-side HTML rendering (JSF)
- Cloud deployment
- *intro/exercises/quiz-game/part-11/frontend/src/test/java/org/webp/intro/exercises/quizgame/LocalApplicationRunner.java'dan başlatılabilir.*
- *localhost:8080*



Sadece  
geliştirme  
değil...

Bu derste, **TEST** ve  
**GÜVENLİĞE** de oldukça  
fazla önem vereceğiz.

*Peki neden???*

# Ariane 5

- 4 Haziran 1996 yılında Ariane 5 fırlatması başarısızlıkla sonuçlandı.
- \$500 milyon maliyet
- **Software bug**



a key player in finding the cause of the Challenger explosion

# Ölümcül Therac-25 Radyasyonu

1986, Texas, insan öldü



# 2003 yılında elektrik kesintisi

Kanada ve Amerikada yaklaşık  
50 milyon insan etkilendi.



2010, Toyota,  
fren sisteminde  
yazılım hatası,  
436,000 araç  
geri çağırıldı



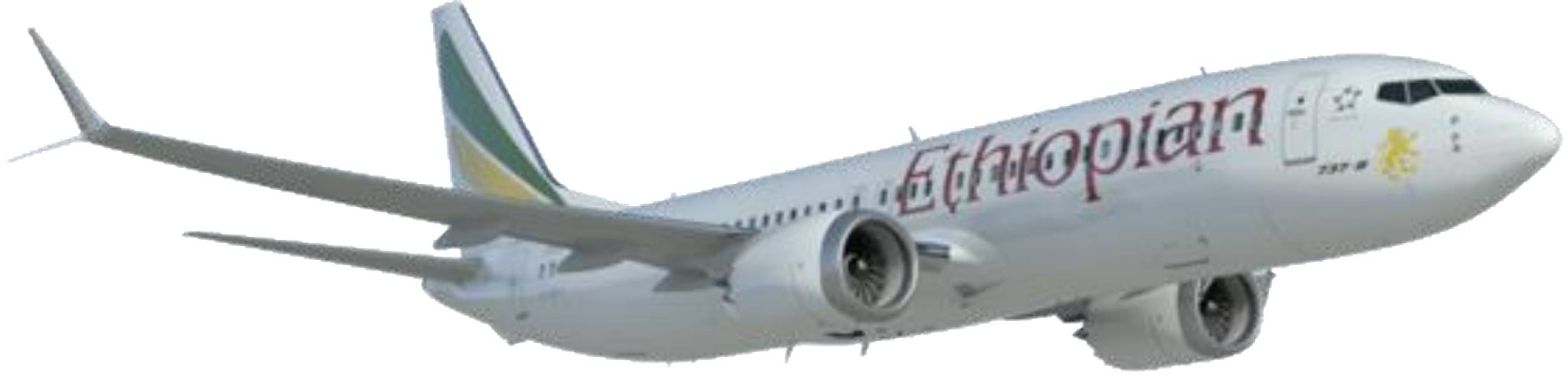


\$460 milyon kayıp, ticari işlemler sırasında 45 dakika süren bir bug sebebiyle


# Knight Capital Group 2012







2019, bir Boeing 737 Max yazılım hatası sebebiyle kaza yaptı; **157** insan öldü.



# Ve bütün bir gün bunun gibi örnekler devam edebilir...

2013 itibariyle, yazılım testinin dünya çapında 312 milyar dolara mal olduğu tahmin ediliyor

2016 yılında, **548** kayıtlı ve belgelenmiş yazılım hatası, dünya çapında **4,4 milyar insanı** ve **1,1 trilyon dolarlık mal varlığını** etkiledi.

# Güvenlik

- Önceki slaytlar “*fonksiyonel*” buglardan oluşmaktaydı
- Ancak güvenlik açıkları da enterprise sistemlerde bug olarak geçmektedir.
- Yalnızca güvenlikle ilgili en iyi uygulamaları ve farklı saldırı türlerini bilmek yetmemektedir, aynı zamanda uygulamalarınızı nasıl koruyacağınızı da bilmeniz gerekir.

# 2013-2014:

- 3 Milyar hesap etkilenmiştir.
  - Evet milyar, milyon değil...
- Bilgilerin çalınması oldukça kötü bir durumdur özellikle de kredi kartı numaraları çalındıysa...
- Ancak bir diğer temel problem nedir? İnsanlar aynı şifreleri farklı uygulamalar için de kullanır. Ör: Gmail, Facebook, vs.
- Yahoo şifreleri oldukça kolay bir şekilde çözüldü (ör, MD5)
- *Kaçınız farklı siteler için farklı şifreler kullanıyorsunuz?*

The image shows the classic Yahoo! logo in a purple serif font. The letters are bold and slightly stylized, with a registered trademark symbol (®) at the end of the exclamation mark.

# 2014: eBay

---

- 145 million hesap etkilendi
- Ancak kredi kartı bilgileri farklı bir veritabanında tutuluyordu.



# 2017: Equifax

- 143 milyon müşteri etkilenmiştir.
- Kişisel bilgiler
  - Sosyal güvenlik numarası, doğum tarihleri, adresler ve sürücü belgesi numaralarını da içeren pek çok bilgi çalınmıştır.
- 209,000 müşterinin kredi kartı bilgileri de çalınmıştır.

# 2011: Sony's PlayStation Network

---

- 77 milyon hesap etkilendi
- 12 milyon şifrelenmemiş kredi kartı numarası çalındı.
- Site bir ay boyunca kapalı kaldı
- Yaklaşık \$200 milyon kayıp



# Java Enterprise Edition (JEE)



# JEE nedir?

- Enterprise uygulama geliştirmek için bir dizi kütüphaneler barındırır.
- Farklı olası uygulamalara ait bir dizi “arayüz” olarak düşünebilirsiniz.
  - Ör, Hibernate ve EclipseLink JPA özellikleri için iki farklı uygulamadır.

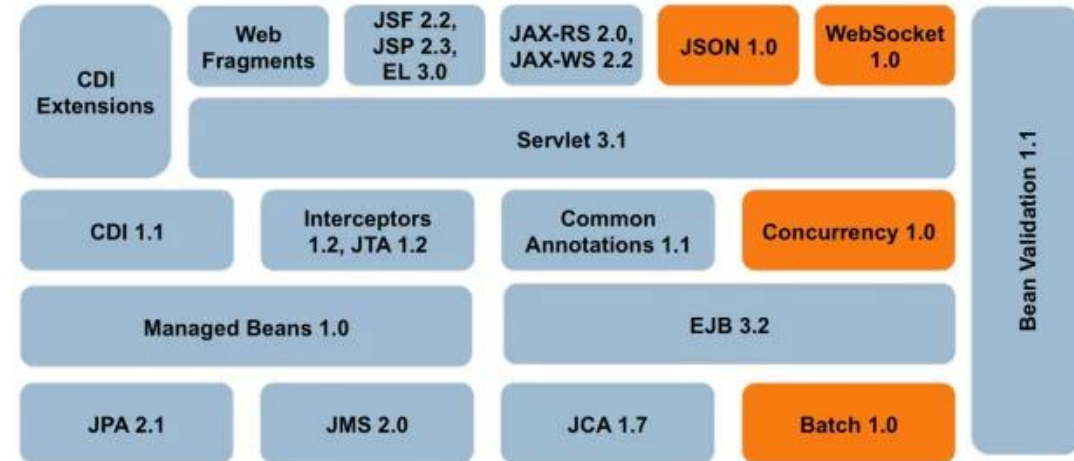


# Tarih

- 1998: JPE (Java Platform for the Enterprise)
    - Sun Microsystems çatısı altında, Java geliştirme
  - 2009: Oracle, Sun'ı satın aldı
  - 2013: Java EE 7
  - 2017: Java EE 8
  - 2017: Oracle, EE'yi Eclipse Foundation'a verdi.
    - Artık “*Jakarta EE*” olarak adlandırılıyor.
-

# Bu dersteki Java EE Özellikleri

- **JPA:** Java Persistence API
  - Veritabanı erişimi için
- **Bean Validation**
  - Veri üzerindeki kısıtlamalar için
- **EJB:** Enterprise Java Beans
  - İş katmanı için
- **Servlet**
  - HTTP isteklerini ele almak için
- **JSF:** JavaServer Faces
  - Web GUI oluşturmak için
- Ve diğerleri...



# JEE Dağıtıcıları

- Bir Java EE uygulama geliştirildiğinde bunları koşturmak için bir container seçmeniz gerekir.
- Farklı dağıtıcı ve uygulamaları:
  - RedHat: JBoss ve **Wildfly**
  - Oracle: GlassFish ve WebLogic
  - IBM: WebSphere
    - 2018 yılında IBM RedHat'i satın aldı.
  - Payara Services: Payara
  - vs.

# Neden container???

## İdeal dünyada...

- JAR/WAR dosyaları küçüktür ve kütüphaneleri paketlemeye ihtiyaç yoktur.
- Farklı EE uygulamaları aynı container üzerinde çalışır.
- Farklı containerlara deploy edilebilir ve tek bir implementasyona bağımlı değildir.

## Gerçek dünyada...

- Çok çok fazla container'ı ele almak ve ayarlama yapmak gerekir.
- Çok daha kötü testler: az otomasyon, development ve production ortamlarındaki uyumsuzluklar
- Container değiştirmek oldukça zor...

# “Kısmi” Container’lar: Web Servers

- JEE özelliklerini tamamen desteklemez
  - Temel olarak **Servlet** and web assets desteklenir.
  - **Tomcat** ve **Jetty** en ünlüleridir.
  - EE kütüphane olarak eklenebilir (ör, JPA için *Hibernate*)
  - Uygulamalar gömülebilir.
    - Ör: çalıştırılabilir JAR dosyaları
  - Bu yaklaşım *SpringBoot* Framework tarafından kullanılır.
-



# Maven

# Ders İçeriği

- Bu derste **Maven** kullanılacaktır:
- 150'den fazla Maven alt modülü ve çok fazla katman bulunmaktadır.
- Enterprise sistemlerde bu oldukça olağan bir durumdur.
- Maven'in nasıl çalıştığını anlamaya ihtiyaç vardır.



# Build Araçları

## Maven

- *En popüleridir*, XML tabanlıdır, kendi birincil tercihim
- Biraz ayrıntılıdır, ancak IntelliJ'in otomatik tamamlama özelliği ile çok sorun yaratmamaktadır.

## Gradle

- Android için popülerdir, script tabanlıdır.
- Script tabanlı olması **en iyi** ve **en kötü** özelliğidir.
  - İyi: Çok esnektir.
  - Kötü: bakımı zor ve yeni geliştiriciler için kullanımı zordur.
- Not: Maven bağımlılık kütüphanelerini ele alır.

## Ant

- Eskidir ve artık pek kullanılmamaktadır.

# Build Araçlarının Görevi

- Kodu compile etmek
- Karmaşık modülerliği çözmek
- Üçüncü parti uygulamaları otomatik indirmek
- Özel pre/post işlemleri uygulamak
- Test senaryolarını koşturmak
  - Ör, Continuous Integration'da bir test başarısız olursa bütün build işlemi de başarısız olur.
- *Kolay bir şekilde yeni bir makineye uygulamayı build etmek.*

# Maven “pom.xml” dosyası

- POM: Project Object Model
- XML dosyası bir modülün nasıl build edileceğini tanımlar
- Proje birden fazla modülden oluşabilir ve her birinin de ayrı bir *pom.xml* dosyası bulunur.
- Modül ve submodül hiyerarşisi tanımlanabilir

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<groupId>org.webp</groupId>
```

```
<artifactId>tsdes</artifactId>
```

```
<version>0.0.1-SNAPSHOT</version>
```

```
<packaging>pom</packaging>
```

```
<name>Root of WEBP</name>
```

```
<modules>
```

```
  <module>intro</module>
```

```
  <module>advanced</module>
```

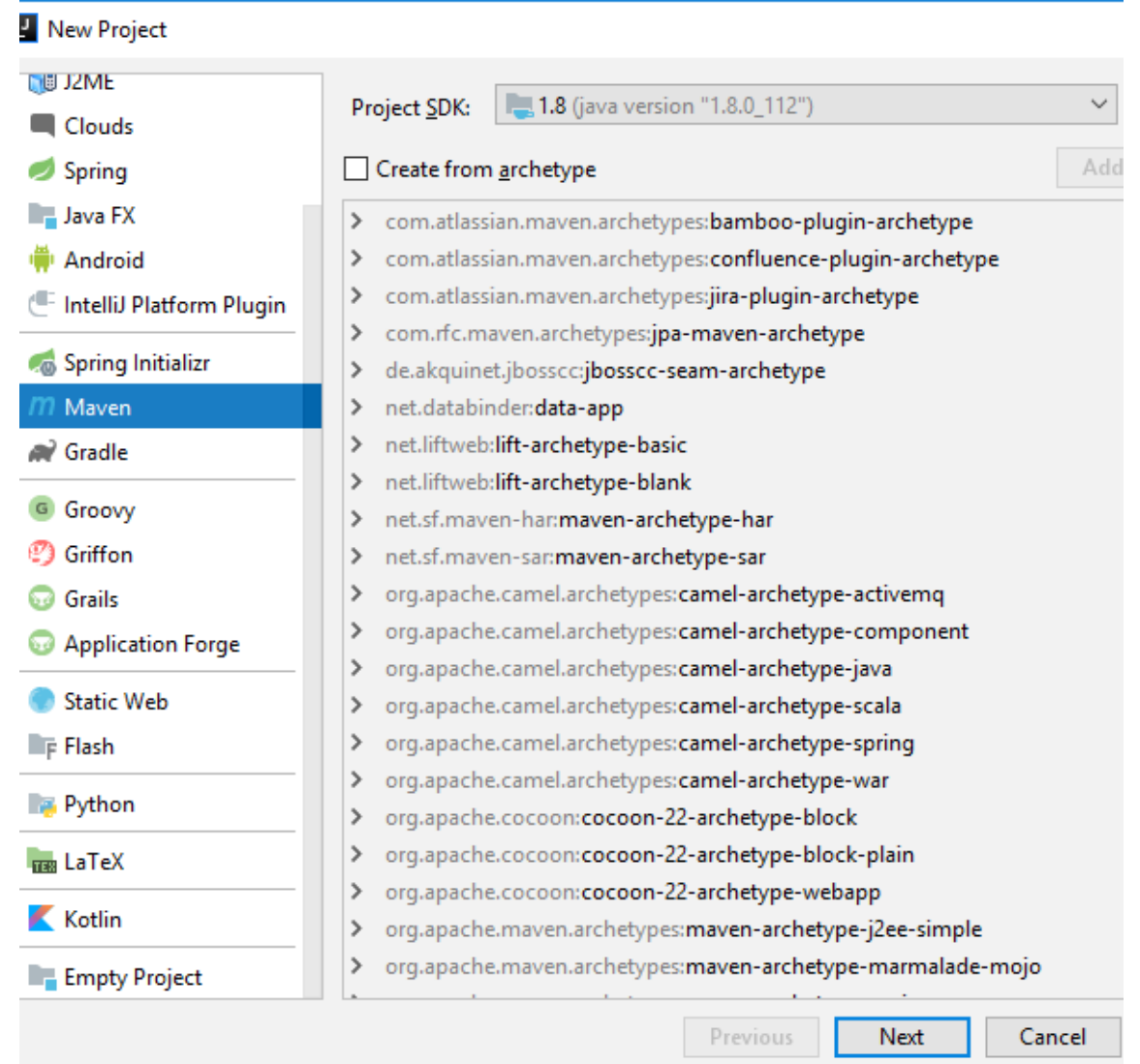
```
</modules>
```

...

- **<project>** namespaces ve XSD (XML Schema Definition) tanımı yapılır
- *pom.xml* bütün XML taglarını barındırmaz yalnızca XSD schema'da tanımlanan taglar bulunabilir.
- Maven komutu çalıştırıldığında, *pom.xml*'in içeriğini parse eder.

# Yeni Proje

- IDE bir pom.xml oluşturmak için sorar
- ... veya sadece var olan bir pom.xml'i kopyala/yapıştır/düzenle aşamaları ile de çalıştırabilirsiniz.



## Module/Artifact

- Her bir module/artifact tekil 3 tag ile tanımlanır.
  - **<groupId>**: İlgili artifact grubunu tanımlayan bir string id'dir.
  - **<artifactId>**: Bir grup içerisinde benzersiz bir id'dir.
  - **<version>**: Modül/artifact versiyon numarasıdır. Genellikle M.m.p sayısal format kullanılır, örnek, Major-Minor-Patch version
    - Genellikle geliştirme ortamındaysa ve yayınlanmadıysa SNAPSHOT eklenir.
- 
- **<groupId>**org.webp**</groupId>**  
**<artifactId>**webp**</artifactId>**  
**<version>**0.0.1-SNAPSHOT**</version>**

# Paketleme tipleri

- **<packaging>**: **pom**, **war** veya **jar** olarak paketlenebilir.
  - **pom**: Bu modül **<modules>** tagi içinde tanımlı diğer modülleri oluşturmaktan sorumludur.
    - Farklı alt modüllerde ortak olan ayarları paylaşmak için kullanışlıdır.
  - **war**: bir WAR (**W**eb application **AR**chive) dosyası oluşturur.
    - Bu dosyalar WildFly gibi EE containerlarına deploy edilebilir.
  - **jar**: bir JAR (**J**ava **AR**chive) dosyası oluşturur.
    - Bütün derlenmiş kodu barındıran tek bir dosyadır.
    - Teknik olarak, ziplenmiş bir dosyadır ve unzip yaparak kullanılabilir.
-

# Maven kullanımı

- Maven bir IDE üzerinde çalıştırılabilir ama öğrenmek için bir komut satırında çalıştırılması daha doğrudur.
- Son versiyonun indirilmesi gerekmektedir.
- Geliştirici olarak, GUI'si olmayan ve işimizi kolaylaştıracak pek çok araç bulunmaktadır ve bunları kullanabilmek önemlidir.
- Kendi kendine çalışabilen JAR dosyaları ve Docker ile uğraşırken bu noktaya geri döneceğiz.



MINGW64:/c/Users/Vita

Vita@DESKTOP-DKKIB9P MINGW64 ~

\$ mvn -version

Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-27T18:06:16+03:00)

Maven home: F:\Program Files\apache-maven-3.6.2

Java version: 1.8.0\_231, vendor: Oracle Corporation, runtime: F:\Program Files\Java\jdk1.8.0\_231\jre

Default locale: tr\_TR, platform encoding: Cp1254

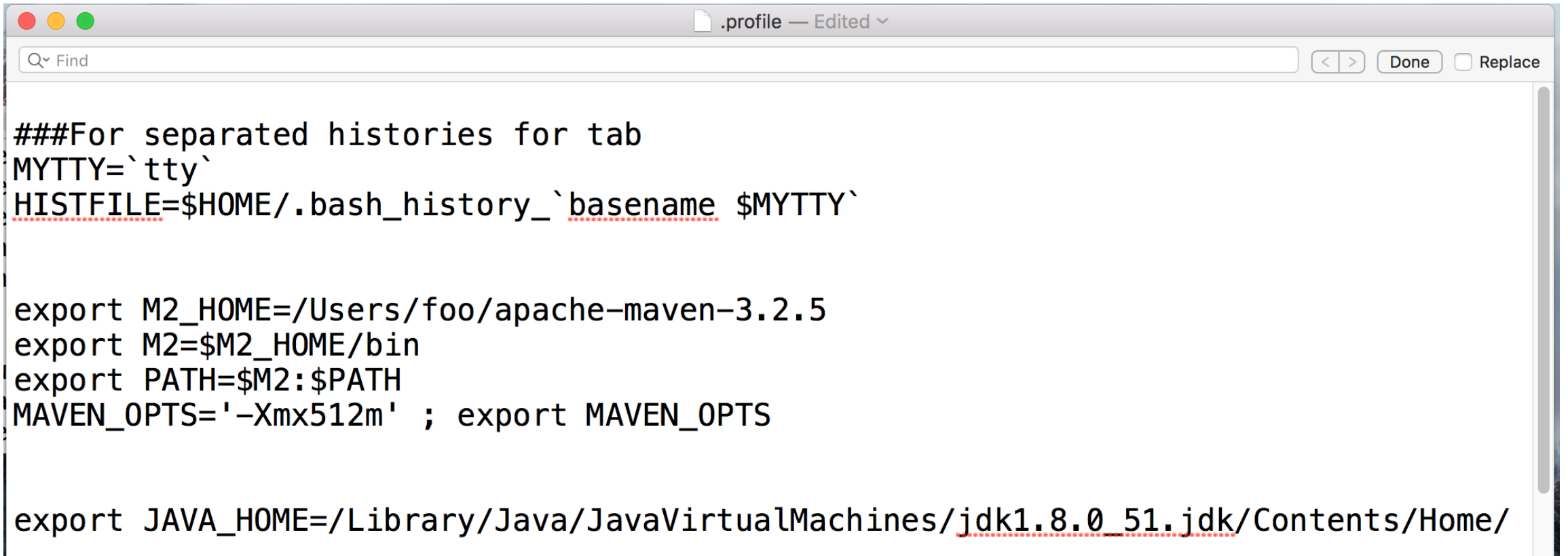
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

Vita@DESKTOP-DKKIB9P MINGW64 ~

\$ |

- Maven'i komut satırında çalıştırdığından emin olmalısın.
- Windows'ta, GitBash'ı deneyebilirsiniz.
- **PATH** tanımı yapmaya ihtiyaç vardır.
- Eğer her şey tamamlandıysa, "*mvn -version*"

- Mac sistemlerde, home klasörü içerisindeki “*.profile*” dosyasının düzenlenmesi gerekmektedir.
- Tabii ki, JDK ve Maven’i nereye kurduysanız mevcut yol da o olmalıdır.



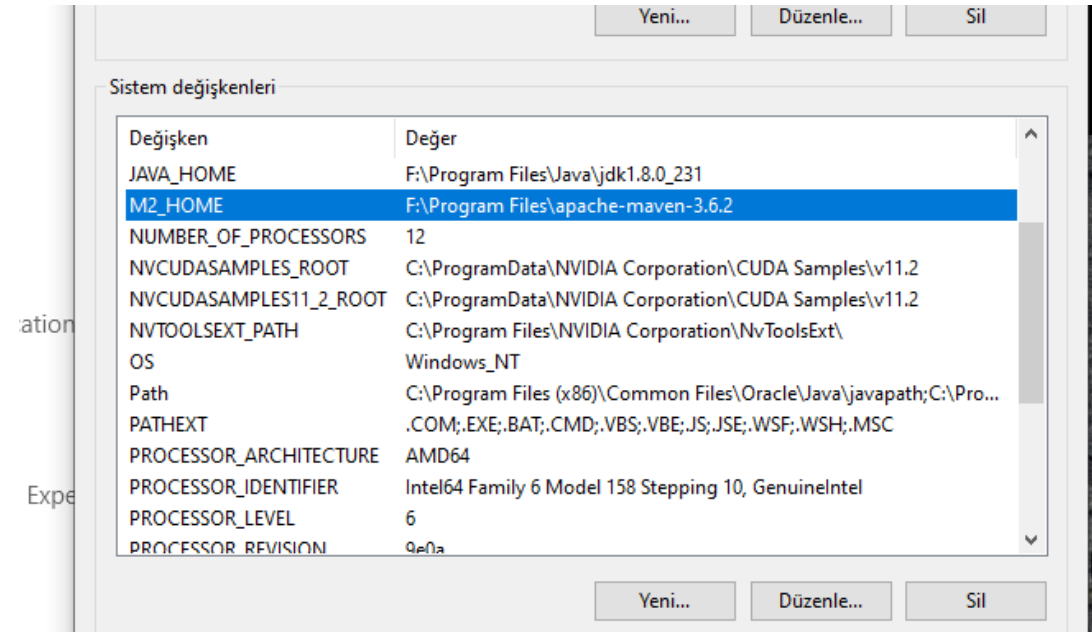
The screenshot shows a Mac text editor window titled ".profile — Edited". The window contains the following text:

```
###For separated histories for tab
MYTTY=`tty`
HISTFILE=$HOME/.bash_history_`basename $MYTTY`

export M2_HOME=/Users/foo/apache-maven-3.2.5
export M2=$M2_HOME/bin
export PATH=$M2:$PATH
MAVEN_OPTS='-Xmx512m' ; export MAVEN_OPTS

export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_51.jdk/Contents/Home/
```

- Windows'ta, **MAVEN\_HOME** ve **JAVA\_HOME**'un ayarlanması, ardından **PATH**'in güncellenmesi gerekmektedir.

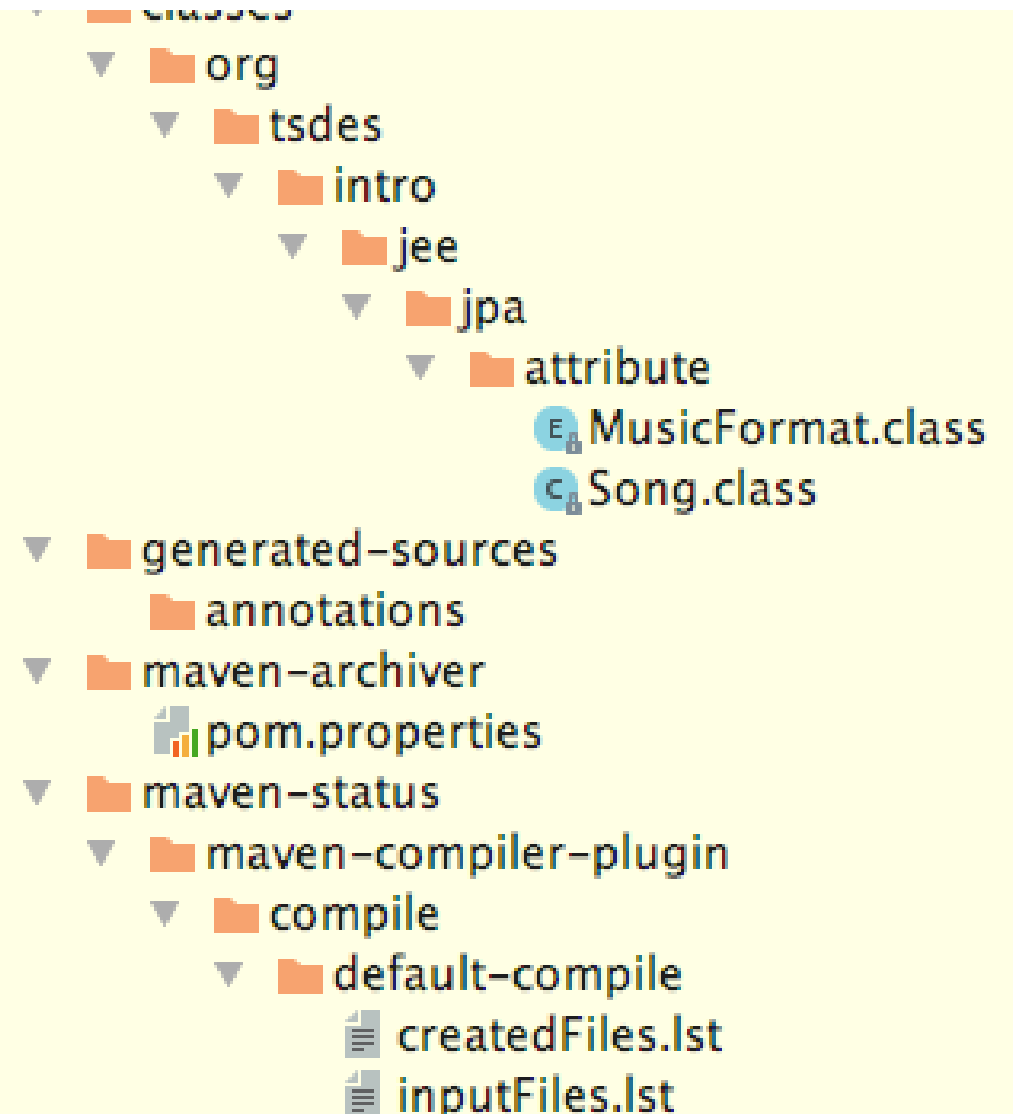


# PATH değişkeninin ayarlanması

- Bir komut çalıştırdığında, OS PATH değişkeninin içeriğine bakar ve içerisinde çalıştırılabilir uygulama olup olmadığını kontrol eder.
- Bash terminaline, *“echo \$PATH”* yazarsanız içeriğini görebilirsiniz.
  - *“echo”* string ifadeyi yazdır
  - *“\$”* değişkeni çözümle
- PATH’e eklenmiş *“mvn”* komutunun dosya konumunda çalıştırılabilir dosyaya ihtiyaç vardır.
- *“MVN\_HOME”* tanımladıysan, then *PATH* şöyle olabilir:
  - Linux/Mac: *“export PATH=\$MAVEN\_HOME/bin:\$PATH”*
  - Windows: *“%MAVEN\_HOME%\bin”*
    - Windows’ta, değişkenler iki % arasında gösterilir, \$ işareti ile değil.

# Mvn Clean

- “*mvn clean*”
- Proje içerisinde “*clean*” komutu kullanırsan generate edilmiş bütün dosyalar silinir.
- Bir proje build edildiğinde “target” klasörü içerisine bütün derlenmiş dosyalar (.class dosyaları) ve diğer build çıktıları (ör, JAR ve WAR files) kaydedilir.

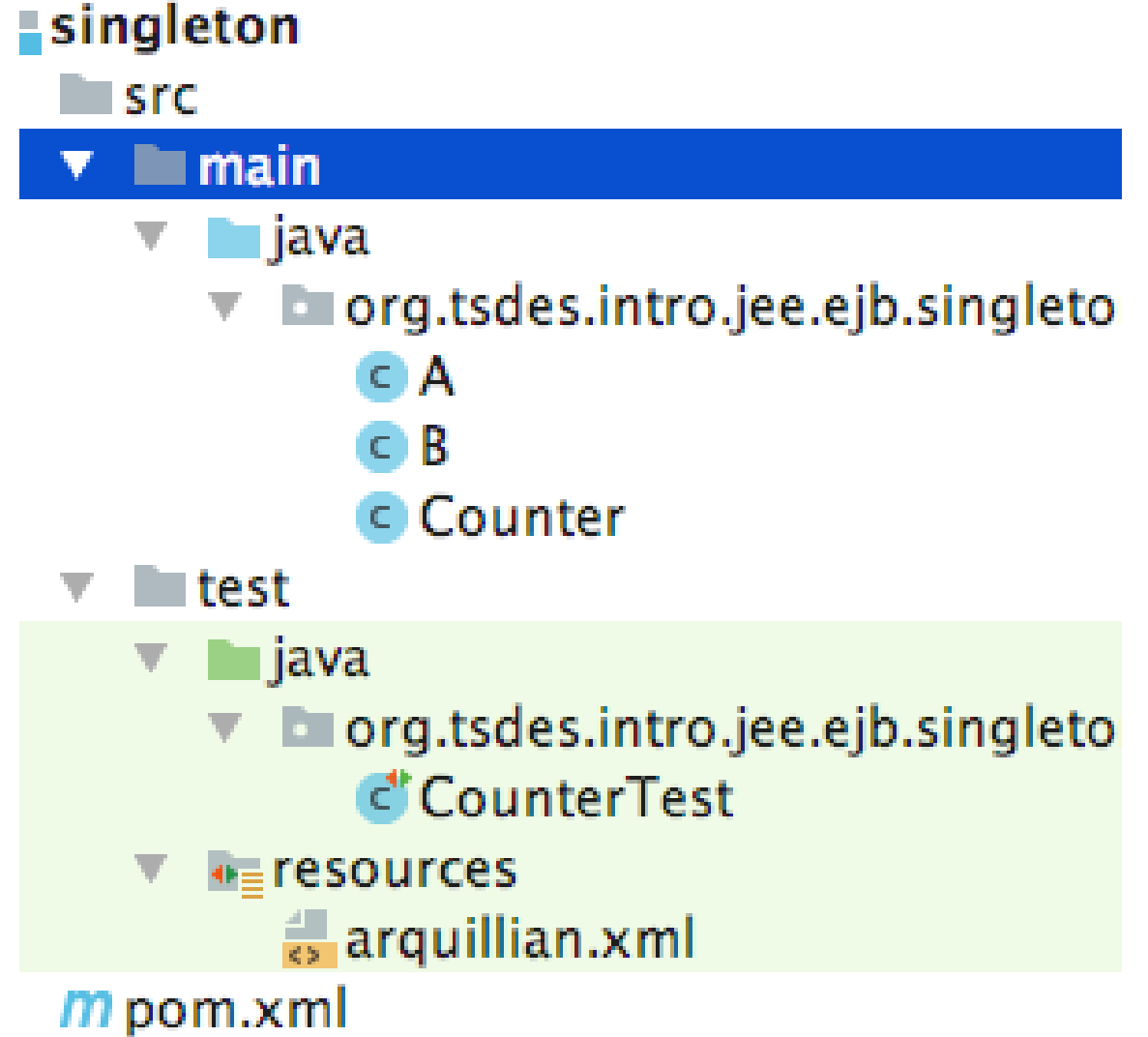


# Maven Temel Aşamaları

- **compile**
- bütün *.java* dosyalarını derleyerek *.class'a* çevirir
- **test**
- Bütün birim testleri çalıştırır
- **package**
- WAR/JAR dosyası oluşturur
- **verify**
- integrasyon testlerini çalıştırır
- **install**
- WAR/JAR dosyasını yerel maven repository'sine kopyalar

- “**mvn package**” çalıştırıldığında önceki bütün aşamalar da çalıştırılır.
- Not: Daha farklı aşamalar da bulunmaktadır ancak en önemlileri bunlardır.

- Maven dosyaların uygun klasörlerde olmasını istemektedir.
- **src/main/java**: Java kaynak kodları
- **src/main/resources**: JAR/WAR dosyasına eklenecek dosyalar
- **src/test/java**: Testler bulunur
- **src/test/resources**: Test için kaynakların bulunduğu klasör
- Bu default klasörleri değiştirebilirsiniz ancak pek önerilmemektedir.



# IDE'de Maven

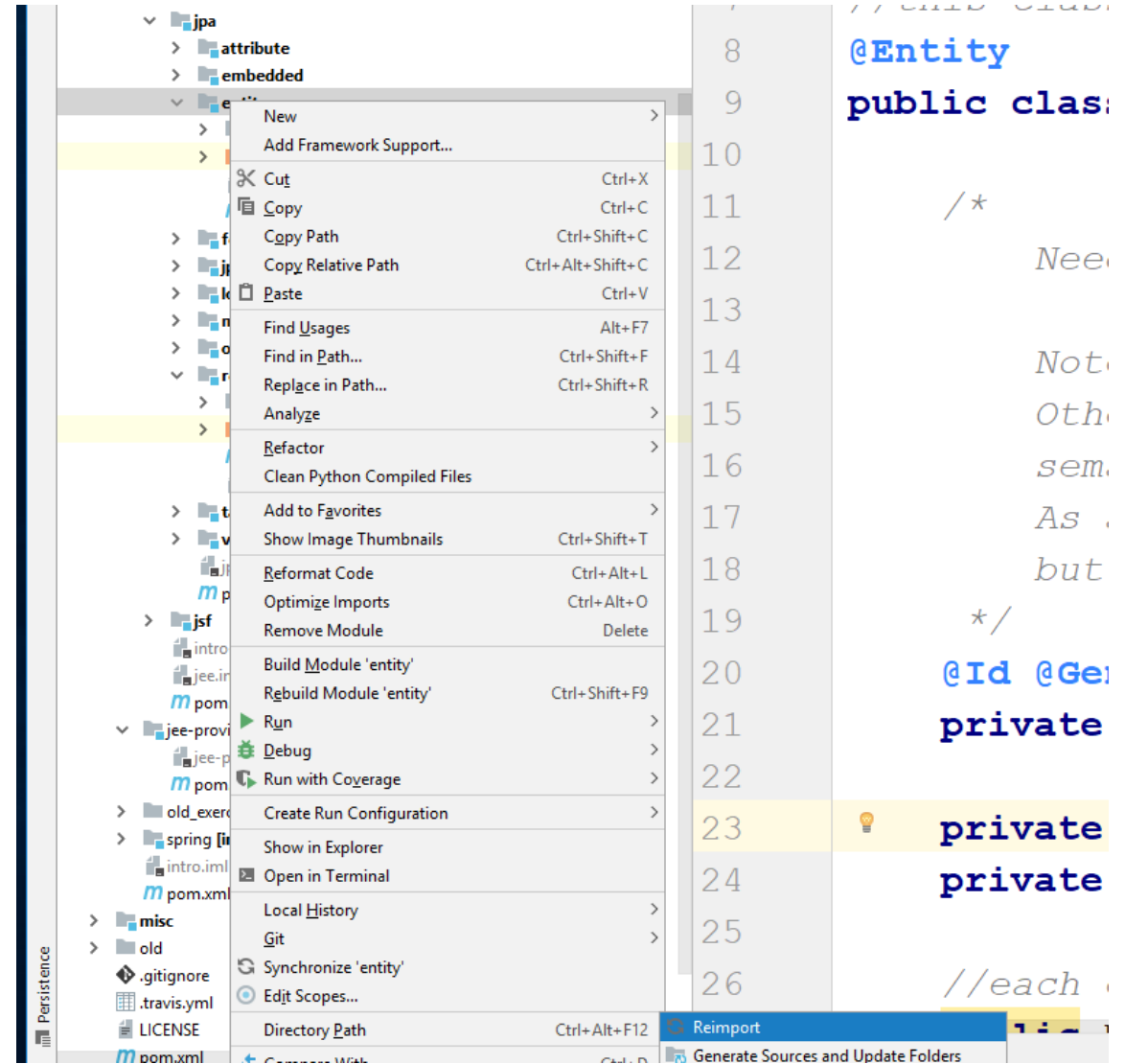
- Bir IDE'de Maven projesi açtığınızda genelde IDE otomatik olarak ayarları yapacaktır.
- Maven desteğinin aktif olduğundan emin olmalısınız.
  - Projeye sağ tıklayın ve maven'i seçerek *"Add Framework Support..."* deyin.
- Örnek: Eğer **src/test/java** klasörünü proje açıldıktan sonra eklerseniz, IDE bunun test klasörü olduğunu bilemeyebilir.
  - Maven'in otomatik güncellemesi aktif edilmelidir, genellikle pom.xml içeren bir proje ilk defa açıldığında bunu size sorar.



# Eğer IDE'de ayarlar algılanmazsa

- Eğer değişiklikleri IDE otomatik olarak algılayamazsa, *reimport* diyerek bunu zorlayabilirsiniz.
- Klasörleri elle seçmeyiniz.

Örnek: IntelliJ'de “*Mark Directory as*”



## 3. Parti Kütüphaneler

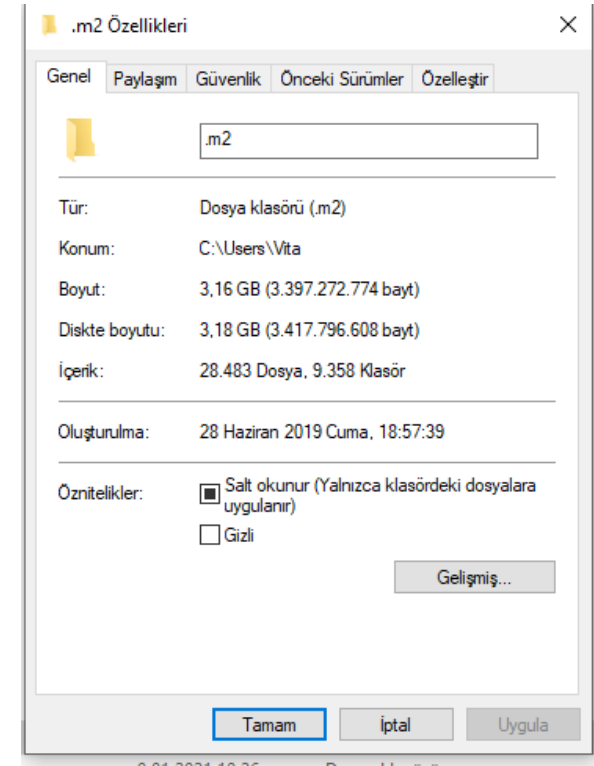
- Maven'in temel faydalarından biri bağımlılıkları otomatik olarak indirmesidir.
  - *pom.xml* dosyasına ekleyerek gerçekleştirilir.
  - *Maven* böyle bir jar dosyasının “~/.m2” klasörü içinde bulunup bulunmadığını kontrol eder.
    - “~” kullanıcı hesabının home klasörüdür.
  - Eğer yoksa, indirir.
- 
- **<dependency>**  
  **<groupId>junit</groupId>**  
  **<artifactId>junit</artifactId>**  
  **<version>4.12</version>**  
  **<scope>test</scope>**  
**</dependency>**

- Yıllar geçtikçe, boyutu da artmaktadır (ör: benimki 3.16 GB)
- Dosya adının önündeki “.” karakteri Mac/Linux sistemlerde gizli dosyadır.
- “2” ifadesi eski *Maven 2.x* versiyonu gösterir (3.x geriye dönük uyumludur ancak 1.x değildir)

Görünüm

bilgisayar > Yerel Disk (C:) > Kullanıcılar > Vita > .m2 > repository

Ad	Değiştirme tarihi	Tür
antlr	28.06.2019 19:02	Dosya klasör
aopalliance	28.06.2019 19:00	Dosya klasör
args4j	4.01.2021 19:22	Dosya klasör
asm	20.10.2019 20:14	Dosya klasör
avalon-framework	28.06.2019 19:08	Dosya klasör
backport-util-concurrent	28.06.2019 18:58	Dosya klasör
batik	13.04.2021 15:11	Dosya klasör
biz	10.11.2019 14:22	Dosya klasör
bouncycastle	11.04.2021 13:35	Dosya klasör
cglib	13.06.2021 23:17	Dosya klasör
ch	10.11.2019 14:22	Dosya klasör
classworlds	28.06.2019 18:57	Dosya klasör
com	13.06.2021 23:18	Dosya klasör
commons-beanutils	10.11.2019 14:20	Dosya klasör
commons-chain	28.06.2019 19:08	Dosya klasör
commons-cli	28.06.2019 18:58	Dosya klasör
commons-codec	28.06.2019 18:58	Dosya klasör
commons-collections	28.06.2019 19:08	Dosya klasör
commons-configuration	20.10.2019 14:38	Dosya klasör
commons-digester	28.06.2019 19:08	Dosya klasör
commons-fileupload	4.01.2021 19:21	Dosya klasör
commons-httpclient	28.06.2019 18:58	Dosya klasör
commons-io	28.06.2019 18:58	Dosya klasör
commons-jxpath	20.10.2019 14:38	Dosya klasör
commons-lang	28.06.2019 19:08	Dosya klasör
commons-logging	28.06.2019 19:01	Dosya klasör



# Temel Bağımlılık Scope'lar <scope>

- **compile**: varsayılan olan
  - **provided**: derlerken ihtiyaç duyulur ancak oluşturulan JAR/WAR dosyasına dahil edilmez.
  - **test**: yalnızca test için gereklidir ve JAR/WAR dosyası içerisinde oluşturulmaz
    - ör, testleri çalıştırmak için JUnit kütüphanesi
  - **import**: POM bağımlılıkları için kullanılır.
    - Kütüphaneler pek çok kütüphaneye bağımlıdır ve her birini tek tek manuel olarak eklemek zorunda değilsiniz.
-

# Bill of Materials (BOM)

- Spring örneği
- Pek çok bağımlılık bulunduğunda, versiyonları senkron hale getirmek istendiğinde
  - ör, 100 bağımlılık varsa ve bunları aynı versiyon numarası ile update etmek istiyorsak sadece 1 yerde değişiklik yaparak diğer modüllere de yansması sağlanabilir.

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-parent</artifactId>  
  <version>${version.springboot}</version>  
  <scope>import</scope>  
  <type>pom</type>  
</dependency>
```

// pom.xml

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
```

//JAR/WAR oluşturulan modül içinde

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  </dependency>
```

- Bir kütüphaneyi farklı farklı modüllerde kullanmak isteyebilirsiniz.
- Kopyala/Yapıştır engellemek ve **<version>/<scope>** ile her yerde bakımını gerçekleştirmek için, ata pom.xml içerisinde **<dependencyManagement>** kullanılır, ör: root dizinde
- Bütün alt modüller kalıtımla **<version>/<scope>** değerlerini alır.

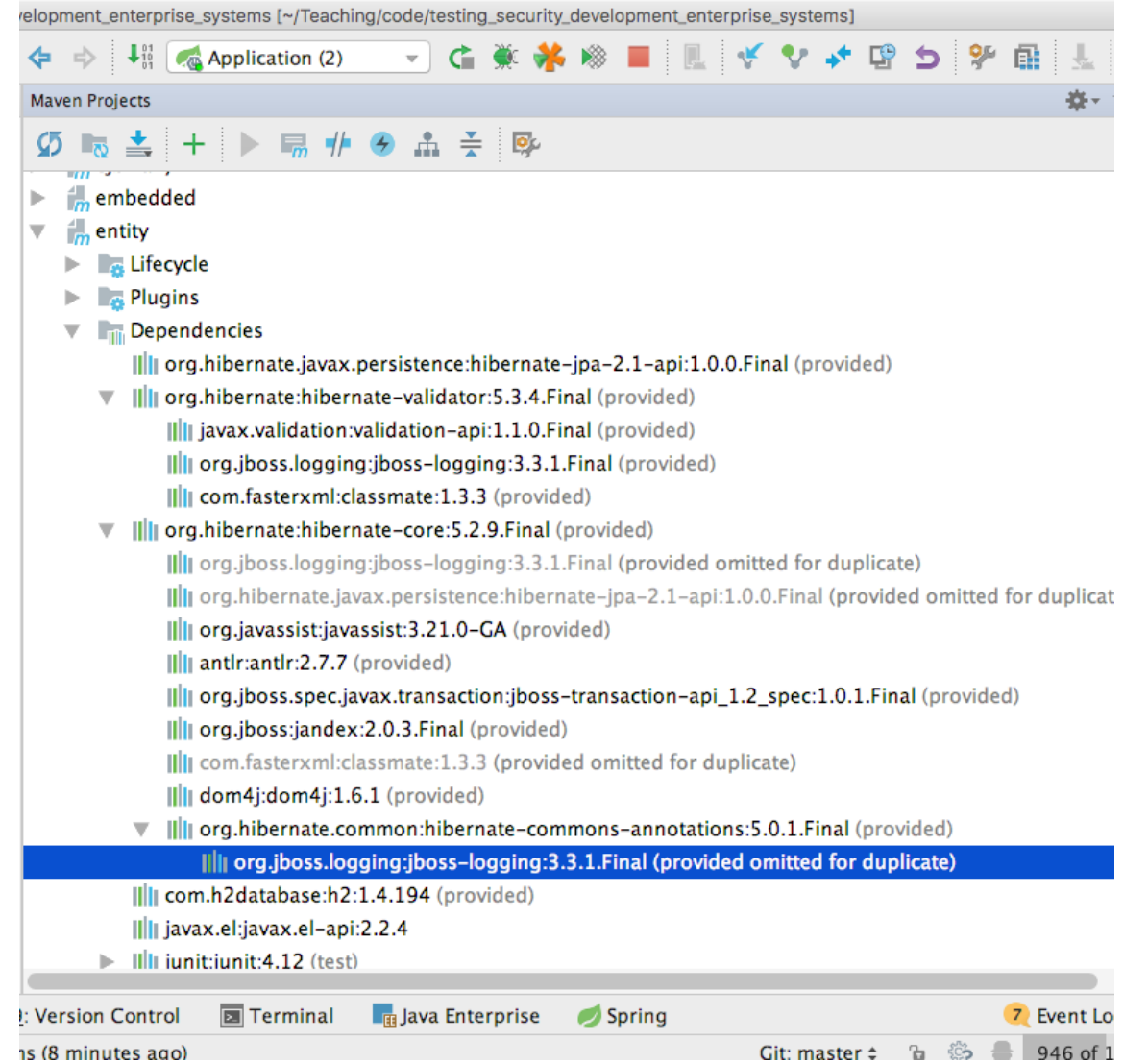


# <dependencyManagement> Karmaşası

- Bağımlılıklar **<dependencies>** tagi içindeki **<dependency>** tagında belirtilir
    - Ardından uygulamada kullanılır olacaktır.
  - Ayarlama tanımlamaları ise **<dependencies>** ve **<dependency>** tagı bulunan **<dependencyManagement>** içerisinde yapılır.
  - Burada tanımlananlar uygulama içerisinde kullanılmaz, sadece ayarlama için kullanılır.
-

# Bağımlılık Bağımlılıkları

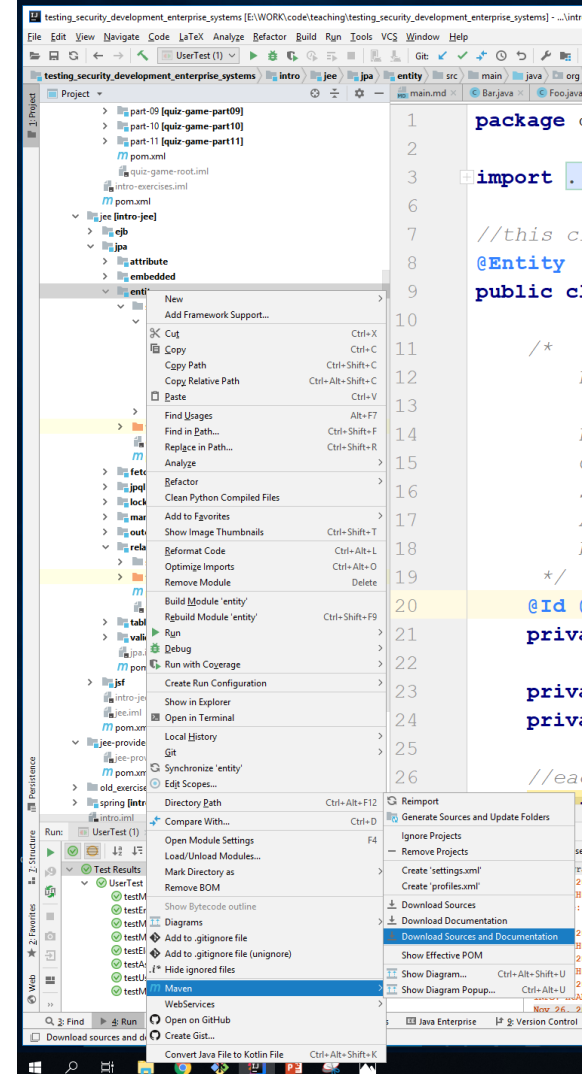
- Bir bağımlılık kendi bağımlılıklarına sahip olabilir.
- Ve diğer bağımlılıklar da kendi bağımlılıklarına sahip olabilir, bu da böylece gider...
- IntelliJ'de “*Maven Projects*” içerisinde hangi bağımlılıkların bulunduğu görünebilir.
- Örnek: *hibernate-core*, *hibernate-commons-annotations*'i çağırırken o da *jboss-logging*'i çağırır.





# Debugging

- Uygulamayı çalıştırmak için bağımlılığa ait JAR dosyasındaki bytecode'a ihtiyaç vardır.
- Debug için o bağımlılığa ait *JavaDocs* ve *source-code*'a ihtiyaç duyabilirsiniz.
- Not: Kaynak koda sağ tıkladıktan sonra gelen “**Go To -> Implementation(s)**” komutu ile erişebilirsiniz.
- IDE'ye kaynak kodları indirmesini söylemenize ihtiyaç olabilir.



# Java Packages

- Sınıflar daha iyi organize edilebilmesi için farklı klasörler içerisinde (anlamli isimlerle) bulunabilir.
- Java, *packages*'ı ilgili sınıfları gruplamak için kullanır.
- Bir sınıf ismi ve package'ı ile tanımlanır.
  - Ör, *java.lang.String*, *String* sınıf ismi ve *java.lang* package adıdır.
- Java'da, bir package dosya yapısıyla uyumlu olmalıdır.
- Örnek: *org.webp.intro.jee.jpa.entity.User01*,  
*src/main/java/org/webp/intro/jee/jpa/entity/User01.java* içerisinde bulunmalıdır.
  - Her bir "." klasör içindeki klasörü temsil eder



# Modules vs. Packages

- *Packages* büyük çaplı projelerin organizasyonunda oldukça önemlidir
  - *Maven Module'leri* de aynı rolü oynarken çok büyük projelerin farklı parçalara ayrılmasını sağlar
    - Package ile kıyaslandığında bir seviye üstüdür
  - Modüller bağımsız olarak derlenebilir ve kendi farklı çıktılarına sahip olabilirler
    - Ör. JAR/WAR dosyaları
-

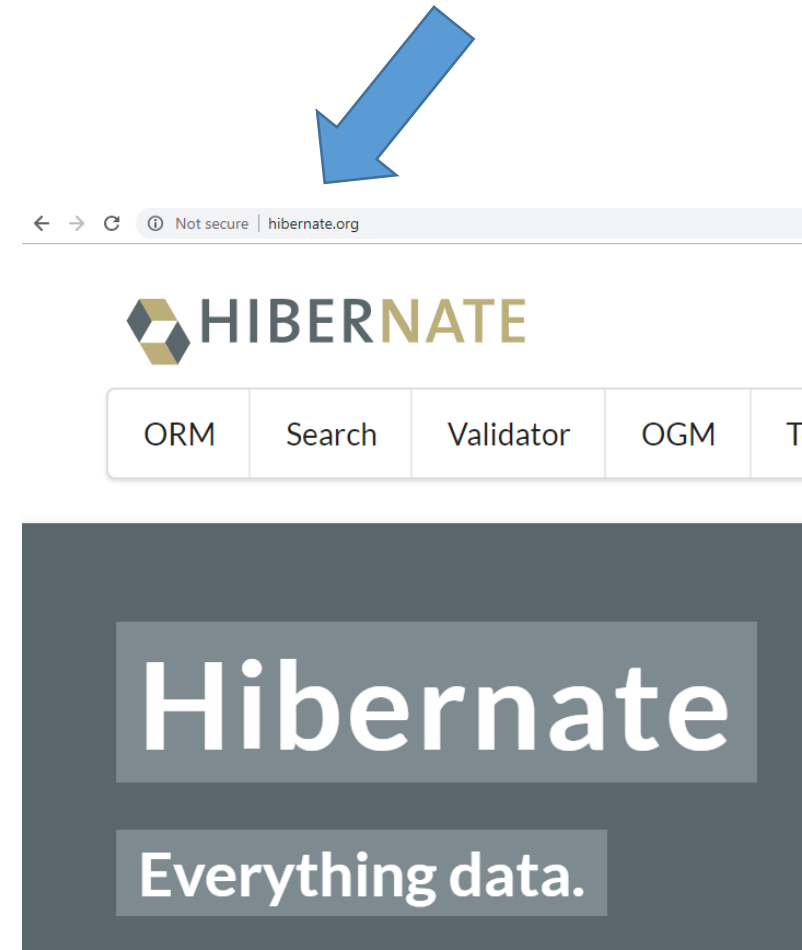
# Packages Adlandırmaları

- Teknik olarak, bir package istediğiniz gibi adlandırılabilir ancak çeşitli kuralları vardır
- Package prefix'i genellikle bir hostname'dir. Ör: *org.hibernate*
  - Ör, *com.*, *tr.*, *org.*, vs. gibi adlandırmalar projeyi tanımlar
- Prefix'ten sonra, isimlendirmeler anlamlı gruplandırmalarla yapılır
  - Ör *org.webp.intro.jee.jpa.entity.User01*, burada *org.webp* prefix'tir

# Kütüphane yayınlama: İsimlendirme

- Genellikle `<groupId>` projenin prefix'idir
- `<groupId>` adresinizle yayınlamak isterseniz, güvenlik sebebiyle (phishing saldırıları) Maven Central kendi domain adresinizi ister
  - Ör, `http://hibernate.org/`

```
<dependency>  
  <groupId>org.hibernate</groupId>  
  <artifactId>hibernate-validator</artifactId>  
</dependency>
```



# Properties \${}

**<properties>**

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
<fs>${file.separator}</fs>  
<version.java>1.8</version.java>  
<version.jacoco>0.7.9</version.jacoco>  
<version.javax.el>2.2.4</version.javax.el>  
<version.hibernate.jpa>1.0.0.Final</version.hibernate.jpa>  
<version.hibernate.core>5.2.9.Final</version.hibernate.core>  
<version.hibernate.validator>5.3.4.Final</version.hibernate.validator>  
<version.h2>1.4.194</version.h2>  
<version.postgres>42.1.4</version.postgres>  
<version.resteasy>3.1.3.Final</version.resteasy>  
<version.testcontainers>1.4.3</version.testcontainers>
```

**</properties>**

**<dependency>**

```
<groupId>javax.el</groupId>  
<artifactId>javax.el-api</artifactId>  
<version>${version.javax.el}</version>
```

**</dependency>**

**<dependency>**

```
<groupId>org.glassfish.web</groupId>  
<artifactId>javax.el</artifactId>  
<version>${version.javax.el}</version>
```

**</dependency>**

# Plugins

- Maven'in fonksiyonelliğini artırmak için kullanılır
- Plugin'ler 3. parti kütüphaneler gibi indirilip ayarlamalar yapılabilir
- Maven'in pek çok temel fonksiyonu plugin olarak adlandırılır
  - Ör, Java kodu derleme

```
<pluginManagement>  
  <plugins>  
    <plugin>  
      <groupId>org.apache.maven.plugins</groupId>  
      <artifactId>maven-compiler-plugin</artifactId>  
      <version>3.1</version>  
      <inherited>>true</inherited>  
      <configuration>  
        <source> ${version.java} </source>  
        <target> ${version.java} </target>  
      </configuration>  
    </plugin>  
  </plugins>  
</pluginManagement>
```

# Surefire ve Failsafe

- *Surefire*: *unit* test koşturmak için kullanılan plugin
  - Varsayılan olarak, *src/main/test* klasörü içindeki *\*Test.java* patterinine sahip bütün dosyalar testtir
  - **package** aşamasından önce testler çalıştırılır
- *Failsafe*: *integration* test'ler için plugin
  - Varsayılan olarak, *src/main/test* klasörü içindeki *\*IT.java* patternine sahip bütün dosyalar testtir.
  - Integration test'leri **package** aşamasından sonra çalıştırılır bu yüzden JAR/WAR dosyaları kullanılabilir
- Not: her iki durumda da, testler JUnit ile yazılır
- **UYARI**: Eğer *\*Test.java/\*IT.java* yanlış yazarsanız, testler maven tarafından çalıştırılmayacaktır



# Uygulamayı ilk defa çalıştırmak için

- Root klasörde: “**mvn clean install -DskipTests**”
  - Bütün modülleri build edecektir
  - **clean**: start yazarak başladığınızdan emin olun
  - **install**: Önceki bütün aşamaları (**compile** ve **package** dahil olmak üzere) çalıştırır
  - **-DskipTests**: testlerin çalışmasını engeller
  - UYARI: ilk defa çalıştırıldığında çok fazla kütüphane indireceği için biraz zaman almaktadır.
-

# JPA: Java Persistence API

---

# Object-Relational Mapping (ORM)

- SQL veritabanını (DB) map'lemek için kullanılır
- Programlarınızda, DB içerisindeki veriyi Java sınıfları ile temsil edeceksiniz
- JPA'nın amacı: DB'deki her bir tablo için *@Entity* sınıfı tasarlanır ve JPA framework read/write/update işlemlerini gerçekleştirir
- Teori: SQL'e ihtiyaç olmaması. Ancak bazı durumlarda (karmaşık sorgularda) ihtiyaç olabilir veya JPA implementasyonumuz garip sonuçlar verebilir

# Hibernate

- Bir JPA implementasyonudur
- Java'da en popüler olandır
  - WildFly gibi containerlarda varsayılandır
  - SpringBoot'ta varsayılandır
- Kütüphane olarak her türlü Java uygulamasında kullanılabilir
  - Ör, EE veya Spring olmasına gerek yok
- **src/main/resources/META-INF/persistence.xml**
  - JPA ayar dosyası



# Database Schema

- Verilen DB'de, Her bir tablo için *@Entity* sınıfının yazılması gerekir
  - Bir diğer seçenek: Önce *@Entity* sınıfını yaz ve veritabanı şeması ardından otomatik olarak oluşsun
    - Java'ya SQL'den daha aşina iseniz daha kolaydır
    - Prototipleme için iyidir
-

# Wrapper Objects

- *@Entity* sınıflarda *primitive* types denilen **int** ve **long** değerler kullanılmaz
- Onun yerine wrapped objects adı verilen *java.lang.\** paketinden gelen değerler kullanılır
  - Ör, **int** yerine **java.lang.Integer**
- Sebep: Veritabanı sütunları ele alındığında gelen değer **NULL** olabilir
  - Ancak primitive tipler'de bu olamaz



# Kütüphane olarak Hibernate

- Bu derste Hibernate'i kullanacağız
  - `start/commit/close transaction` komutlarını manuel olarak yapacağız
  - Ardından, Hibernate/JPA'yı JEE container (*WildFly*) içinde kullanacağız
  - H2 gibi embededed veritabanlarını kullanacağız
    - Test için kullanılan temel veritabanıdır
    - Daha sonra *Postgres'i* de göreceğiz
-

# Dersle İlgili Modüller

- *Not: pek çok açıklama yorum satırı olarak bulunmaktadır. Slaytlar içerisinde bulunmamaktadır.*
- **intro/jee-dependencies**
- **intro/jee/jpa/entity**
- **intro/jee/jpa/table**
- **intro/jee/jpa/embedded**
- **intro/jee/jpa/attribute**
- Ders 01 uygulaması