

# HIBERNATE



**Hibernate Java geliştiriciler için geliştirilmiş bir  
ORM**

**kütüphanesidir. Nesne yönelimli modellere göre**

**veritabanı ile olan ilişkiyi sağlayarak, veritabanı**

**üzerinde yapılan işlemleri kolaylaştırmakla  
birlikte**

**kurulan yapıyı da sağlamlaştırmaktadır.**

ORM Object Relational Mapping anlamına gelmektedir. ORM toolları nesne tabanlı programlamada bulunan objeler ile veritabanı sistemimizdeki tablolar arasında köprü görevi kurulmasını sağlar. Objeleri ilişkisel veritabanında mapping yapmaya yarar. ORM toollarının kullanımının en büyük avantajı OOP'da yer alan inheritance, polymorphism gibi konseptleri veritabanımız ile kolaylıkla kullanabilmektir.

Ayrıca bazı durumlarda projelerde SQL sorguları yazılımcıya büyük yük olabiliyor. Özellikle proje büyüdüğünde bu işlemler zorlaşıyor. ORM toollarından önce yazılımcılar kendileri objeleri veritabanındaki tablolarla eşleştirecek kodlar yazıyorlardı. Hibernate gibi toollar yazılımcıları bu yükten kurtardı.

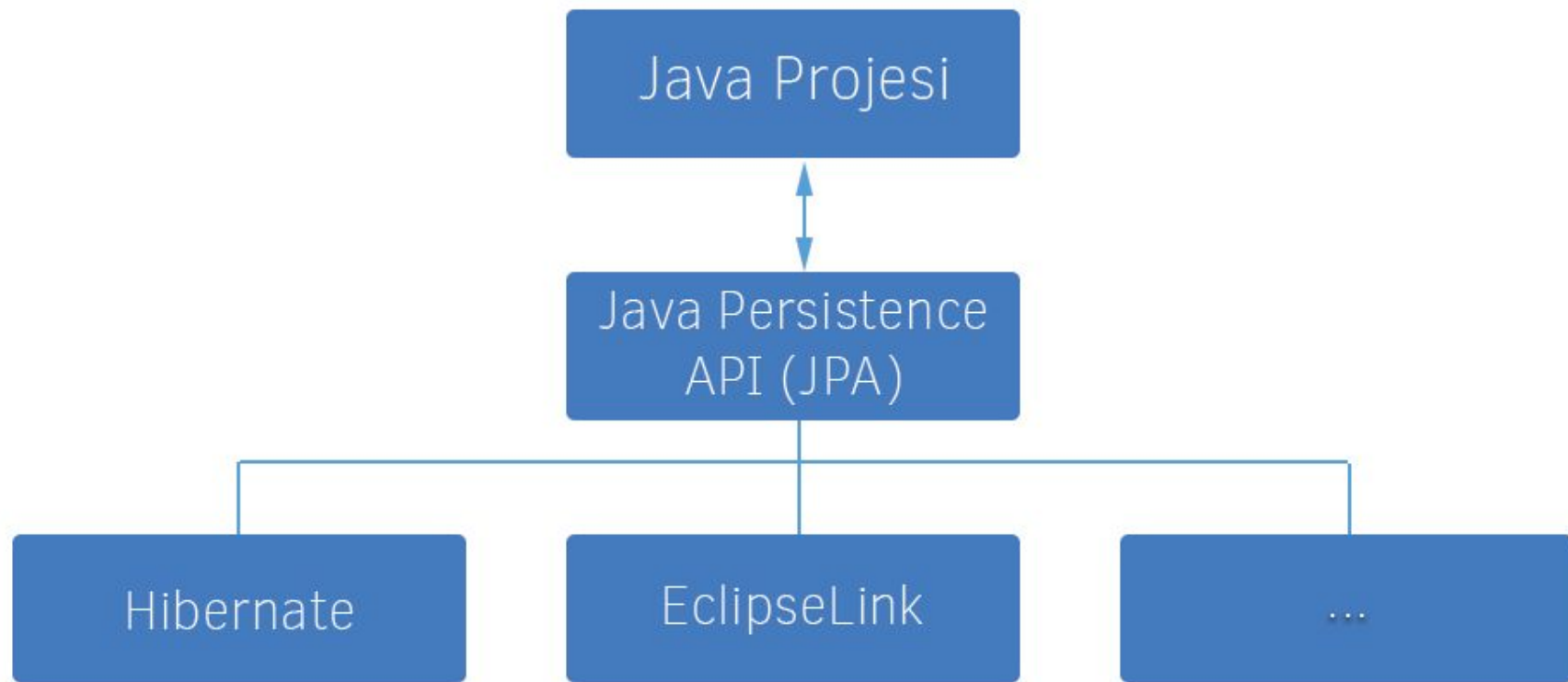
Burada bir not: ORM bu konseptin adıdır, bir araç değildir. Hibernate bir ORM aracıdır ve EclipseLink, OpenJPA, TopLink gibi birçok farklı ORM aracı bulunmaktadır. Hibernate, Java dünyasındaki popüler ORM araçlarından biridir

## JDBC

Java Database Connectivity (JDBC), Java projelerinizin veritabanı ile etkileşimini sağlayan, veritabanı işlemlerini SQL sorguları ile yapmanıza imkan veren API'dir. JDBC, Java Geliştirme Kiti'nin(JDK) 1997'den beri bir parçasıdır. Geliştiriciler Java projelerinde SQL sorguları yazmak istediğinde ilgili veritabanının driverını ve JDBC paketlerini ekleyerek projelerinde SQL sorgularını kullanabiliyorlar. Ancak gelişen süreçte JDBC bazı ihtiyaçları karşılayamaz hale geliyor ve farklı çözümler bulunuyor. JDBC, database ve java arasında aracıdır. Yalnız java da kod yazarken kullandığımız database in syntax ı kullanılır, mesela java daki OOP den de yararlanamayız, bu kod yazmayı zorlaştırır, kodu çoğaltır

## JPA

Java Persistent Api-> java kalıcılık interface i, bir translate diyebiliriz. .JPA bir yol haritası, bende metodlar var istediğin markayı kullanarak (hibernate) bu metodları kullanabilirsin.Hibernate, kullanılan database ile javayı birbirine bağladığı gibi, java kodlarını kullanarak kod yazarız, hangi database i kullandığımız önemli değildir. bu sayede veri tabanı değişse sıkıntı olmaz, hibernate (marka) bizim için halleder. hatta aynı projede hem sql hem mysql kullanabiliriz. diğer bir faydası cash sistemi



## Entity Nedir?

**Object Relational Mapping'de önemli kavramlardan birisi Entity'dir. Veritabanında kalıcı olacak nesneye Entity denir.**

**Nesnenin entity sayılabilmesi ve veritabanı tarafından tanınabilmesi için class tanımının hemen üstünde “@Entity” annotation ile tanımlanmalıdır. Entity sınıfımız bir Java POJO sınıfıdır. Entity annotation ile tanımlanmış sınıf inner class olamaz ya da final olarak tanımlanamaz.**

**Özetle bir Entity, veritabanımızda yer alan tablomuzdaki bir satıra denk geliyor. Veritabanımızdaki satırların tamamı da entity set oluyor. Önceki yazımızdaki Entity class örneği aşağıda yer alıyor. '@' ile başlayan ifadeler Annotationlar olarak adlandırılıyor. Annotationlar hakkında daha detaylı bilgiler ileriki yazılarda yer alacak.**

```
@Entity
@Table(name = "Students")
public class Student {

    @Id
    @Column(name = "id")
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

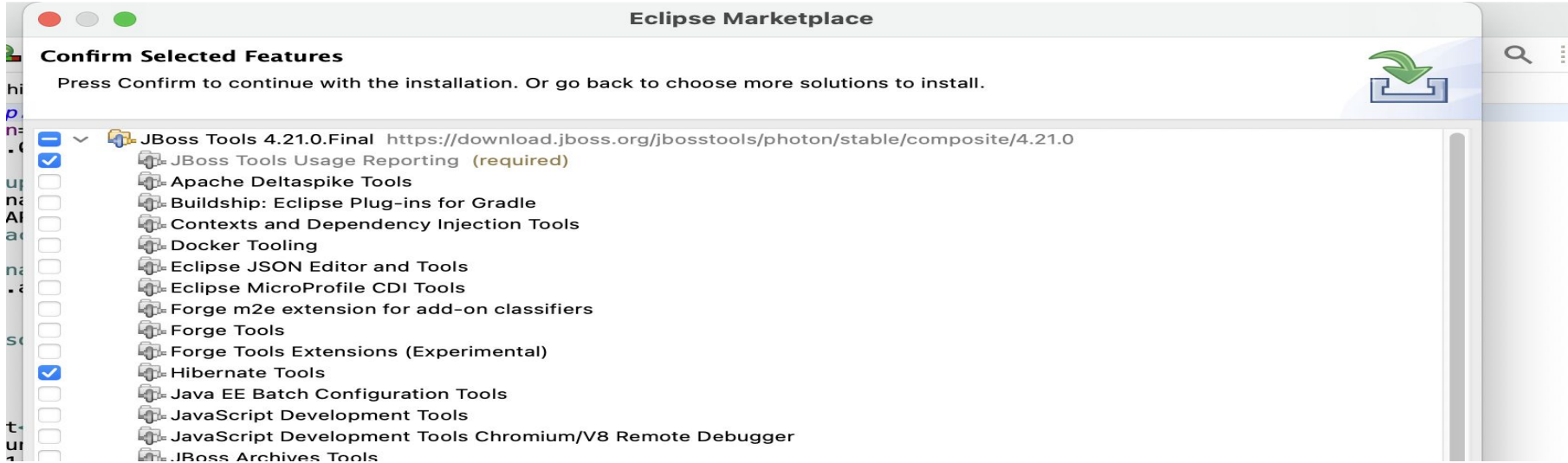
    @Column(name = "email")
    private String email;
```



id	first_name	last_name	email
1	Will	Smith	will@test.com
2	Leonardo	DiCaprio	leonardo@test.com
NULL	NULL	NULL	NULL

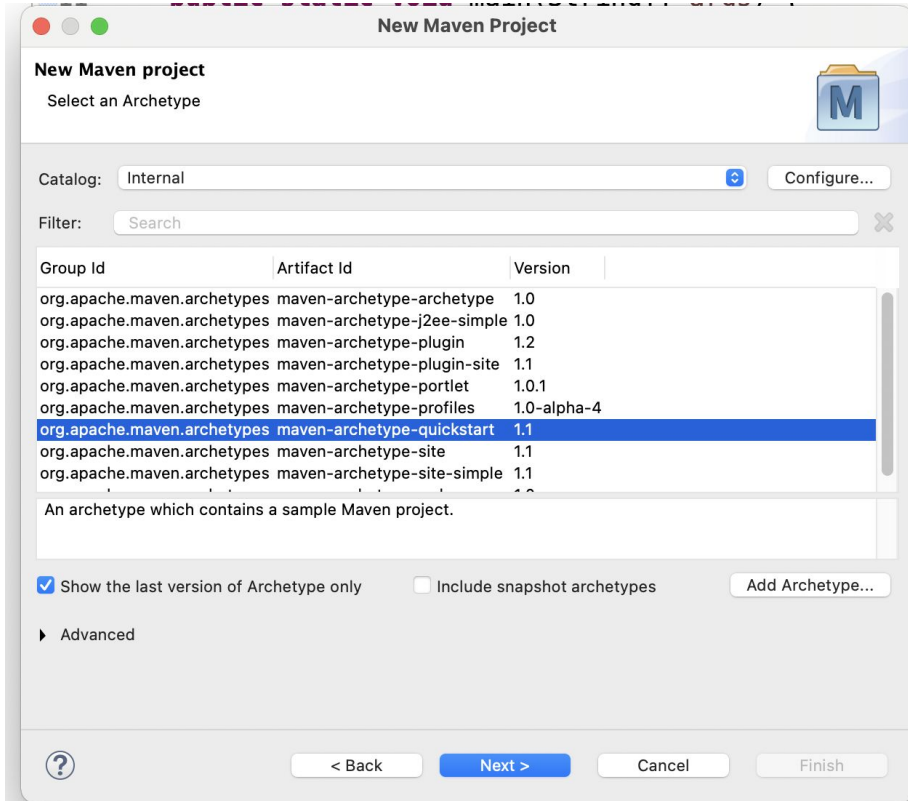
# Projeye Hibernate'in Dahil Edilmesi

1-Help-> Eclipse marketplace den jbos hibernate install ediyoruz  
ve sadece alttakileri seçip confirm



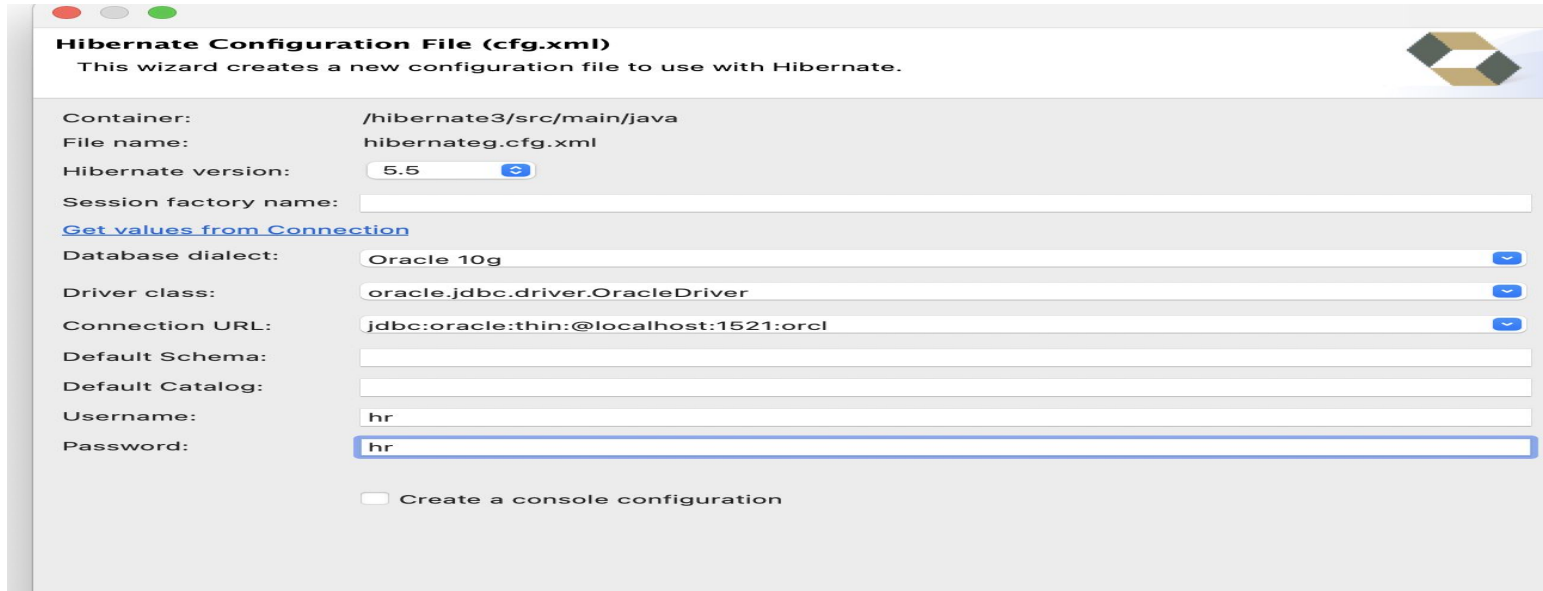


**2-**Öncelikle basit bir Maven projesi oluşturuyoruz. File->new-> project->maven project->next->next



# 3-configuration dosyası

projeye sağ tıklatıp, new-> other->hibernate->hibernate conf. file (cfg.xml)->next->next.. ardından resimdeki dolu satırları kendi database bilgilerimizle dolduruyoruz



**Hibernate Configuration File (cfg.xml)**

This wizard creates a new configuration file to use with Hibernate.

Container: /hibernate3/src/main/java

File name: hibernateg.cfg.xml

Hibernate version: 5.5

Session factory name:

[Get values from Connection](#)

Database dialect: Oracle 10g

Driver class: oracle.jdbc.driver.OracleDriver

Connection URL: jdbc:oracle:thin:@localhost:1521:orcl

Default Schema:

Default Catalog:

Username: hr

Password: hr

☐ Create a console configuration

# cfg.xml e alttakileri de ekliyoruz

```
<property name="hbm2ddl.auto">update</property>  
  <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>  
  <property name="show_sql">true</property>  
  <property name="format_sql">true</property>
```

# Kurulumdaki property lerin nedeni:

Konfigürasyon dosyası içerisinde tanımlamasakta Hibernate içerisinde birçok property default değere sahiptir. Yani onları değiştirmek yazılımcının insiyatifine kalmıştır. Ancak JDBC driver class, JDBC Url, database username/password gibi propertyler zorunludur. Hibernate'in hangi driver ile, hangi veritabanına, hangi kullanıcı ile bağlanacağı gibi bilgileri bilmesi gerekmektedir. Bu sebeple de biz bunları konfigürasyon dosyası içerisinde tanımlıyoruz. Konfigürasyon dosyamızı src/ dizini altında oluşturuyoruz. Tanımlamaların ne anlama geldiğini tek tek inceleyeceğiz.

hibernate.connection.driver\_class : JDBC driver classının yerini tanımlar.

hibernate.connection.url : Veritabanımızın çalıştığı url bilgisini barındırır. Burada veritabanının çalıştığı portu doğru yazmak önemlidir.

hibernate.connection.username : Veritabanına erişimi olan kullanıcının adı.

hibernate.connection.password : Veritabanına erişimi olan kullanıcının parolası

hbm2ddl.auto : Veritabanına bağlantı yapıldığında var olan veritabanı üzerine mi devam edilsin, yoksa uygulama her çalıştığında baştan mı veritabanı oluşturması için ayar. Yalnızca üzerine devam etmesi için "update", baştan oluşturması için "create" belirlememiz gerekiyor.

hibernate.dialect : Bu property belirtilen veritabanı için SQL sorgularının Hibernate tarafından oluşturulması için gereklidir.

show\_sql : Console ekranında Hibernate tarafından oluşturulan SQL sorgularını text olarak görmek isterseniz "true" olarak atamanız gerekmektedir.

## 4- pom.xml dosyası için mvn.repository adresinden alttaki dependency leri yükleyip sağtık->maven update ediyoruz

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>techproed</groupId>
7   <artifactId>hibernate</artifactId>
8   <version>0.0.1-SNAPSHOT</version>
9   <packaging>jar</packaging>
10
11   <name>hibernate</name>
12   <url>http://maven.apache.org</url>
13
14   <properties>
15     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16   </properties>
17
18   <dependencies>
19
20     <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
21     <dependency>
22       <groupId>org.hibernate</groupId>
23       <artifactId>hibernate-core</artifactId>
24       <version>5.5.2.Final</version>
25     </dependency>
26
27     <!-- https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc8 -->
28     <dependency>
29       <groupId>mysql</groupId>
30       <artifactId>mysql-connector-java</artifactId>
31       <version>8.0.26</version>
32     </dependency>
33
34     <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-ehcache -->
35     <dependency>
36       <groupId>org.hibernate</groupId>
37       <artifactId>hibernate-ehcache</artifactId>
38       <version>5.5.3.Final</version>
39     </dependency>
40
41   </dependencies>
42 </project>
```

## 5-database java bağlantısı için run class ına alttaki işlemleri yapıyoruz

```
public class H02_Save {  
    public static void main(String[] args) {  
        //İPORTLARI HİBERNATE OLANDAN yap  
        // Veritabanı bağlantı ayarlarını Hibernate'e göstermeliyiz.hibernate den import et  
        Configuration con = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(H01_Sehir.class);  
        //        //üstteki url lere göre hibernate i (şifre isim vs) ayarla demek//üstteki class ı göz önüne al  
        //  
        // con objesinden bir oturum grubu oluşturduk.session açıyoruz işlemiz bitince kapatıyoruz en altta  
        SessionFactory sf = con.buildSessionFactory();//tüm session ların anası, parent ı, bunun altına bir sürü session oluşturabilir  
  
        // Oturum grubundan bir oturumu başlattık.  
        Session session = sf.openSession();  
  
        // Acılan session içerisinde işlemlere başlayabilmek için Transaction acıyoruz.  
        /*Transactionlar ile bir işlem yarıda kaldıysa veya  
        * tam olarak tamamlanadıysa tüm adımlar başa alınır  
        * veri ve işlem güvenliği için önemlidir.Kısacası ya hep  
        * ya hiç prensibine göre çalışır  
        */  
        Transaction tx = session.beginTransaction();  
  
        // ilk burayı yaz,sonra üstleri  
        // Tabloya eklenecek verileri (record, kayıt) oluşturmamız gerekiyor.her run ettiğimizde id ler değiştirilmeli çünkü önceki ka  
        H01_Sehir sehir1 = new H01_Sehir(49, "Istanbul");  
        H1_Sehir sehir2 = new H1_Sehir(38,"Izmir", 2500000);  
  
        // Veritabanına kayıtların eklenmesi.(kalıcı persistion) bu=> yaz demek, commit=> yazmayı kesinleştiriyor  
        session.save(sehir1);  
        //session.save(new H1_Sehir(38, "Istanbul", 10000000));//insert into gibi  
        session.save(new H01_Sehir(41,"Izmir"));  
  
        // İşlemlerin veritabanına aktarılması(yazmakta fayda var)  
        tx.commit();  
    }  
}
```