



Object-Relational Mapping (ORM) - Hibernate

Αθ. Ανδρούτσος



Μεθοδολογίες Υλοποίησης ΒΔ

Hibernate

- Η σύνδεση μίας εφαρμογής με τη ΒΔ σε στρωματοποιημένες αρχιτεκτονικές τυπικά υλοποιείται με την αντιστοίχιση του **Model** της εφαρμογής με το **Schema** της ΒΔ
- Το βασικό πρόβλημα στην υλοποίηση της αντιστοίχισης είναι ότι το **Model** είναι γραμμένο σε μία αντικειμενοστραφή γλώσσα προγραμματισμού ενώ το **DB Schema** είναι γραμμένο σε μία σχεσιακή ΒΔ



The Paradigm Mismatch (1)

Hibernate

- Αυτή η **ετερογένεια** μεταξύ: 1) από τη μια πλευρά του ***Object-Oriented Model*** με κλάσεις, πεδία κλάσεων, συγκεκριμένους τύπους δεδομένων, κληρονομικότητα και σχέσεις μεταξύ κλάσεων, και 2) από την άλλη πλευρά του ***Relational Database Model*** με πίνακες, ιδιότητες πινάκων και σχέσεις μεταξύ πινάκων, είναι γνωστή ως **Paradigm Mismatch**



The Paradigm Mismatch (2)

Hibernate

- Στόχος είναι να γεφυρώσουμε το χάσμα και να αντιστοιχίσουμε (map) το Object-Oriented Μοντέλο με το Σχεσιακό Μοντέλο (**Object-Relational Mapping – ORM**)
- Τυπικά αυτό μπορεί να γίνει με δύο τρόπους: 1) Ξεκινώντας από το Domain Model και αντιστοιχώντας το στο Schema της ΒΔ (Model First ή Code First), και 2) Ξεκινώντας από τη ΒΔ και αντιστοιχώντας τη στο Domain Model (Database First)



Model First

Hibernate

- Ιδανικά, ιδιαίτερα σε νέες εφαρμογές, είναι πιο εύκολο να ξεκινάμε με το Domain Model και στη συνέχεια με κάποιο framework (όπως το Hibernate που θα αναλύσουμε στο παρόν κεφάλαιο) να δημιουργείται αυτόματα το σχήμα της ΒΔ
- Η προσέγγιση αυτή ονομάζεται **Model-First** ή **Code-First**



Database-First (1)

Hibernate

- Σε υπάρχουσες εφαρμογές όπου η ΒΔ έχει δημιουργηθεί ήδη, μπορούμε να κάνουμε το αντίθετο, να δημιουργήσουμε δηλαδή αυτόματα το Domain Model με τη χρήση κάποιου framework όπως το Hibernate
- Αυτή η προσέγγιση ονομάζεται Database-First



Database-First (2)

Hibernate

- Σε καινούργιες εφαρμογές, η προσέγγιση DB First έχει μειονεκτήματα:
 - Οι προγραμματιστές θα πρέπει να ξέρουν SQL κάτι το οποίο είναι μεν επιθυμητό, αλλά στο επίπεδο της ανάπτυξης εφαρμογών δεν θα πρέπει να αποτελεί περιορισμό
 - Η εφαρμογή δεν είναι portable, δηλαδή δεν μπορούμε εύκολα να αλλάξουμε ΒΔ και να μεταφέρουμε την εφαρμογή μας σε άλλη ΒΔ



Full Stack Εφαρμογές

Hibernate

- Συνοπτικά, μια Full-Stack εφαρμογή περιλαμβάνει δύο διαφορετικά μοντέλα σχεδιασμού και ανάπτυξης δεδομένων, που πρέπει να γεφυρωθούν:
 1. Το **Σχεσιακό μοντέλο** για τη Βάση Δεδομένων
 2. Το **Αντικειμενοστραφές μοντέλο**, για το Domain Model της εφαρμογής



ORM (Object-Relational Mapping)

Hibernate

- Η γεφύρωση των δύο διαφορετικών μοντέλων δεδομένων, ονομάζεται **ORM - Object-Relational Mapping**
- Η **αντιστοίχιση** μεταξύ του Αντικειμενοστραφούς Μοντέλου και των Σχεσιακών Βάσεων περιλαμβάνει την αντιστοίχιση:
 - Κλάσεων (model) σε πίνακες,
 - Πεδίων των κλάσεων σε πεδία πινάκων και
 - Σχέσεων μεταξύ των κλάσεων σε σχέσεις μεταξύ των πινάκων



ORM – Model First

Hibernate

- Στην Model First προσέγγιση, για να μπορέσουμε να υλοποιήσουμε το mapping, χρειαζόμαστε ένα API ώστε να μπορούμε στα Java Beans του Model να ορίσουμε το πως θα γίνει το mapping, δηλαδή πως ακριβώς θα αντιστοιχηθούν τα Java Beans σε πίνακες τα πεδία του Java Bean σε πεδία πινάκων καθώς και οι σχέσεις μεταξύ των Java Beans σε σχέσεις πινάκων της ΒΔ



JPA (1)

- Αυτή τη δυνατότητα μας την παρέχει το **Java Persistence API (JPA)** που είναι ένα Java API (προδιαγραφές για την αντιστοίχιση από Domain Model σε Database Model) δηλαδή μία βιβλιοθήκη με κλάσεις, interfaces και annotations
- Το **JPA** είναι μέρος της Java/Jakarta EE. Το 2017 έγινε release η έκδοση **JPA 2.2** που είναι μέρος της Java EE 8 (συμβατό με Tomcat 9)
- Μετονομάστηκε σε Jakarta Persistence το 2019 και εκδόθηκε το version JPA 3.0 το 2020 και **JPA 3.1 το 2022 ως μέρος της Jakarta EE 10**



JPA (2)

- Το JPA ορίζει:
 - ένα API (**Annotations**) για να περιγράψουμε σε Java, Domain Models, τα οποία στη συνέχεια θα μετατρέπονται αυτόματα σε Database Schema,
 - καθώς και ένα άλλο API (**interfaces και κλάσεις**) για να γράφουμε SQL DML (CRUD εντολές) σε Java, που μετατρέπονται αυτόματα σε SQL εντολές



JPA (3)

- Το Java Persistence API (JPA) είναι ένα σύνολο προδιαγραφών, interfaces και κλάσεων που παρέχουν τη δυνατότητα μόνιμης αποθήκευσης των δεδομένων του Model σε σχεσιακές βάσεις δεδομένων
- **Το JPA είναι προδιαγραφές** που διάφοροι vendors υλοποιούν, όπως **Hibernate**, Eclipse Link, Toplink, Spring Data JPA, κλπ.
- <https://javaee.github.io/tutorial/persistence-intro.html>



JPA(3)

- Το Java Persistence API ορίζεται στο package **javax.persistence**. Από το Νοέμβριο 2020 άλλαξε το namespace από **javax.*** σε **jakarta.*** (επομένως, **jakarta.persistence**)
- Στο JPA περιλαμβάνονται:
 - Interfaces, Classes, Enums, Exceptions
 - Annotations που μπορούμε να εισάγουμε στις Java κλάσεις για να γίνει αυτόματα το ORM mapping
 - Query APIs για ερωτήσεις στο domain model που μεταφράζονται αυτόματα CRUD και SQL Queries



Hibernate

Hibernate

- Το **JPA** είναι **specs** (specifications). Το **Hibernate** παρέχει μία **open-source** λύση που υλοποιεί το **Java Persistence API** και ταυτόχρονα παρέχει αυτή τη λύση με τρόπο ευέλικτο και επεκτάσιμο
- Ο Gavin King δημιούργησε το Hibernate project που είναι στην έκδοση 6.6.* (Αυγ. 2024). Η **έκδοση 6.6** που υλοποιεί το **JPA 3.1** που είναι μέρος της Jakarta 10 και είναι συμβατή με Java 11, 17, 21. Πρόκειται για ένα από τα πιο χρησιμοποιούμενα εργαλεία των προγραμματιστών Java

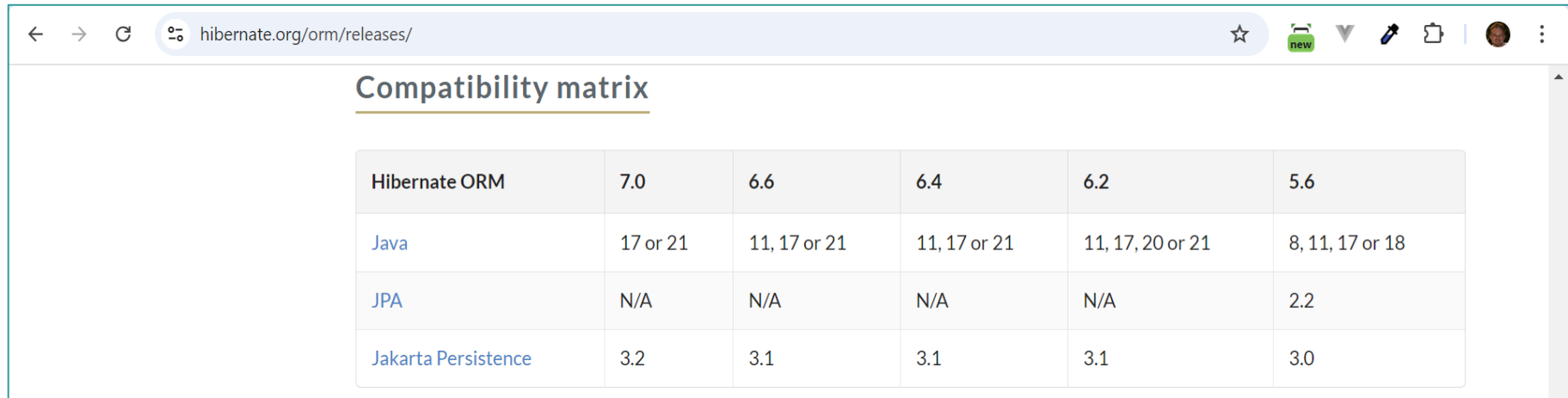
The screenshot shows a web browser window with the URL `hibernate.org/orm/releases/6.6/`. The page has a sidebar with links: About, Releases (selected), Documentation, and Migration guides. The main content area is titled "Compatibility" and contains a table with the following data:

Java	11, 17 or 21
Jakarta Persistence	3.1



Versions compatibility

Hibernate



The screenshot shows a web browser window with the URL hibernate.org/orm/releases/. The page title is "Compatibility matrix". Below the title is a table showing the compatibility of Hibernate ORM versions with Java, JPA, and Jakarta Persistence.

Hibernate ORM	7.0	6.6	6.4	6.2	5.6
Java	17 or 21	11, 17 or 21	11, 17 or 21	11, 17, 20 or 21	8, 11, 17 or 18
JPA	N/A	N/A	N/A	N/A	2.2
Jakarta Persistence	3.2	3.1	3.1	3.1	3.0

- <https://hibernate.org/orm/releases/>



DAO Layer

Hibernate

- Στη συνέχεια θα δούμε θέματα μόνιμης αποθήκευσης δεδομένων σε εφαρμογές Java/Jakarta EE που περιλαμβάνουν ένα Domain Model
- Αντί να υλοποιούμε το DAO με SQL, θα υλοποιούμε με JPA έχοντας ως Persistence Provider, το Hibernate



Εισαγωγικό Παράδειγμα (1)

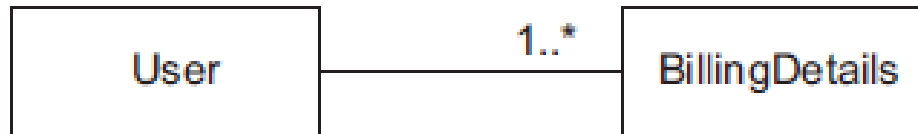
Hibernate

- Έστω ότι έχουμε να σχεδιάσουμε και να υλοποιήσουμε μία online εφαρμογή e-shop
- Χρειαζόμαστε μία κλάση που να αναπαριστά τα στοιχεία **User** και άλλη μία κλάση που να αναπαριστά πληροφορίες πληρωμής, έστω **BillingDetails**



Εισαγωγικό Παράδειγμα (2)

Hibernate



- Όπως βλέπουμε στο Domain Model που αναπαρίσταται ως διάγραμμα UML κάθε instance της κλάσης **User** σχετίζεται με πολλά instances της κλάσης **BillingDetails**, ενώ κάθε instance της BillingDetails σχετίζεται με ένα instance της κλάσης User δηλαδή είναι μια σχέση Ένα-προς-Πολλά (1-N) από την κλάση User προς την BillingDetails
- Όταν θα πάμε να υλοποιήσουμε σε Java το Domain Model, θα πρέπει να ορίσουμε για κάθε κλάση (ή αλλιώς Entity στην ορολογία JPA), τα παρακάτω:
 - Τον **identifier** κάθε κλάσης που μοναδικοποιεί κάθε instance, όπως στη ΒΔ έχουμε το πρωτεύον κλειδί
 - Τα **persistence fields**, τα πεδία δηλαδή της κλάσης που θέλουμε να αποθηκεύονται στη ΒΔ, όπως στη ΒΔ έχουμε τα πεδία των πινάκων
 - Τη **σχέση** μεταξύ των δύο Entities, όπως θα κάναμε και στη ΒΔ με τα ξένα κλειδιά



Εισαγωγικό Παράδειγμα (3)

Hibernate

```
1 package gr.aueb.thanos.model;
2
3 import java.util.Set;
4
5 import javax.persistence.Entity;
6 import javax.persistence.Id;
7
8 @Entity
9 public class User {
10     @Id
11     Long id;
12     String username;
13     String address;
14     Set<BillingDetails> billingDetails;
15
16     // getters, setters, other methods
17 }
```

```
1 package gr.aueb.thanos.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5
6 @Entity
7 public class BillingDetails {
8     @Id
9     Long id;
10    String account;
11    String bankName;
12    User user;
13
14    // getters, setters, other methods
15 }
```

- Το βασικό annotation που ορίζει μία κλάση ως persistence entity είναι το **@Entity**
- Κάθε Entity πρέπει να έχει ένα μοναδικό identifier (που αντιστοιχεί στο πρωτεύον κλειδί) και ορίζεται με το annotation **@Id**
- Έχουμε παραλείψει τις υλοποιήσεις των properties accessors (getters/setters) & business methods για εκπαιδευτικούς λόγους γιατί εστιάζουμε στο state των κλάσεων σε σχέση με την μόνιμη αποθήκευσή τους (persistence)
- Τα **persistent πεδία** είναι τα πεδία των κλάσεων, που όπως θα δούμε μπορούμε να κάνουμε tag με το Annotation **@Column** διαφορετικά αποδίδονται από το Hibernate default τιμές (π.χ. το όνομα των αντίστοιχων πεδίων στη ΒΔ θα έχουμε τα ίδια ονόματα με τα πεδία των κλάσεων)



Εισαγωγικό Παράδειγμα (4)

Hibernate

```
1 package gr.aueb.thanos.model;
2
3 import java.util.Set;
4
5 import javax.persistence.Entity;
6 import javax.persistence.Id;
7
8 @Entity
9 public class User {
10     @Id
11     Long id;
12     String username;
13     String address;
14     Set<BillingDetails> billingDetails;
15
16     // getters, setters, other methods
17 }
```

```
1 package gr.aueb.thanos.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5
6 @Entity
7 public class BillingDetails {
8     @Id
9     Long id;
10    String account;
11    String bankName;
12    User user;
13
14    // getters, setters, other methods
15 }
```

- Όσο αφορά τη σχέση μεταξύ των δύο κλάσεων θα πρέπει να μπορούμε να διασχίσουμε τη σχέση και προς τις δύο κατευθύνσεις (διαφορετικά είναι κοστοβόρα τα queries):
 1. Από τον User στο BillingDetails μέσω του *Set<BillingDetails>* και να πάμε σε ένα ή περισσότερα instances του BillingDetails, και
 2. από την άλλη πλευρά της σχέσης, δηλ. από το BillingDetails να πάμε στον user μέσω του πεδίου User
- Τα annotations που χρησιμοποιούμε για την περιγραφή σχέσεων θα τα δούμε εκτενώς τη συνέχεια



Εισαγωγικό Παράδειγμα (5)

Hibernate

- Το SQL Schema που δημιουργείται αυτόματα από το Hibernate για τις προηγούμενες κλάσεις είναι το παρακάτω:

```
1 • create table USER (  
2     ID bigint not null primary key,  
3     USERNAME varchar(255),  
4     ADDRESS varchar(255)  
5 );
```

```
8 • create table BILLINGDETAILS (  
9     ID bigint not null primary key,  
10    ACCOUNT varchar(255),  
11    BANKNAME varchar(255),  
12    USER_ID bigint not null,  
13    foreign key (USER_ID) references USER(ID)  
14 );
```

Εδώ η σχέση ορίζεται από το foreign key constraint που βρίσκεται στο μέρος του 'προς-πολλά' (δηλαδή στον πίνακα BILLINGDETAILS)



Domain Model Mapping

Hibernate

- Ιστορικά το Hibernate χρησιμοποιούσε ένα XML αρχείο για να ορίζει το mapping μεταξύ Domain Classes και Πινάκων της ΒΔ
- Με τον ερχομό του JPA μπορούμε να εκφράσουμε το mapping με **annotations** στις ίδιες τις κλάσεις κάνοντας το μοντέλο μας portable και σε άλλα ORM εκτός του Hibernate



@Entity

Hibernate

- Το **@Entity** (*javax.persistence.Entity*) annotation είναι το βασικό JPA annotation για να ορίσουμε μία κλάση ότι θέλουμε να είναι Persistent
- Επομένως στο Domain Model μιλάμε για **Entities**, δηλαδή κλάσεις που έχουν σημανθεί με το **@Entity** annotation
- Ένα Entity είναι ένα lightweight persistence domain object (JavaBean) και τυπικά αναπαριστά ένα πίνακα της ΒΔ



Entities (1)

- Ένα Entity είναι κατά βάση ένα *Lightweight persistence domain object*, δηλαδή ένα POJO/JavaBean Class:
 - Η κλάση πρέπει να έχει ένα public ή protected default (no-argument) constructor (και πιθανά και άλλους constructors)
 - Οι ονομασίες των getters και setters ακολουθούν το JavaBeans-style δηλαδή αν έχουμε ένα πεδίο με όνομα property τότε ο getter είναι ο *getProperty* και ο setter ο *setProperty*. Αν το πεδίο είναι boolean μπορεί ο getter να είναι και *isProperty*



Entities (2)

- Τυπικά ένα **Entity** είναι ένα abstraction δηλαδή μία **αναπαράσταση ενός πίνακα** σε μία σχεσιακή βάση δεδομένων και **κάθε instance του Entity αντιστοιχεί σε μία γραμμή (πλειάδα, tuple) του πίνακα**
- Τόσο το ίδιο το Entity, όσο και τα πεδία καθώς οι σχέσεις μεταξύ των Entities του Domain Model χρησιμοποιούν **annotations** για να αντιστοιχηθούν σε πίνακες, πεδία πινάκων και σχέσεις πινάκων στην σχεσιακή ΒΔ
- Το *persistent state* του Entity περιλαμβάνει τα persistent πεδία (Όλα τα πεδία που δεν είναι annotated ως Transient)



Entities (3)

- Τα Entities **δεν πρέπει να είναι final**. Και οι μέθοδοι καθώς και τα πεδία που θέλουμε να κάνουμε persist δεν πρέπει να είναι final, ώστε να μπορούν να γίνουν subclass και να μπορούν να γίνουν lazy-loaded
- Αν τα instances ενός Entity αποθηκεύονται σε ένα HTTP Session object πρέπει το Entity να κάνει implements το `java.io.Serializable`



Persistent fields (1)

Hibernate

- Ο τύπος δεδομένων των persistent πεδίων της κλάσης μπορεί να είναι:
 - Primitives
 - **java.lang.String**
 - **Wrappers των primitive types (Long, Integer, κλπ.)**
 - java.math.BigInteger
 - java.math.BigDecimal
 - java.util.Date
 - java.time.LocalDate
 - java.time.LocalDateTime
 - java.util.Calendar
 - java.sql.Date
 - java.sql.Time
 - java.sql.Timestamp
 - User-defined serializable types
 - byte[]
 - Byte[]
 - char[]
 - Character[]
 - Enumerated types
 - **Άλλα entities και/ή Collections των entities**
 - Embeddable classes



Persistent fields (2)

Hibernate

- Συνήθως ωστόσο δεν χρησιμοποιούμε primitives αλλά **κλάσεις** για να μπορούμε να ελέγχουμε για **nulls**, μιας και το null είναι η default τιμή των instances
- Όλα τα πεδία που δεν έχουν γίνει annotate ως `javax.persistence.Transient` (`@Transient`) θα αποθηκευτούν στη ΒΔ



Persistent fields (2)

Συλλογές ως πεδία

Hibernate

- Τα *collection-valued* πεδία των Entities θα πρέπει να είναι όχι απλές κλάσεις, αλλά **Java Collection Interfaces**:
 - `Java.util.Collection`
 - `Java.util.Set`
 - `Java.util.List`
 - `Java.util.Map`
- Μπορούμε να χρησιμοποιούμε Generics



Primary Keys στα Entities

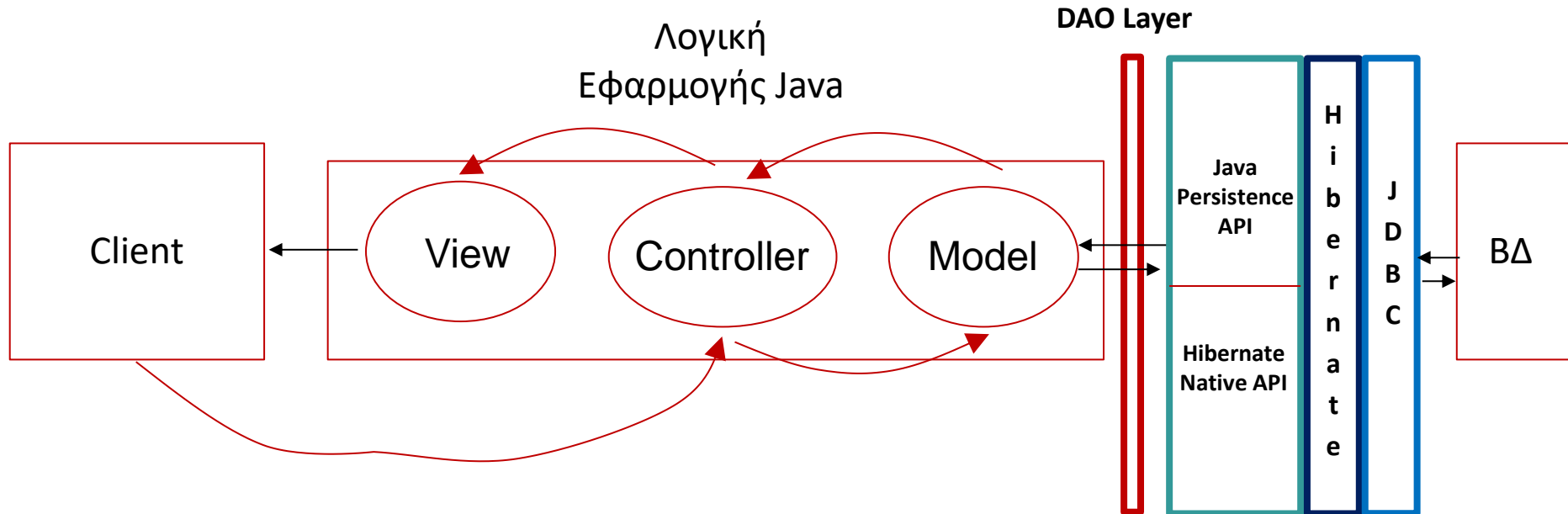
Hibernate

- Κάθε Entity πρέπει να έχει ένα **unique object identifier**
- Η *primary key class* πρέπει να υλοποιεί την *hashCode* και *equals* ώστε να μπορούμε σίγουρα να εκφράσουμε την ισότητα μιας και στη ΒΔ έχουμε ισότητα τιμών (*equals*) και όχι ισότητα δεικτών (*references*)



Αρχιτεκτονική (1)

Hibernate

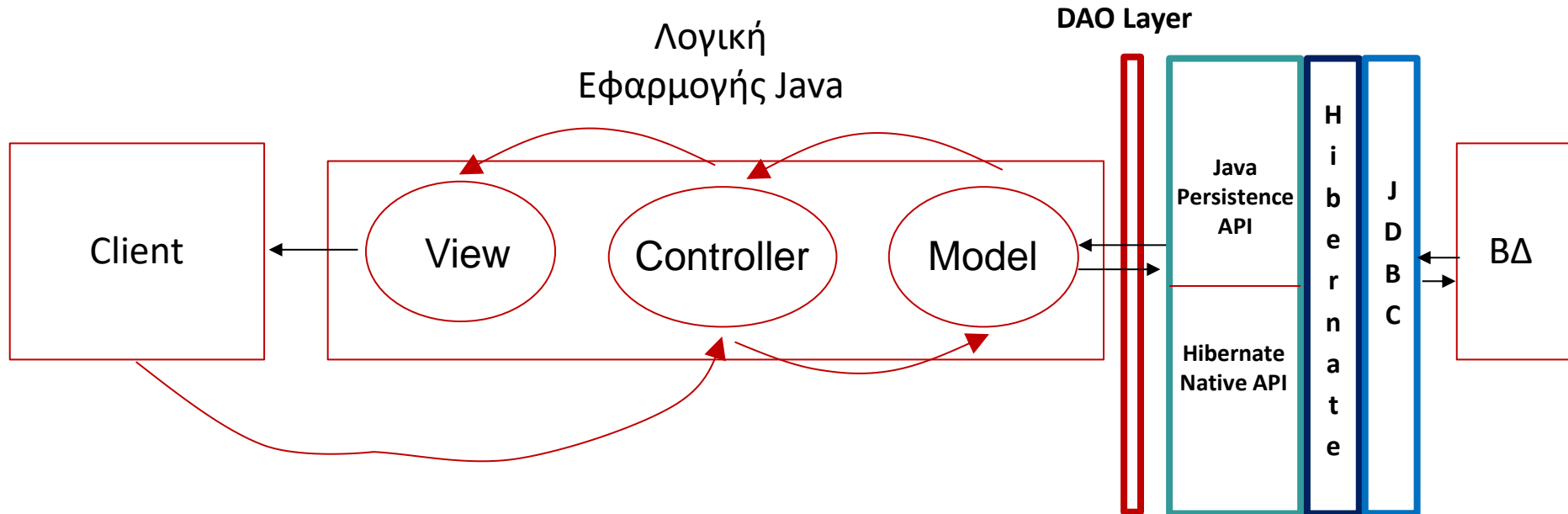


- Το Hibernate είναι ένα επίπεδο μεταξύ του DAO και της Βάσης Δεδομένων
- Η εφαρμογή Java μπορεί να χρησιμοποιεί τα δύο Hibernate APIs (κυρίως το JPA) για να διαβάσει, αποθηκεύσει και πραγματοποιήσει queries στο domain model της εφαρμογής



Αρχιτεκτονική (2)

Hibernate



- Το Hibernate υλοποιεί το **JPA** αλλά υλοποιεί και ένα δικό του **Hibernate Native API**
- Αν θέλουμε η εφαρμογή μας να μπορεί να είναι portable μεταξύ διαφορετικών ORM τότε θα πρέπει να χρησιμοποιούμε το JPA API



- Η σουίτα προγραμμάτων του Hibernate περιλαμβάνει τα ακόλουθα:
 - Hibernate ORM – Hibernate Native API
 - Hibernate EntityManager – **Υλοποίηση του JPA**
 - Hibernate Validator
 - Hibernate Envers
 - Hibernate Search
 - Hibernate OGM



First Level Cache

Hibernate

- Για να καταλάβουμε καλύτερα πως λειτουργεί το JPA θα πρέπει να καταλάβουμε την έννοια της **Cache**, ενός ενδιάμεσου buffer μεταξύ της εφαρμογής και της ΒΔ
- Όλες οι persistence λειτουργίες της εφαρμογής δεν εκτελούνται κατευθείαν στη ΒΔ αλλά πρώτα εκτελούνται στην Cache (**First-Level Cache**)
- Πρακτικά η First-Level Cache στο JPA υλοποιείται με ένα map, που σχετίζει τους identifiers των entities με τα instances των entities



Flush/Commit

- Μόνο όταν εκτελούνται οι μέθοδοι **flush** ή **commit** όλες οι αλλαγές που έχουν γίνει στην cache εκτελούνται στη ΒΔ
- Για την ακρίβεια, η μέθοδος **flush** του **EntityManager** (ο **EntityManager** είναι το βασικό interface που διαχειρίζεται την **1st-level cache**, όπως θα δούμε στη συνέχεια) συγχρονίζει την cache με τη ΒΔ δημιουργώντας τις αντίστοιχες SQL εντολές, αλλά δεν τις εκτελεί, οπότε αν δημιουργηθεί Exception κατά τη διάρκεια ενός flush μπορούμε να κάνουμε rollback και να ανακληθούν όλες οι αλλαγές που τυχόν έχουν γίνει sync στη ΒΔ
- Το commit περιλαμβάνει -πριν το commit μία εντολή flush (που όπως είπαμε κάνει sync την cache με τη ΒΔ χωρίς να κάνει commit). Αν συμβεί κάτι λοιπόν μπορεί να γίνει rollback το transaction, αλλιώς γίνεται commit και μετά δεν μπορεί να γίνει rollback



PersistenceContext (1)

Hibernate

- Το **PersistenceContext** είναι ένα όρος (και annotation) του JPA
- Είναι ουσιαστικά η μνήμη cache (first-level cache)
- Τα **Entities** που βρίσκονται στο **PersistenceContext** είναι **managed** (από το PersistenceContext και τον αντίστοιχο EntityManager), αλλιώς **unmanaged**



PersistentContext (2)

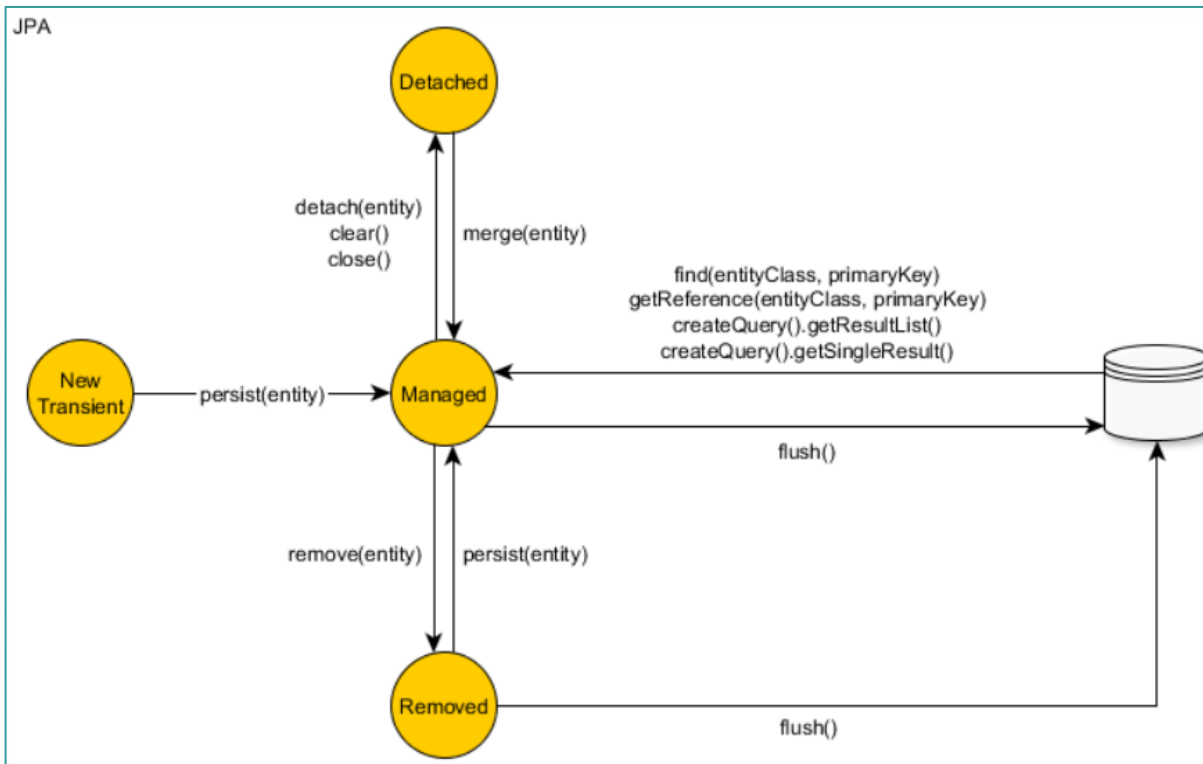
Hibernate

- Τα Entities που δεν βρίσκονται στο PersistenceContext είναι -όταν δημιουργηθούν- σε κατάσταση **Transient** και πρέπει να γίνουν persist ώστε να εισαχθούν στο PersistenceContext και να γίνουν managed (δηλαδή να αντιστοιχηθούν σε κάποιο Database Record)



PersistenceContext (3)

Hibernate



- Μία JPA entity μπορεί να είναι σε μία από τις ακόλουθες καταστάσεις (states):
- New (Transient)
- Managed (Associated)
- Detached (Dissociated)
- Removed (Deleted)

- Για να αλλάξουμε το state μίας entity χρησιμοποιούμε τις μεθόδους **persist**, **merge**, ή **remove** του JPA EntityManager, όπως θα δούμε στη συνέχεια



EntityManager

Hibernate

- Ο **EntityManager** είναι το βασικό interface του JPA που αναπαριστά και διαχειρίζεται το *PersistenceContext*, δηλαδή την cache
- Το **EntityManager API** παρέχει μεθόδους για να κάνουμε **persist** entity instances (δηλαδή insert στη ΒΔ), **merge** entities (δηλαδή update στη ΒΔ) & **remove** entity (δηλαδή delete από τη ΒΔ), για να κάνουμε **find** entities (δηλαδή select από τη ΒΔ) με το primary key, καθώς και να δημιουργούμε **queries**
- Για να εκτελεστούν τα παραπάνω στη ΒΔ πρέπει να γίνουν commit. Ο **EntityManager** **δεν είναι thread-safe**, δηλαδή δεν μπορεί να διασφαλίσει concurrent transactions από πολλά threads



JPA CRUD API

Hibernate

- Ο EntityManager παρέχει το παρακάτω JPA API:
 - ***persist*** – εισάγει στο PersistentContext ένα entity και το αποθηκεύει στη ΒΔ στο επόμενο commit
 - ***merge***- κάνει update το state ενός entity στο PersistentContext και στη ΒΔ στο επόμενο commit
 - ***remove***- διαγράφει ένα entity από το PersistentContext καθώς και από τη ΒΔ στο επόμενο commit
 - ***find*** – κάνει return ένα reference του Entity με βάση το id αν υπάρχει στην cache, μετά ψάχνει στην 2nd level cache (αν έχει ενεργοποιηθεί) και αν δεν υπάρχει πουθενά το αναζητά στην ΒΔ και το επιστρέφει



Hibernate Native API

Hibernate

- Στο Hibernate Native API, η first-level cache αναπαρίσταται από το **Session** interface που κάνει extends το JPA EntityManager interface
- Το Hibernate παρέχει επίσης, επιπλέον του JPA και δικές του μεθόδους για τις ίδιες λειτουργίες, όπως *save*, *saveOrUpdate*, *update*, *get*, κλπ..
- Αν όμως θέλουμε ανεξαρτησία από το ORM platform θα πρέπει να γράφουμε όσο περισσότερο μπορούμε JPA-compliant CRUD methods



JPA persist

Hibernate

persist

```
void persist(Object entity)
```

Make an instance managed and persistent.

Parameters:

entity - entity instance

Throws:

`EntityExistsException` - if the entity already exists. (If the entity already exists, the `EntityExistsException` may be thrown when the `persist` operation is invoked, or the `EntityExistsException` or another `PersistenceException` may be thrown at flush or commit time.)

`IllegalArgumentException` - if the instance is not an entity

`TransactionRequiredException` - if there is no transaction when invoked on a container-managed entity manager of that is of type `PersistenceContextType.TRANSACTION`

- Αν δημιουργήσουμε ένα instance, είναι σε κατάσταση Transient (unmanaged).
- Αν το κάνουμε persist, εισάγεται στην cache (PersistenceContext) και γίνεται managed



remove

Hibernate

remove

```
void remove(Object entity)
```

Remove the entity instance.

Parameters:

entity - entity instance

Throws:

`IllegalArgumentException` - if the instance is not an entity or is a detached entity

`TransactionRequiredException` - if invoked on a container-managed entity manager of type `PersistenceContextType.TRANSACTION` and there is no transaction

- Αν κάνουμε remove ένα managed instance, γίνεται removed



merge

Hibernate

merge

```
<T> T merge(T entity)
```

Merge the state of the given entity into the current persistence context.

Parameters:

entity - entity instance

Returns:

the managed instance that the state was merged to

Throws:

`IllegalArgumentException` - if instance is not an entity or is a removed entity

`TransactionRequiredException` - if there is no transaction when invoked on a container-managed entity manager of that is of type `PersistenceContextType.TRANSACTION`

- Η μέθοδος merge κάνει update το state του instance στο PersistenceContext



find

Hibernate

find

```
<T> T find(Class<T> entityClass,  
           Object primaryKey)
```

Find by primary key. Search for an entity of the specified class and primary key. If the entity instance is contained in the persistence context, it is returned from there.

Parameters:

`entityClass` - entity class

`primaryKey` - primary key

Returns:

the found entity instance or null if the entity does not exist

Throws:

`IllegalArgumentException` - if the first argument does not denote an entity type or the second argument is not a valid type for that entity's primary key or is null

- Η `find` αναζητά ένα Entity instance με βάση το id του Entity από το Persistence Context και αν υπάρχει, το επιστρέφει
- Αν δεν υπάρχει, ψάχνει στο 2nd level cache του Hibernate, αν έχει ενεργοποιηθεί
- Αν δεν ισχύει τίποτα από τα παραπάνω, ψάχνει στη ΒΔ



flush

```
void flush()
```

Synchronize the persistence context to the underlying database.

Throws:

`TransactionRequiredException` - if there is no transaction or if the entity manager has not been joined to the current transaction

`PersistenceException` - if the flush fails

- Συγχρονίζει το `PersistenceContext` με την ΒΔ δημιουργώντας (χωρίς να εκτελεί) DB εντολές



EntityTransaction

Hibernate

getTransaction

```
EntityTransaction getTransaction()
```

Return the resource-level EntityTransaction object. The EntityTransaction instance may be used serially to begin and commit multiple transactions.

Returns:

EntityTransaction instance

Throws:

IllegalStateException - if invoked on a JTA entity manager

- Όλες οι JPA CRUD πράξεις πρέπει να λαμβάνουν χώρα μέσα σε ένα transaction που αναπαρίσταται από το interface **EntityTransaction**. Το EntityTransaction (το οποίο λαμβάνουμε με EntityManager.getTransaction()) μας παρέχει διάφορες μεθόδους για να κάνουμε transactions όπως θα δούμε στη συνέχεια, μία από τις οποίες είναι η begin()
- Μπορούμε να κάνουμε **em.getTransaction.begin()** για να ξεκινήσουμε ένα transaction – όπου *em* είναι ένα instance του EntityManager που έχουμε δημιουργήσει με τον EntityManagerFactory()
- Ένα transaction μπορεί να αποτελείται από πολλές επιμέρους CRUD πράξεις, που θεωρούνται ως μία atomic operation. Πρέπει δηλαδή να εκτελεστούν **όλες ή καμία**. Γιαυτό αν αποτύχει το commit, θα πρέπει να γίνει rollback



EntityTransaction (1)

Hibernate

Interface EntityTransaction

```
public interface EntityTransaction
```

Interface used to control transactions on resource-local entity managers. The `EntityManager.getTransaction()` method returns the `EntityTransaction` interface.

Since:

Java Persistence 1.0

Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type		Method and Description
void		begin() Start a resource transaction.
void		commit() Commit the current resource transaction, writing any unflushed changes to the database.
boolean		getRollbackOnly() Determine whether the current resource transaction has been marked for rollback.
boolean		isActive() Indicate whether a resource transaction is in progress.
void		rollback() Roll back the current resource transaction.
void		setRollbackOnly() Mark the current resource transaction so that the only possible outcome of the transaction is for the transaction to be rolled back.

- Οι μέθοδοι που παρέχει το EntityTransaction



EntityTransaction (2)

Hibernate

begin

```
void begin()
```

Start a resource transaction.

Throws:

`IllegalStateException` - if `isActive()` is true

commit

```
void commit()
```

Commit the current resource transaction, writing any unflushed changes to the database.

Throws:

`IllegalStateException` - if `isActive()` is false

`RollbackException` - if the commit fails

rollback

```
void rollback()
```

Roll back the current resource transaction.

Throws:

`IllegalStateException` - if `isActive()` is false

`PersistenceException` - if an unexpected error condition is encountered

- Με ***begin*** ξεκινάμε ένα transaction
- Με ***commit*** εκτελούνται στη ΒΔ τα `persist`, `merge` κλπ. `persistence` εντολές που έχουμε ήδη εκτελέσει (εισάγει) στο `PersistentContext`
- Με ***rollback*** ανακαλούμε όλες τις εντολές του transaction αν κάτι δεν πάει καλά



EntityManagerFactory

Hibernate

- Είναι ένα interface του JPA που δημιουργεί ένα ή περισσότερους Entity managers
- Είναι thread-safe
- Μπορούμε να ενεργοποιήσουμε second-level cache σε επίπεδο EntityManagerFactory που διαμοιράζεται σε όλους τους Entity Managers