ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Criteria API, Lombok, και ειδικά θέματα σχεδιασμού

Αθ. Ανδρούτσος

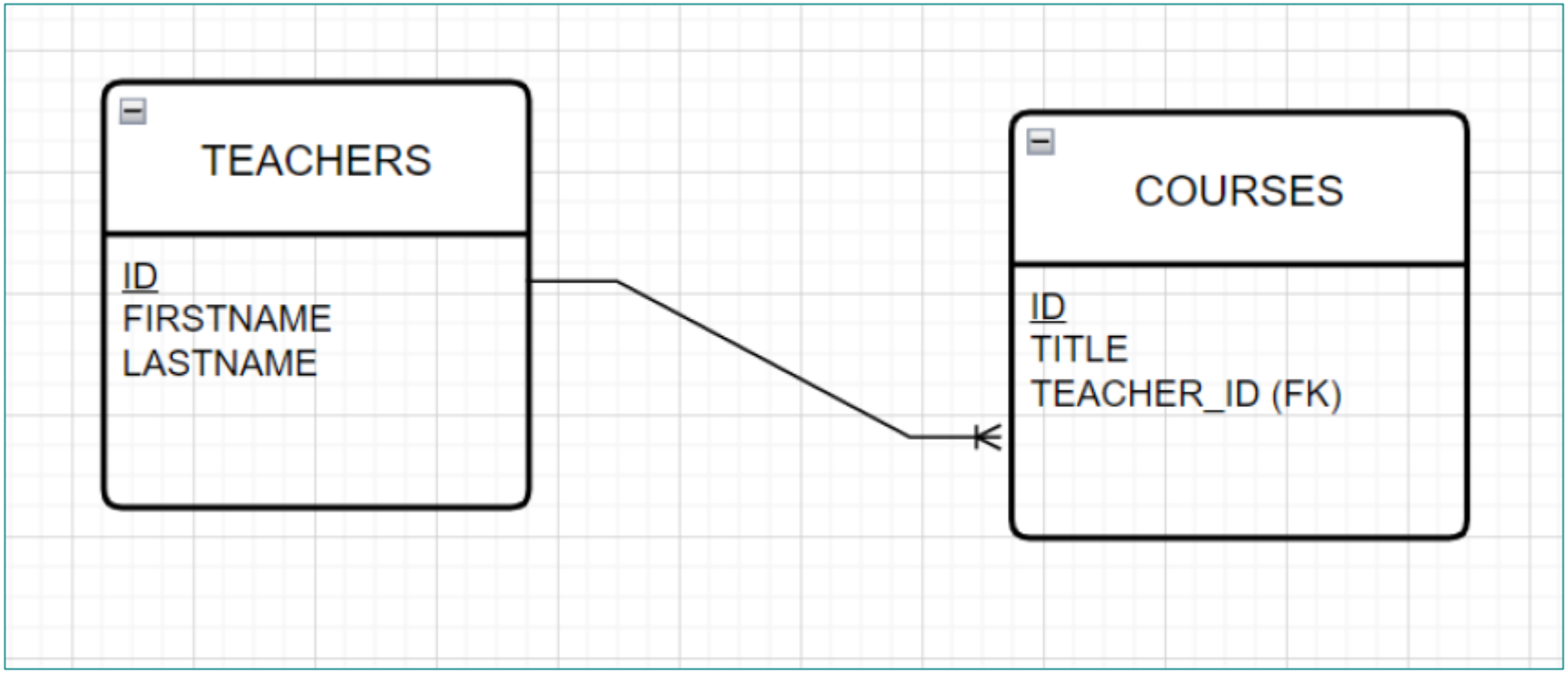
JPQL vs Criteria API

- Τόσο η **JPQL** όσο και τα **Native Queries** έχουν μειονεκτήματα ιδιαίτερα όσο αφορά το γεγονός ότι δεν είναι εύκολο να εκτελούμε δυναμικά queries, δηλαδή queries που μεταβάλλεται η δομή τους @runtime.
- Για παράδειγμα αν έχουμε φίλτρα όπως σε ένα e-shop όπου φιλτράρουμε τα προϊόντα ως προς διάφορα κριτήρια (τιμή, κατασκευαστή, χρώμα, κλπ.), τότε με JPQL θα πρέπει να συνενώνουμε WHERE Strings αν ο χρήστης έχει επιλέξει κάποια κριτήρια
- Για τον λόγο αυτό τα **Criteria Queries**, που θα δούμε στη συνέχεια, έχουν περισσότερα πλεονεκτήματα

Criteria API

- Η βασική ιδέα στο Criteria API είναι η προγραμματιστική δόμηση ενός Query
- Το βασικό interface, ο CriteriaBuilder, παρέχει μεθόδους για να ορίζουμε κριτήρια (Where) ως predicates
- Το άλλο βασικό interface το CriteriaQuery, παρέχει μεθόδους για να ορίσουμε τι δεδομένα επιστρέφονται και ποιους πίνακες ρωτάμε

E-R Model



- Έστω το domain model που έχουμε δει και αποτελείται από ένα πίνακα teachers ένα-προς-πολλά προς τον courses

Domain Model


```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import org.hibernate.annotations.NaturalId;

import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "teachers")
public class Teacher {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstname;
    private String lastname;

    @OneToMany(mappedBy = "teacher")
    private Set<Course> courses = new HashSet<>();

```

```java
package gr.aueb.cf.model;

import jakarta.persistence.*;

@Entity
@Table(name = "courses")
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "teacher_id")
    private Teacher teacher;
```

Το Domain Model αποτελείται από δύο entities

SQL Insert


```
INSERT INTO `` (`id`,`firstname`,`lastname`) VALUES (1,'Αθανάσιος','Ανδρούτσος');
INSERT INTO `` (`id`,`firstname`,`lastname`) VALUES (2,'Μάκης','Καπέτης');
INSERT INTO `` (`id`,`firstname`,`lastname`) VALUES (3,'Άννα','Γιαννούτσου');
INSERT INTO `` (`id`,`firstname`,`lastname`) VALUES (4,'Μάρκος','Καραμπάτσης');
INSERT INTO `` (`id`,`firstname`,`lastname`) VALUES (5,'Παναγιώτης','Μόσχος');
INSERT INTO `` (`id`,`firstname`,`lastname`) VALUES (6,'Σοφοκλής','Στουραϊτης');
INSERT INTO `` (`id`,`firstname`,`lastname`) VALUES (7,'Ανδρέας','Ανδρούτσος');
```

```
INSERT INTO `` (`id`,`title`,`teacher_id`) VALUES (1,'Java',1);
INSERT INTO `` (`id`,`title`,`teacher_id`) VALUES (2,'Cobol',1);
INSERT INTO `` (`id`,`title`,`teacher_id`) VALUES (3,'C#',2);
INSERT INTO `` (`id`,`title`,`teacher_id`) VALUES (4,'Python',3);
INSERT INTO `` (`id`,`title`,`teacher_id`) VALUES (5,'SQL',4);
INSERT INTO `` (`id`,`title`,`teacher_id`) VALUES (6,'React',4);
```

- Εισάγουμε στη ΒΔ δεδομένα ώστε να τρέξουμε queries

Queries (1)


```
23          em.getTransaction().begin();
24
25          // List all teachers
26          CriteriaBuilder cb = em.getCriteriaBuilder();
27          CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
28          Root<Teacher> teacher = query.from(Teacher.class);
29
30          query.select(teacher);
31          List<Teacher> teachers = em.createQuery(query).getResultList();
32
33          em.getTransaction().commit();
34
35          teachers.forEach(System.out::println);
```

- Τα πιο σημαντικά interfaces είναι ο CriteriaBuilder και ο CriteriaQuery<T>. Εδώ πρόκειται για απλό select χωρίς κριτήρια (where) και επομένως επιστρέφει όλα τα teacher instances

Queries (2)


```
// List All Course Titles
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<String> query = cb.createQuery(String.class);
Root<Course> course = query.from(Course.class);

query.select(course.get("title"));
List<String> titles = em.createQuery(query).getResultList();
```

- Ο CriteriaQuery ορίζει τι επιστρέφεται από το Query. Στο παράδειγμα επιστρέφεται String. Το RootEntity ορίζει το from και από που ξεκινά το navigation
- Το course.get(”title”) είναι path –σε όρους JPQL είναι θα ήταν course.title

Queries (3)


```java
// List All Teachers with a Specific Last Name
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
query.select(teacher).where(cb.equal(teacher.get("firstname"), "Αθανάσιος"));
List<Teacher> teachers = em.createQuery(query).getResultList();
```

```java
// List All Teachers with a Specific Last Name - SQL Injection safe
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
ParameterExpression<String> lastnameParam = cb.parameter(String.class, "lastname");
query.select(teacher).where(cb.equal(teacher.get("firstname"), lastnameParam));
List<Teacher> teachers = em.createQuery(query).setParameter("lastname", "Αθανάσιος").getResultList();
```

- Η 2η έκδοση χρησιμοποιεί ParameterExpression<T> ώστε να μην υπεισέρχονται Strings μέσα στα criteria όπως το «Αθανάσιος» στην 1η έκδοση. Αυτό προφυλάσσει από SQL Injection attack

Queries (4)


```
// Find Courses Taught by a Specific Teacher
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Course> query = cb.createQuery(Course.class);
Root<Course> course = query.from(Course.class);
Join<Course, Teacher> teacher = course.join("teacher");

ParameterExpression<String> lastnameParam = cb.parameter(String.class, "lastname");
query.select(course).where(cb.equal(teacher.get("lastname"), lastnameParam));
List<Course> courses = em.createQuery(query).setParameter("lastname", "Ανδρούτσος").getResultList();
```

```
//  List Teachers with More Than One Course
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
query.select(teacher).where(cb.greaterThan(cb.size(teacher.get("courses")), 1));
List<Teacher> teachers = em.createQuery(query).getResultList();
```

- Μέσα στο where έχουμε predicates με την μορφή cb.* όπως cb.equal ή cb.greaterThan, κλπ. Μέσα τους τα predicates παίρνουν ένα key (που είναι ένα path) και ένα value

Queries (5)


```
// List Courses Along with Teacher Names
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Object[]> query = cb.createQuery(Object[].class);
Root<Course> course = query.from(Course.class);
Join<Course, Teacher> teacher = course.join("teacher");
query.multiselect(course.get("title"), teacher.get("lastname"),teacher.get("firstname"));
List<Object[]> coursesTeachers = em.createQuery(query).getResultList();
```

```
// List All Teachers Who Don't Teach Any Courses
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
query.select(teacher).where(cb.isEmpty(teacher.get("courses")));
List<Teacher> teachersNoCourses = em.createQuery(query).getResultList();
```

- Υπάρχει και η multiselect (βλ. 1ο παράδειγμα) με την οποία επιλέγουμε πολλά πεδία διαφόρων entities. Επίσης, υπάρχει ένα Join<> object που υλοποιεί τη σύνδεση μεταξύ δύο οντοτήτων και από όπου μπορούμε να πάρουμε paths όπως και από το root entity

Queries (6)


```
// Find Teachers and Count of Their Courses
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Object[]> query = cb.createQuery(Object[].class);
Root<Teacher> teacher = query.from(Teacher.class);
Join<Teacher, Course> course = teacher.join("courses");
query.multiselect(teacher, cb.count(course)).groupBy(teacher);
List<Object[]> teachersWithCourseCount = em.createQuery(query).getResultList();
```

```
//  Find Teachers Who Teach Courses with a Specific Title
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
query.select(teacher).where(cb.equal(teacher.get("courses").get("title"), "SQL"));
List<Teacher> teachersWithCourses = em.createQuery(query).getResultList();
```

Queries (7)


```
// Find Teachers Whose First Name Starts with a Specific Letter and Have More Than One Course
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
query.select(teacher).where(cb.like(teacher.get("firstname"), "A%"), cb.gt(cb.size(teacher.get("courses")), 1));
List<Teacher> teachersNameCoursesMoreThanOne = em.createQuery(query).getResultList();
```

Γενική προσέγγιση (1)


```
private static  <K> List<K> getByCriteria(Class<K> clazz, Map<String , Object> criteria) {
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<K> query = cb.createQuery(clazz);
    Root<K> kappa = query.from(clazz);

    List<Predicate> predicates = getPredicatesFromCriteria(cb, kappa, criteria);
    query.select(kappa).where(predicates.toArray(new Predicate[0]));

    TypedQuery<K> selectQuery = em.createQuery(query);
    addParametersToQuery(selectQuery, criteria);
    return selectQuery.getResultList();
}
```

- Μπορούμε να ορίσουμε μία γενική μέθοδο getByCriteria που λαμβάνει κριτήρια με τη μορφή Map<String, Object> και μία επιστρεφόμενη κλάση Class<K>
- Χρησιμοποιεί βοηθητικές μεθόδους για να λάβει τα predicates και να κάνει add παραμέτρους στο query

Γενική προσέγγιση (2)


```
private static List<Predicate> getPredicatesFromCriteria(CriteriaBuilder cb, Root<?> entityRoot,
                                                         Map<String, Object> criteria) {
    List<Predicate> predicates = new ArrayList<>();

    for (Map.Entry<String , Object> entry : criteria.entrySet()) {
        String key = entry.getKey();
        Object value = entry.getValue();
        ParameterExpression<?> val = cb.parameter(value.getClass(), buildParameter(key));
        //Predicate equal = cb.equal(resolvePath(entityRoot, key), val);
        Predicate predicateLike = cb.like((Expression<String>) resolvePath(entityRoot, key), (Expression<String>) val);
        //predicates.add(equal);
        predicates.add(predicateLike);
    }
    return predicates;
}
```

- Μετατρέπει το map των κριτηρίων σε predicates list. Το key (π.χ. firstname) μετατρέπεται σε Path (με τη resolvePath(root, key)) και το value μετατρέπεται σε String alias της παραμέτρου μέσα σε ένα ParameterExpression<?> (αν η παράμετρος είναι firstname, τότε το alias θα είναι “firstname” – το παίρνουμε με buildParameter(key))

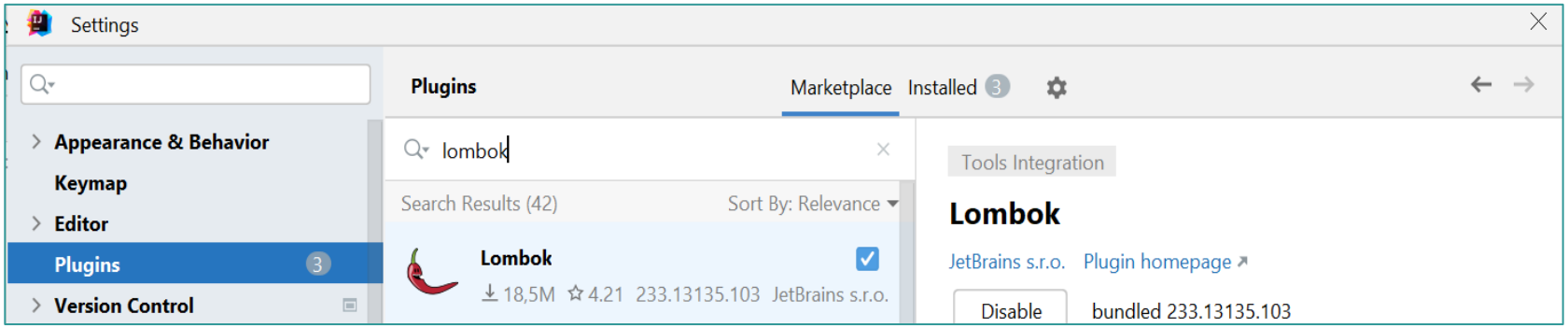
Γενική προσέγγιση (3)


```java
private static void addParametersToQuery(TypedQuery<?> query, Map<String , Object> criteria) {
    for (Map.Entry<String , Object> entry : criteria.entrySet()) {
        Object value = entry.getValue();
        query.setParameter(buildParameter(entry.getKey()), value);
    }
}

private static String buildParameter(String key) {
    return key.replaceAll("\\.", "");
}

private static Path<?> resolvePath(Root<?> entityRoot, String key) {
    String[] fields = key.split("\\.");
    Path<?> path = entityRoot.get(fields[0]);
    for (int i = 1; i < fields.length; i++) {
        path = path.get(fields[i]);
    }
    return path;
}
```

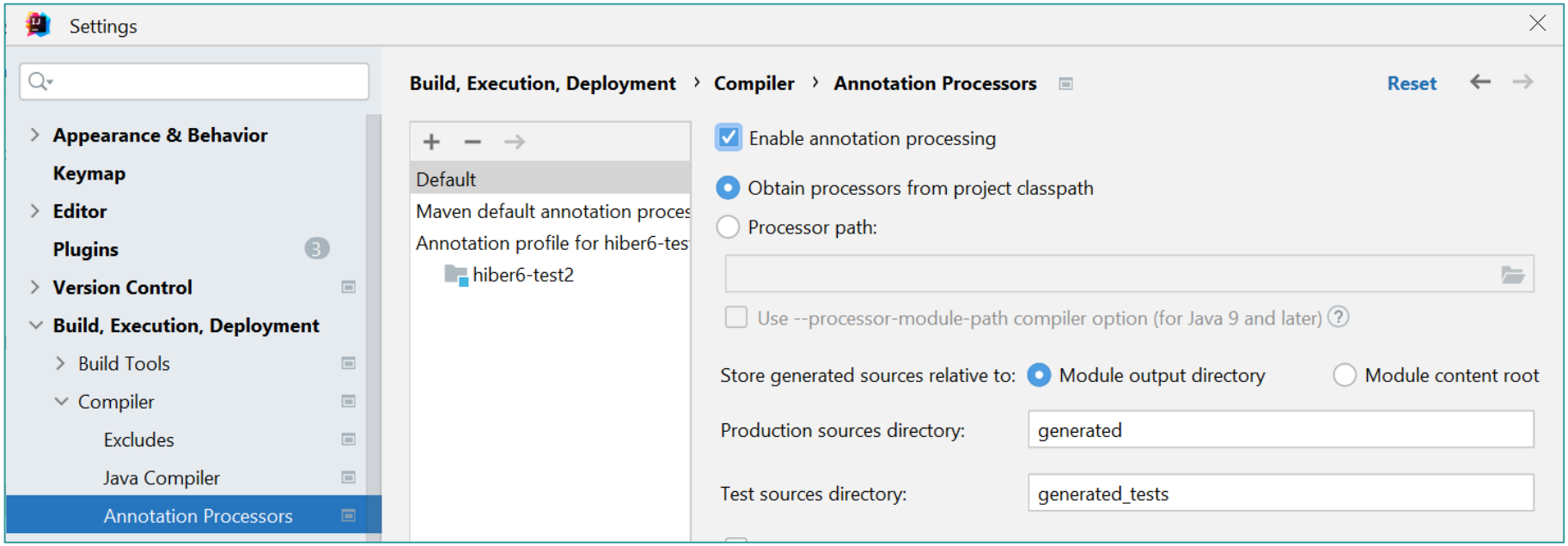
- Βοηθητικές μέθοδοι για add παραμέτρων –από τα κριτήρια- στο query. Για build παραμέτρων – αν για παράδειγμα μία παράμετρος είναι moreInfo.vat, τότε επιστρέφεται morInfovat. Επίσης, η resolvePath επιστρέφει το Path<> από το entityRoot και το key (π.χ. αν έχουμε teacher και firstname, τότε επιστρέφεται το αντίστοιχο path)

Lombok (1)

- Το Lombok είναι μία Java library που μειώνει το boilerplate παρέχοντας αυτόματα Constructors, Getters, Setters, toString(), equals(), hashCode(), κλπ
- Στο IntelliJ θα πρέπει να εγκατασταθεί το Lombok plugin



Lombok (2)



- Θα πρέπει επίσης να γίνει enable το annotation processing

Lombok (3)


```
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <version>1.18.30</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
          <configuration>
            <annotationProcessorPaths>
              <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <version>1.18.30</version>
              </path>
            </annotationProcessorPaths>
          </configuration>
        </plugin>
```

- Στο POM.xml του Maven θα πρέπει να εισάγουμε το Lombok dependency
- Καθώς και λίγο config στο maven-compiler-plugin ώστε να ενσωματωθεί το Lombok annotation processing
- Το scope είναι provided υπό την έννοια ότι το Lombok χρειάζεται μόνο @compile time, δεν είναι μέρος του final jar

Teacher


```java
package gr.aueb.cf.model;

import ...

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
@Table(name = "teachers")
public class Teacher {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstname;
    private String lastname;

    @Getter(AccessLevel.PRIVATE)
    @OneToMany(mappedBy = "teacher")
    private Set<Course> courses = new HashSet<>();
```

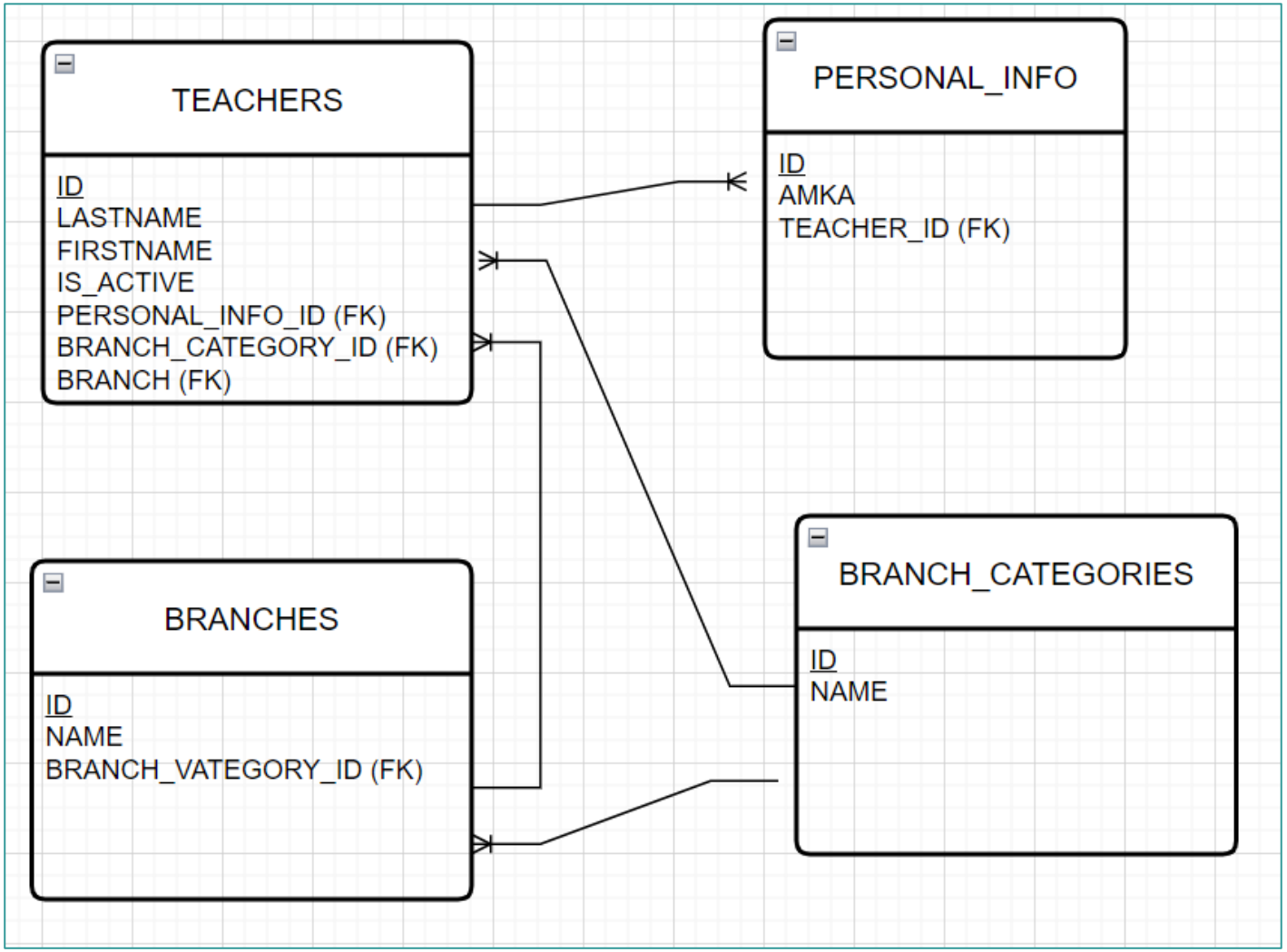
- Το Lombok παρέχει annotations όπως για **@NoArgsConstructor**, **@AllArgsConstructor**, **@Getter** και **@Setter**, που είναι βασικά annotations
- Παρέχονται επίσης:
- **@Data** για που περιλαμβάνει getters, setters, toString, equals, και hashCode
- **@RequiredArgsConstructor** που δίνει constructor για final πεδία και πεδία που είναι @NonNull
- @toString, που δημιουργεί την toString()

Course


```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import lombok.*;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
@Table(name = "courses")
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String description;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "teacher_id")
    private Teacher teacher;
```

- Με τον ίδιο τρόπο ορίζουμε και το Course entity

More Advanced Παράδειγμα



@MappedSupperclass (1)


```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.DynamicUpdate;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.UUID;

@Getter
@Setter
@DynamicUpdate      // Only fields that have changes will be included in UPDATE
@MappedSuperclass
public abstract class AbstractEntity implements Serializable {

    @Column(name = "created_at", nullable = false, updatable = false)
    private LocalDateTime createdAt;

    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    @Column(unique = true)
    private String uuid;
```

- Η AbstractEntity δεν χαρακτηρίζεται ως @Entity αλλά ως @MappedSupperclass μιας και **θέλουμε να κληρονομείται χωρίς να δημιουργείται πίνακας**
- Τα createdAt και updatedAt είναι πεδία για audit και συνηθίζεται να υπάρχουν σε όλους τους πίνακες.
- Θα ενημερώνονται αυτόματα

@MappedSupperclass (2)


```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.DynamicUpdate;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.UUID;

@Getter
@Setter
@DynamicUpdate      // Only fields that have changes will be included in UPDATE
@MappedSuperclass
public abstract class AbstractEntity implements Serializable {

    @Column(name = "created_at", nullable = false, updatable = false)
    private LocalDateTime createdAt;

    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    @Column(unique = true)
    private String uuid;
```

- Το uuid (universal unique id) είναι πεδίο που χαρακτηρίζει μοναδικά μία εγγραφή με ένα τυχαίο Hex (δεκαεξαδικό) 32 χαρακτήρων
- Τα UUID είναι χρήσιμα ως χαρακτηρισμοί (σαν πρωτόκολλο) εγγράφων καθώς και για να χρησιμοποιούμε αυτά που είναι τυχαία αντί να κάνουμε export το id που είναι ακολουθιακό και προβλέψιμο

@MappedSupperclass (3)


```
26
27      @PrePersist
28      protected void onCreate() {
29          createdAt = LocalDateTime.now();
30          updatedAt = LocalDateTime.now();
31          if (uuid == null) uuid = UUID.randomUUID().toString();
32      }
33
34      @PreUpdate
35      protected void onUpdate() {
36          updatedAt = LocalDateTime.now();
37      }
38  }
```

- Με @PrePersist ορίζουμε μία μέθοδο να εκτελείται αμέσως πριν το INSERT ενώ με @PreUpdate αμέσως πριν το Update

BranchCategory (1)


```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import lombok.*;

import java.util.HashSet;
import java.util.Set;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Table(name = "branch_categories")
public class BranchCategory extends AbstractEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    @Getter(AccessLevel.PROTECTED)
    @OneToMany(mappedBy = "branchCategory")
    private Set<Branch> branches = new HashSet<>();

```

- Ορίζουμε το BranchCategory με Lombok οπότε επικεντρωνόμαστε στο state και στα Relations

BranchCategory (2)


```
26      @Getter(AccessLevel.PROTECTED)
27      @OneToMany(mappedBy = "branchCategory")
28      private Set<Teacher> teachers = new HashSet<>();
29
30      public void addBranch(Branch branch) {
31          if (branches == null) branches = new HashSet<>();
32          branches.add(branch);
33          branch.setBranchCategory(this);
34      }
35
36      public void removeBranch(Branch branch) {
37          if (branches == null) branches = new HashSet<>();
38          branches.remove(branch);
39          branch.setBranchCategory(null);
40      }
41
42      public void addTeacher(Teacher teacher) {
43          if (teachers == null) teachers = new HashSet<>();
44          teachers.add(teacher);
45          teacher.setBranchCategory(this);
46      }
47
48      public void removeTeacher(Teacher teacher) {
49          if (teachers == null) teachers = new HashSet<>();
50          teachers.remove(teacher);
51          teacher.setBranchCategory(null);
52      }
53  }
```

- Εισάγουμε επίσης convenient methods για add, remove στα Collections branches και teachers

Branch (1)


```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import lombok.*;

import java.util.HashSet;
import java.util.Set;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Table(name = "branches")
public class Branch extends AbstractEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "branch_category_id")
    private BranchCategory branchCategory;

```

- Το Branch έχει μία σχέση @ManyToOne, οπότε έχει και το @JoinColumn

Branch (2)


```
26      @Getter(AccessLevel.PROTECTED)
27      @OneToMany(mappedBy = "branch")
28      private Set<Teacher> teachers = new HashSet<>();
29
30      public void addTeacher(Teacher teacher) {
31          if (teachers == null) teachers = new HashSet<>();
32          teachers.add(teacher);
33          teacher.setBranch(this);
34      }
35
36      public void removeTeacher(Teacher teacher) {
37          if (teachers == null) teachers = new HashSet<>();
38          teachers.remove(teacher);
39          teacher.setBranch(null);
40      }
41  }
```

Teacher

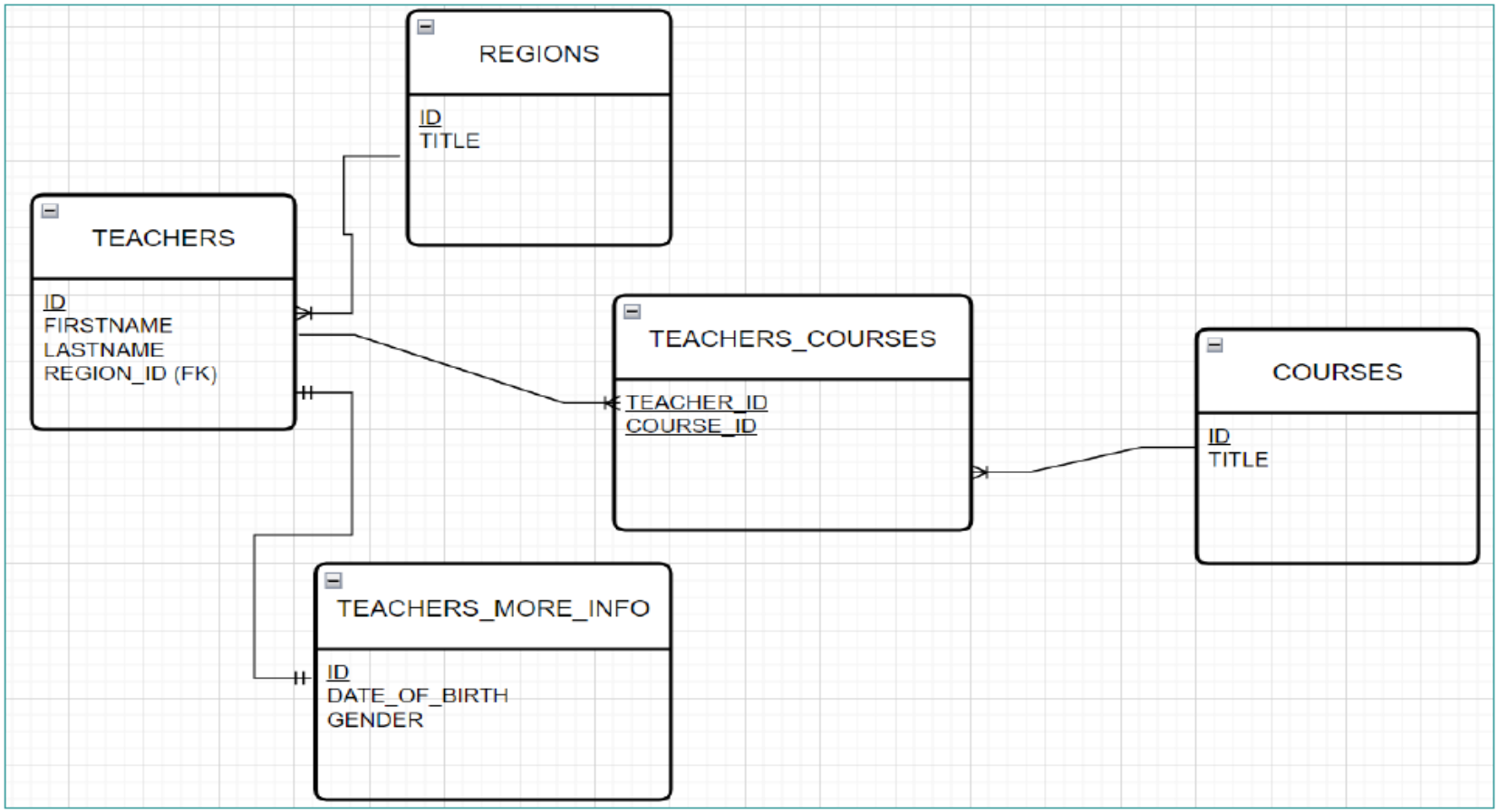

```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.ColumnDefault;
import org.hibernate.annotations.DynamicInsert;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
@DynamicInsert
@Table(name = "teachers")
public class Teacher extends AbstractEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstname;
    private String lastname;

    @ColumnDefault("true")
    @Column(name = "is_active", nullable = false)
    private Boolean isActive;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "branch_category_id")
    private BranchCategory branchCategory;

    @ManyToOne
    @JoinColumn(name = "branch_id")
    private Branch branch;

    @OneToOne(cascade = CascadeType.ALL,
            orphanRemoval = true)
    @JoinColumn(name = "personal_info_id")
    private PersonalInfo personalInfo;
}
```

PersonalInfo


```java
package gr.aueb.cf.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Table(name = "personal_info")
public class PersonalInfo extends AbstractEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String amka;
}
```

- Η σχέση Teacher και PersonalInfo είναι unidirectional από το Teacher στο PersonalInfo
- Από το PersonalInfo δεν έχει relation προς το Teacher

Εργασία



- Υλοποιήστε το παραπάνω Domain Model με Lombok