Samenvatting van de cursus Entrepirise Web Development

Robbe Magerman

11/02/2025

Inhoudstafel

1 Java Persistence API

1.1 Maven opstart

- Een `JPA`-project heeft in zijn `pom.xml`-file enkele dependencies nodig
- Plak dit alles onder de **version** tag

```
<properties>
  <maven.compiler.source>21</maven.compiler.source>
  <maven.compiler.target>21</maven.compiler.target>
</properties>

<dependencies>
  <!-- Lombok -->
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.36</version>
    <scope>provided</scope>
  </dependency>

  <!-- JUnit -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.11.0</version>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>5.11.0</version>
  </dependency>

  <!-- EclipseLink JPA -->
  <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa</artifactId>
    <version>4.0.4</version>
  </dependency>

  <!-- MySQL Connector -->
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>9.0.0</version>
  </dependency>
</dependencies>
```

1.2 Klein projectje - zonder relaties

1.2.1 Theorie

- **Serializable**:
  - Serializable: aangeraden voor sessiebeheer en netwerkcommunicatie. EclipseLink vereist het niet, maar sommige frameworks en andere JPAimplementaties wel.

1.2.2 Conneteren naar MySQL

- Onder tablad **query** klik op het database icoontje en maak daar je nieuwe databank aan
- Nu hebben we onze databank en kunnen we dit koppelen via een **Java-klasse**
- In `Java/Spring` maken we in **src/main/java** een nieuwe `folder` genaamd:

```
META-INF
```

- In deze folder maken we een nieuw bestand via `new -> other`
- Typ daar **xml** in
- Kies dan `XML/XML File` en noem die **persistence**
- Je moet hier `.xml` niet schrijven want hij doet dat voor jou als het klaar is
- In dat bestand plaats je het volgende:

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
             https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">

  <persistence-unit name="school" transaction-type="RESOURCE_LOCAL"> <!-- Dit
     is de koppeling tussen de fabriek/factory die we gaan maken -->
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>domein.Docent</class> <!-- Welke javaklasse moet in de databank -->
    <properties>
      <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://
         localhost:3306/JOUW_DATABANK?serverTimezone=UTC"/> <!-- schooldb -->
      <property name="jakarta.persistence.jdbc.user" value="JOUW_USERNAME"/> <!
         -- root -->
      <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc
         .Driver"/>
      <property name="jakarta.persistence.jdbc.password" value="JOUW_WACHTWOORD
         "/>
      <property name="jakarta.persistence.schema-generation.database.action"
         value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

- Zorg ervoor dat je **JOUW_DATABANK JOUW_USERNAME JOUW_WACHTWOORD** zeker invult
- Maak nu in **src/main/java** `util/JPAUtil.java` met volgende code:
- Hier maken we onze **persistenceFactory**

```
package util;

import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;
import lombok.AccessLevel;
import lombok.Getter;
import lombok.NoArgsConstructor;

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class JPAUtil {

    @Getter
```

```
    private final static EntityManagerFactory entityManagerFactory =
                            Persistence.createEntityManagerFactory("school");

}
```

1.2.3 Docent class

- Nu maken we `domein/Docent` wat een **Entity**-klasse is
- Al de attributen die we hier maken, zijn de 'velden' van MySQL:

```
package domein;

import java.io.Serializable;
import java.math.BigDecimal;
import lombok.Getter;
import lombok.Setter;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Table;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Column;

@Entity
@Table(name = "docenten")
@NoArgsConstructor (access = AccessLevel.PROTECTED) // default constructor is
   verplicht of je krijgt een EntityException
public class Docent implements Serializable{

    private static final long serialVersionUID = 1L;

    // Dit zorgt voor autonummering bij het maken van Docenten
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name = "PERSONEELSNR")
    @Getter private int docentNr;

    @Getter private String voornaam;
    @Getter private String familienaam;
    @Getter @Setter private BigDecimal wedde;

    public Docent(int docentNr, String voornaam, String familienaam, BigDecimal
        wedde) {
        this.docentNr = docentNr;
        this.voornaam = voornaam;
        this.familienaam = familienaam;
        this.wedde = wedde;
    }

    public void opslag(BigDecimal bedrag) {
        wedde = wedde.add(bedrag);
    }

    @Override
```

```
    public String toString() {
            return "%d %s %s %s".formatted(docentNr, voornaam, familienaam,
                wedde);
    }

}
```

1.2.4 Main class

- Nu voeren we dit via `main/MAINoef1` :

```
package main;

import java.math.BigDecimal;

import domein.Docent;
import jakarta.persistence.EntityManager;
import util.JPAUtil;

public class MAINoef1 {

    public static void main(String args[]) {

        Docent d1 = new Docent(123, "Jan", "Baard", new BigDecimal(8000));
        Docent d2 = new Docent(456, "Piet", "Baard", new BigDecimal(10000));
        Docent d3 = new Docent(789, "Joris", "ZonderBaard", new BigDecimal
            (12000));

        // Vraag aan de factory een entityManager
        EntityManager entityManager = JPAUtil.getEntityManagerFactory().
            createEntityManager();

        // Start een transactie
        entityManager.getTransaction().begin();

        // Persisteer de 3 objecten - zet de docenten in de databank
        entityManager.persist(d1);
        entityManager.persist(d2);
        entityManager.persist(d3);

        // Commit
        entityManager.getTransaction().commit();

        // Sluit de entityManager
        entityManager.close();

        // Sluit de factory
        JPAUtil.getEntityManagerFactory().close();
    }
}
```

- Als nu nu MySQL refreshen zien we dat onze docenten in de Databank staan
- Dit is de basis van een project

1.3 Klein projectje - met relaties

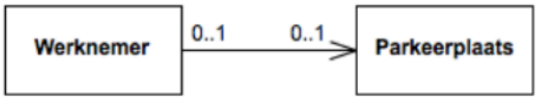
1.3.1 Theorie Relaties

Hoe weet je welke annotation je moet gebruiken?

Stel dat je dit voor het voorbeeld `many-to-one - Unidirectioneel` wil doen vanuit **Werknemer**, moet je eerst kijken naar de maximumwaarde van die tabel. Hier is dat '*' dus we kiezen alvast `ManyTo` en de waarde van de tegenovergestelde tabel is '0..1' dus `One` .

Verschil tussen bidirectioneel en unidirectioneel

- Bidirectioneel
  - Beide klassen kunnen aan beide klassen
- Unidirectioneel
  - 1 klas kan aan de andere van waar de pijl uit vertrekt, maar omgekeerd niet
- **One-to-one - Unidirectioneel**

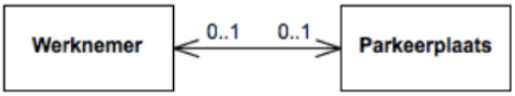



```
class Werknemer {
  @Id int id;

  @OneToOne
  Parkeerplaats p;
}
```

```
class Parkeerplaats {
  @Id int id;
}
```

Figure 1: One-to-one - Unidirectioneel

  -
- **One-to-one - Bidirectioneel**

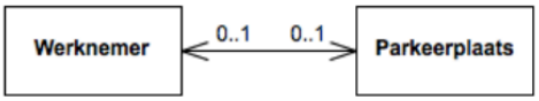



```
class Werknemer {
  @Id int id;

  @OneToOne
  Parkeerplaats p;
}
```

```
class Parkeerplaats {
  @Id int id;

  @OneToOne(mappedBy="p")
  Werknemer w;
}
```

Figure 2: One-to-one - Bidirectioneel

  -
- **One-to-one - Bidirectioneel(2)**

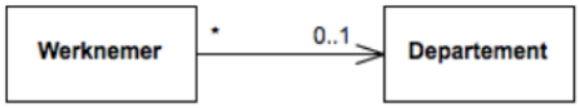



```
class Werknemer {
  @Id int id;

  @OneToOne(mappedBy="w")
  Parkeerplaats p;
}
```

```
class Parkeerplaats {
  @Id int id;

  @OneToOne
  Werknemer w;
}
```

Figure 3: One-to-one - Bidirectioneel(2)

- **Many-to-one - Unidirectioneel**

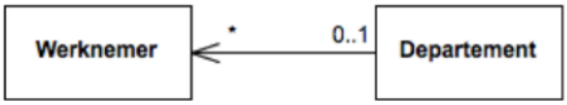



```
class Werknemer {
  @Id int id;

  @ManyToOne
  Departement d;
}
```

```
class Departement {
  @Id int id;
}
```

Figure 4: Many-to-one - Unidirectioneel

- **One-to-Many - Unidirectioneel**

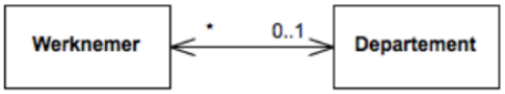



```
class Werknemer {
  @Id int id;
}
```

```
class Departement {
  @Id int id;

  @OneToMany
  List<Werknemer> w;
}
```

Figure 5: One-to-Many - Unidirectioneel

- **One-to-many / Many-to-one Bidirectioneel**



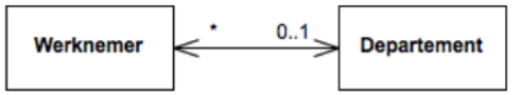

```
class Werknemer {
  @Id int id;

  @ManyToOne
  Departement d;
}
```

```
class Departement {
  @Id int id;

  @OneToMany(mappedBy="d")
  List<Werknemer> w;
}
```

Figure 6: One-to-many / Many-to-one Bidirectioneel

- **One-to-one / Many-to-one Bidirectioneel(2)**

One-to-many / Many-to-one - Bidirectioneel (2)

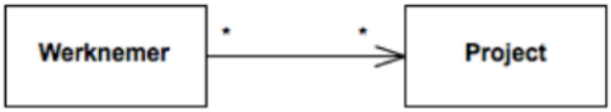



```
class Werknemer {
  @Id int id;

  @ManyToOne
  @JoinTable
  Departement d;
}
```

```
class Departement {
  @Id int id;

  @OneToMany(mappedBy="d")
  List<Werknemer> w;
}
```

Figure 7: One-to-one / Many-to-one Bidirectioneel(2)

- **Many-to-many - Unidirectioneel**




```
class Werknemer {
  @Id int id;

  @ManyToMany
  List<Project> p;
}
```

```
class Project {
  @Id int id;
}
```

Figure 8: Many-to-many - Unidirectioneel

- **Many-to-many - Bidirectioneel**

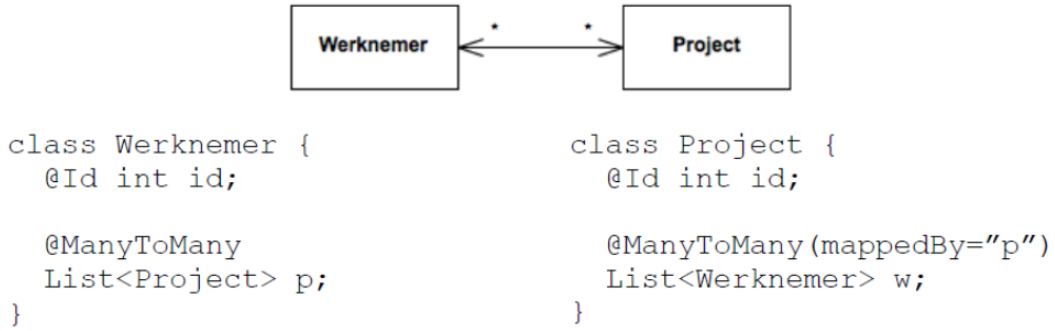


Figure 9: Many-to-many - Bidirectioneel

  -

- **Many-to-many - Bidirectioneel(2)**

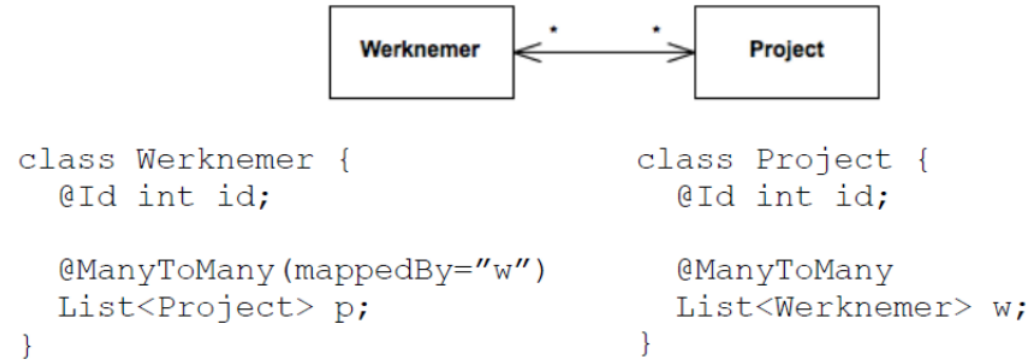


Figure 10: Many-to-many - Bidirectioneel(2)

  -

1.3.2 Uitwerking

- We passen eerst onze `META-INF/persistence.xml` aan om 3 nieuwe klassen te includen:

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
             https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">

  <persistence-unit name="school" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>domein.Docent</class>
    <class>domein.Campus</class>
    <class>domein.Werkruimte</class>
    <properties>
      <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://
         localhost:3306/JOUW_DATABANK?serverTimezone=UTC"/> <!-- schooldb2 -->
```

```
      <property name="jakarta.persistence.jdbc.user" value="JOUW_USERNAME"/> <!
         -- root -->
      <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc
         .Driver"/>
      <property name="jakarta.persistence.jdbc.password" value="JOUW_WACHTWOORD
         "/>
      <property name="jakarta.persistence.schema-generation.database.action"
         value="drop-and-create"/>
    </properties>
  </persistence-unit>
</persistence>
```

- Daarna maken we twee nieuwe klassen waaronder: `domein/Campus` :

```
package domein;

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

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.Table;
import lombok.AccessLevel;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Table(name = "campussen")
@NoArgsConstructor(access = AccessLevel.PROTECTED)
// @EqualsAndHashCode(of = "campusID") - kies nooit de primary key om dit op te
    doen
@EqualsAndHashCode(of = "campusNaam")

public class Campus implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int campusID;

    @Getter
    @Setter
    private String campusNaam;

    @ManyToMany
    private final Set<Docent> docenten = new HashSet<>();

    public Campus(String campusNaam) {
        this.campusNaam = campusNaam;
    }
```

```
    public Set<Docent> getDocenten() {
        return Collections.unmodifiableSet(docenten);
    }

    public void addDocent(Docent docent) {
        docenten.add(docent);
    }

    public void removeDocent(Docent docent) {
        docenten.remove(docent);
    }

    @Override
    public String toString() {
        return "%d %s".formatted(campusID, campusNaam);
    }

}
```

- Daarna `domein/Werkruimte` :

```
package domein;

import java.io.Serializable;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AccessLevel;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@Entity
@Table(name = "werkruimtes")
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@EqualsAndHashCode(of = "lokaalcode")
public class Werkruimte implements Serializable {

    private static final long serialVersionUID = 1L;

    @Setter(AccessLevel.PRIVATE)
    @Id
    private String lokaalcode;

    private String naam;
    private int aantalStoelen;
    private int aantalComputers;

    public Werkruimte(String lokaalcode, String naam, int aantalStoelen, int
       aantalComputers) {
        setLokaalcode(lokaalcode);
        setNaam(naam);
        setAantalStoelen(aantalStoelen);
```

```
        setAantalComputers(aantalComputers);
    }

    @Override
    public String toString() {
        return "%s %s %d %d".formatted(lokaalcode, naam, aantalStoelen,
            aantalComputers);
    }

}
```

- Daarna moeten we enkel nog `domein/Docent` een paar kleine aanpassingen aan aanbrengen:

```
package domein;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import lombok.AccessLevel;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Table(name = "docenten")
@NoArgsConstructor(access = AccessLevel.PROTECTED) // default constructor is
   verplicht of je krijgt een EntityException
@EqualsAndHashCode(of = "docentNr")
public class Docent implements Serializable {

    private static final long serialVersionUID = 1L;

    // Dit zorgt voor autonummering bij het maken van Docenten
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name = "PERSONEELSNR")
    @Getter
    private int docentNr;

    @Getter
    private String voornaam;

    @Getter
    private String familienaam;
```

```
    @Getter
    @Setter
    private BigDecimal wedde;

    // ManyTo kijkt hier naar de campussen
    // Many kijkt naar zichzelf
    // mappedBy maakt niet uit of je dat bij de Campus of Docent schrijft
    @ManyToMany(mappedBy = "docenten")
    private final Set<Campus> campussen = new HashSet<>();

    @Setter
    @ManyToOne
    private Werkruimte werkruimte;

    public Docent(int docentNr, String voornaam, String familienaam, BigDecimal
        wedde) {
        this.docentNr = docentNr;
        this.voornaam = voornaam;
        this.familienaam = familienaam;
        this.wedde = wedde;
    }

    public void opslag(BigDecimal bedrag) {
        wedde = wedde.add(bedrag);
    }

    @Override
    public String toString() {
        return "%d %s %s %s %s".formatted(docentNr, voornaam, familienaam,
           wedde, werkruimte);
    }

    public void addCampus(Campus campus) {
        campussen.add(campus);
    }

    public void removeCampus(Campus campus) {
        campussen.remove(campus);
    }

    public Set<Campus> getCampussen() {
        return Collections.unmodifiableSet(campussen);
    }

}
```

- Onze launch file `main/StartUp` ziet er dan zo uit:

```
package main;

import java.math.BigDecimal;

import domein.Campus;
import domein.Docent;
import domein.Werkruimte;
import jakarta.persistence.EntityManager;
import util.JPAUtil;
```

```
public class MAINoef4 {
    public static void main(String args[]) {
        Docent jan = new Docent(123, "Jan", "Baard", new BigDecimal(8000));
        Docent piet = new Docent(456, "Piet", "Baard", new BigDecimal(10000));
        Docent joris = new Docent(789, "Joris", "ZonderBaard", new BigDecimal
            (12000));
        Campus gent = new Campus("Gent");
        Campus aalst = new Campus("Aalst");

        Werkruimte zolder = new Werkruimte("SCH123", "zolder", 12, 6);
        Werkruimte kelder = new Werkruimte("SCH555", "kelder", 4, 4);
        Werkruimte dak = new Werkruimte("AA222", "dak", 10, 2);

        jan.addCampus(gent);
        piet.addCampus(gent);
        joris.addCampus(gent);

        jan.addCampus(aalst);
        joris.addCampus(aalst);

        jan.setWerkruimte(zolder);
        piet.setWerkruimte(zolder);
        joris.setWerkruimte(dak);

        EntityManager entityManager = JPAUtil.getEntityManagerFactory().
            createEntityManager();
        entityManager.getTransaction().begin();
        entityManager.persist(jan);
        entityManager.persist(piet);
        entityManager.persist(joris);
        entityManager.persist(gent);
        entityManager.persist(aalst);
        entityManager.persist(zolder);
        entityManager.persist(kelder);
        entityManager.persist(dak);

        entityManager.getTransaction().commit();
        entityManager.close();
        JPAUtil.getEntityManagerFactory().close();
    }

}
```

- Wanneer je deze file dan runt, moet je kijken naar MySQL en ga je merken dat er een tussentabel is verschijnen tussen **Werknemer** en **Campus** en dat hij ook is opgevuld