



samen sterk voor werk

# JPA met Hibernate Takenbundel



Deze cursus is eigendom van de VDAB©

## Inhoudsopgave

<b>1</b>	<b>TAKEN .....</b>	<b>4</b>
1.1	Alles voor de keuken .....	4
1.2	Artikel .....	4
1.3	Zoeken op nummer .....	4
1.4	JPA project - Dali .....	4
1.5	Toevoegen .....	4
1.6	Zoeken op naam .....	4
1.7	Prijsverhoging .....	4
1.8	Food en Non-Food artikels .....	4
1.9	Kortingen .....	5
1.10	Artikelgroepen .....	5
1.11	Artikellijst .....	6
1.12	Muziek .....	6
<b>2</b>	<b>VOORBEELDOPLOSSINGEN .....</b>	<b>7</b>
2.1	Alles voor de keuken .....	7
2.1.1	application.properties .....	7
2.1.2	DataSourceTest .....	7
2.2	Artikel .....	7
2.2.1	Artikel .....	7
2.3	Zoeken op nummer .....	7
2.3.1	application.properties .....	7
2.3.2	ArtikelRepository .....	7
2.3.3	JpaArtikelRepository .....	7
2.3.4	insertArtikel.sql .....	7
2.3.5	JpaArtikelRepositoryTest .....	8
2.3.6	JpaArtikelRepository .....	8
2.4	JPA project – Dali .....	8
2.5	Toevoegen .....	8
2.5.1	Artikel .....	8
2.5.2	ArtikelRepository .....	8
2.5.3	JpaArtikelRepository .....	9
2.5.4	JpaArtikelRepositoryTest .....	9

2.5.5	JpaArtikelRepository .....	9
2.6	Zoeken op naam .....	9
2.6.1	ArtikelRepository .....	9
2.6.2	JpaArtikelRepository .....	9
2.6.3	JpaArtikelRepositoryTest .....	9
2.6.4	orm.xml .....	10
2.6.5	JpaArtikelRepository .....	10
2.7	Prijsverhoging .....	10
2.7.1	ArtikelRepository .....	10
2.7.2	JpaArtikelRepository .....	10
2.7.3	JpaArtikelRepositoryTest .....	10
2.7.4	orm.xml .....	10
2.7.5	JpaArtikelRepository .....	10
2.8	Food en non-food artikels .....	11
2.8.1	Artikel .....	11
2.8.2	FoodArtikel .....	11
2.8.3	NonFoodArtikel .....	11
2.8.4	insertArtikel.sql .....	11
2.8.5	JpaArtikelRepositoryTest .....	11
2.9	Kortingen .....	13
2.9.1	Korting .....	13
2.9.2	KortingTest .....	13
2.9.3	Artikel: uitbreiding .....	14
2.9.4	insertArtikel.sql .....	14
2.9.5	JpaArtikelRepositoryTest .....	14
2.10	Artikelgroepen .....	14
2.10.1	Artikelgroep .....	14
2.10.2	Artikel .....	15
2.10.3	FoodArtikel .....	15
2.10.4	NonFoodArtikel .....	15
2.10.5	ArtikelGroepTest .....	16
2.10.6	ArtikelTest .....	16
2.10.7	insertArtikelGroep.sql .....	16
2.10.8	insertArtikel.sql .....	17
2.10.9	JpaArtikelRepositoryTest .....	17
2.11	Artikellijst .....	17
2.11.1	JpaArtikelRepositoryTest .....	17
2.11.2	Artikel .....	17
2.11.3	JpaArtikelRepository .....	18
2.12	Muziek .....	18
2.12.1	pom.xml .....	18
2.12.2	application.properties .....	18
2.12.3	DataSourceTest .....	18
2.12.4	orm.xml .....	18
2.12.5	Artiest .....	19
2.12.6	Track .....	19
2.12.7	Album .....	19

2.12.8	AlbumRepository .....	20
2.12.9	JpaAlbumRepository .....	20
2.12.10	insertArtiest.sql.....	20
2.12.11	insertAlbum.sql.....	20
2.12.12	JpaAlbumRepositoryTest .....	20
2.12.13	JpaAlbumRepository .....	21
2.12.14	AlbumService .....	21
2.12.15	DefaultAlbumService .....	21
2.12.16	IndexController .....	22
2.12.17	index.jsp.....	22
2.12.18	AlbumController .....	22
2.12.19	album.jsp .....	23

<b>3</b>	<b>COLOFON.....</b>	<b>24</b>
----------	---------------------	-----------

## 1 TAKEN

### 1.1 Alles voor de keuken

Je voert het script `AllesVoorDeKerken.sql` uit.  
Dit script maakt een database `allesvoordekeuken`.  
Je kan deze database openen met de gebruiker `cursist`, paswoord `cursist`

Deze database bevat een lege table `artikels`  
Je voegt met de MySQL Workbench enkele records  
(uit de categorieën fruit, groente, keukenapparaten)  
toe aan deze table (later komt er een gerelateerde  
table bij met deze categorieën)



artikels	
id	INT
naam	VARCHAR(50)
aankoop prijs	DECIMAL(10,2)
verkoop prijs	DECIMAL(10,2)

Je maakt een project.

Je test of de DataSource correct werkt.

### 1.2 Artikel

Je maakt een entity class `Artikel`, die bij de table `artikels` hoort.

### 1.3 Zoeken op nummer

Je maakt een interface `ArtikelRepository` en een implementatie `JpaArtikelRepository`.

Je maakt in de repositories layer de code om een artikel te zoeken op nummer.

Je schrijft een test voor deze code.

### 1.4 JPA project - Dali

Je maakt van je project een JPA project en je gebruikt Dali.

### 1.5 Toevoegen

Je maakt in de repositories layer de code om een artikel toe te voegen.

Je schrijft een test voor deze code.

### 1.6 Zoeken op naam

Je maakt in de repositories layer de code om de artikels te zoeken die in hun naam een woord bevatten. Dit woord wordt als parameter meegegeven bij het oproepen van de repositories layer.

Je sorteert de gevonden artikels op naam.

Je schrijft tests voor deze code.

### 1.7 Prijsverhoging

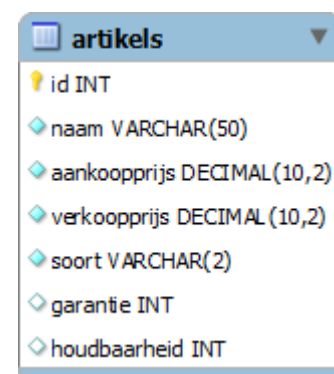
Je voegt aan de repositories layer code toe om de prijs van alle artikels met een percentage te verhogen. Je schrijft tests voor deze code.

### 1.8 Food en Non-Food artikels

Je voert het script `FoodNonFood.sql` uit.

Dit script voegt aan de table `artikels` drie kolommen toe

- `soort`  
bevat F bij food artikels  
bevat NF bij non-food artikels
- `garantie`  
blijft leeg bij food artikels  
bevat het aantal maanden garantie bij non-food artikels
- `houdbaarheid`  
bevat de houdbaarheid (in dagen) bij food artikels.  
blijft leeg bij non-food artikels



artikels	
id	INT
naam	VARCHAR(50)
aankoop prijs	DECIMAL(10,2)
verkoop prijs	DECIMAL(10,2)
soort	VARCHAR(2)
garantie	INT
houdbaarheid	INT

Je doet via de MySQL Workbench volgende aanpassingen

1. Je vult bij de bestaande records de kolom soort met F of NF
2. Je vult de kolom houdbaarheid in bij records met F in in de kolom soort
3. Je vult de kolom garantie in bij records met NF in de kolom soort

Je maakt via inheritance een onderscheid tussen food en non-food artikels.

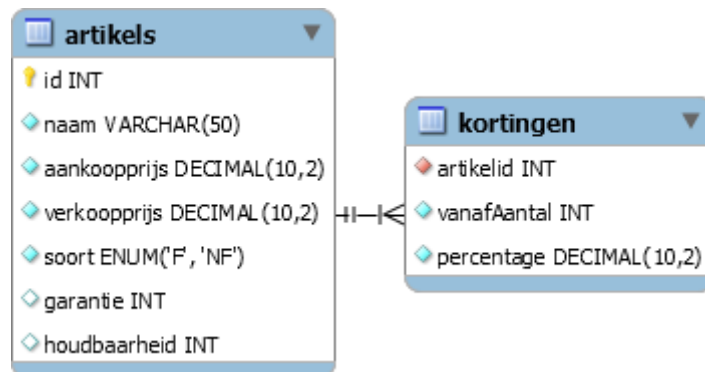
Je maakt Artikel abstract en maak je twee derived classes: FoodArtikel en NonFoodArtikel

- FoodArtikel heeft een int attribuut houdbaarheid
- NonFoodArtikel heeft een int attribuut garantie

Je voegt aan de repositories layer code toe om een artikel toe te voegen. Je test deze code.

## 1.9 Kortingen

Je voert het script  
*Kortingen.sql* uit.  
Dit script voegt een table  
kortingen toe



Deze table bevat hoeveelheidskortingen per artikel.

Een record met artikelid 1, vanafaantal 5 en percentage 2 betekent:  
bij het kopen van artikel 1 krijg je 2% korting vanaf 5 stuks.

Je maakt een bijbehorende value object class Korting. Je werkt equals en hashCode uit.

Je zal nooit *alle* Korting objecten die je leest uit de database in één Set opnemen. Dit is zinloos.

Je zal *per artikel* de Korting objecten in een Set opnemen. Houd hier rekening mee bij het uitwerken van equals en hashCode. Je test de equals en hashCode.

Artikel bevat een Set met een verzameling Korting objecten.

Je test of je de kortingen kan lezen uit de database.

## 1.10 Artikelgroepen

De database artikels bevat een table artikelgroepen



Je doet via de MySQL Workbench volgende aanpassingen

1. Je voegt enkele records toe aan deze table.
2. Je voert het script ArtikelGroepBijArtikel.sql uit.  
De table artikels bevat nu een extra kolom: artikelgroepId.
3. Je vult bij elk record die kolom met een artikelgroepid uit de tabel artikelgroepen
4. Je voert het script ForeignKey.sql uit.  
Dit script maakt van de kolom artikelgroepid in de table artikels een foreign key die verwijst naar de primary key kolom id in de table artikelgroepen.

Je maakt een nieuwe entity class ArtikelGroep.

Je definieert ook een bidirectionele één op veel associatie tussen deze class en Artikel.

Je test de associatie vanuit het standpunt van ArtikelGroep in ArtikelGroepTest.

Je test associatie vanuit het standpunt van Artikel in ArtikelTest.

Je zorgt dat `JpaArtikelRepositoryTest` werkt en je voert een method `artikelGroepLazyLoaded` toe.

Je hoeft geen `JpaArtikelGroepRepositoryTest` te schrijven.

### 1.11 Artikellijst

Je breidt in `JpaArtikelRepositoryTest` de test `findByNaamContains` uit.

Je toon van elk gevonden artikel de naam en de naam van de bijbehorende artikelgroep.

Je lost het  $n + 1$  probleem op dat hierbij optreedt.

Je bewijst met de test dat het probleem opgelost is.

### 1.12 Muziek

Je voert het script `Muziek.sql` uit. Dit script maakt een database muziek.

Je kan deze database openen met de gebruiker `cursist`, paswoord `cursist`



Je maakt een website.

Je toont op de welkompagina een alfabetische lijst van alle albums.

Je toont per album de naam van het album en de naam van de bijbehorende artiest.

Elke albumnaam is een hyperlink. Als de gebruiker deze aanklikt, toon je op een nieuwe pagina de albumtitel en de bijbehorende artiest. Je toont daar onder een lijst met de tracks van dat album.

Je toont per track de naam en de tijd. Je toont onder de tracks de tijd van het album (de totale tijd van de tracks van dat album).

## 2 VOORBEELDOPLOSSINGEN

### 2.1 Alles voor de keuken

#### 2.1.1 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/allesvoordekeuken?useSSL=false
spring.datasource.username=cursist
spring.datasource.password=cursist
```

#### 2.1.2 DataSourceTest

Zelfde als in theorie.

### 2.2 Artikel

#### 2.2.1 Artikel

```
package be.vdab.allesvoordekeuken.entities;
// enkele imports
@Entity
@Table(name = "artikels")
public class Artikel implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    private long id;
    private String naam;
    private BigDecimal aankoopprijs;
    private BigDecimal verkoopprijs;
    // getters voor id, naam, aankoopprijs en verkoopprijs
}
```

### 2.3 Zoeken op nummer

#### 2.3.1 application.properties

Extra regels:

```
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
spring.jpa.hibernate.naming.physical-strategy=\
org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

#### 2.3.2 ArtikelRepository

```
package be.vdab.allesvoordekeuken.repositories;
public interface ArtikelRepository {
    Optional<Artikel> read(long id);
}
```

#### 2.3.3 JpaArtikelRepository

```
package be.vdab.allesvoordekeuken.repositories;
// enkele imports
@Repository
class JpaArtikelRepository implements ArtikelRepository {
    @Override
    public Optional<Artikel> read(long id) {
        throw new UnsupportedOperationException();
    }
}
```

#### 2.3.4 insertArtikel.sql

```
insert into artikels(naam,aankoopprijs,verkoopprijs) values('test',100,120);
```



### 2.3.5 JpaArtikelRepositoryTest

```
package be.vdab.allesvoordekeuken.repositories;
// enkele imports
@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
@Import(JpaArtikelRepository.class)
@Sql("/insertArtikel.sql")
public class JpaArtikelRepositoryTest
    extends AbstractTransactionalJUnit4SpringContextTests{
    @Autowired
    private JpaArtikelRepository repository;
    private long idVanTestArtikel() {
        return super.jdbcTemplate.queryForObject(
            "select id from artikels where naam='test'", Long.class);
    }
    @Test
    public void read() {
        Artikel artikel = repository.read(idVanTestArtikel()).get();
        assertEquals("test", artikel.getNaam());
    }
    @Test
    public void readOnbestaandArtikel() {
        assertFalse(repository.read(-1).isPresent());
    }
}
```

### 2.3.6 JpaArtikelRepository

```
package be.vdab.allesvoordekeuken.repositories;
// enkele imports
@Repository
class JpaArtikelRepository implements ArtikelRepository {
    private final EntityManager manager;
    JpaArtikelRepository(EntityManager manager) {
        this.manager = manager;
    }
    @Override
    public Optional<Artikel> read(long id) {
        return Optional.ofNullable(manager.find(Artikel.class, id));
    }
}
```

## 2.4 JPA project – Dali

Zelfde als in theorie.

## 2.5 Toevoegen

### 2.5.1 Artikel

Extra regel voor de private variabele id:

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

Geparametriseerde constructor (zonder id) en protected default constructor.

### 2.5.2 ArtikelRepository

Extra method declaratie:

```
void create(Artikel artikel);
```

### 2.5.3 JpaArtikelRepository

Extra method:

```
@Override
public void create(Artikel artikel) {
    throw new UnsupportedOperationException();
}
```

### 2.5.4 JpaArtikelRepositoryTest

Extra code:

```
private static final String ARTIKELS = "artikels";
private Artikel artikel;

@Before
public void before() {
    artikel = new Artikel("test2", BigDecimal.ONE, BigDecimal.TEN);
}

@Test
public void create() {
    int aantalArtikels = super.countRowsInTable(ARTIKELS);
    repository.create(artikel);
    assertEquals(aantalArtikels + 1, super.countRowsInTable(ARTIKELS));
    assertNotEquals(0, artikel.getId());
    assertEquals(1, super.countRowsInTableWhere(ARTIKELS, "id="+artikel.getId()));
}
```

### 2.5.5 JpaArtikelRepository

```
@Override
public void create(Artikel artikel) {
    manager.persist(artikel);
}
```

## 2.6 Zoeken op naam

### 2.6.1 ArtikelRepository

Extra method declaratie:

```
List<Artikel> findByNaamContains(String woord);
```

### 2.6.2 JpaArtikelRepository

Extra method:

```
@Override
public List<Artikel> findByNaamContains(String woord) {
    throw new UnsupportedOperationException();
}
```

### 2.6.3 JpaArtikelRepositoryTest

Extra method:

```
@Test
public void findBijNaamContains() {
    List<Artikel> artikels = repository.findByNaamContains("es");
    long aantalArtikels = super.jdbcTemplate.queryForObject(
        "select count(*) from artikels where naam like '%es'", Long.class);
    assertEquals(aantalArtikels, artikels.size());
    String vorigeNaam = "";
    for (Artikel artikel : artikels) {
        String naam = artikel.getNaam();
        assertTrue(naam.toLowerCase().contains("es"));
        assertTrue(naam.compareToIgnoreCase(vorigeNaam) >= 0);
        vorigeNaam = naam;
    }
}
```

### 2.6.4 orm.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
    http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd" version="2.1">
  <named-query name='Artikel.findByNaamContains'>
    <query>
      select a from Artikel a where a.naam like :zoals order by a.naam
    </query>
  </named-query>
</entity-mappings>
```

### 2.6.5 JpaArtikelRepository

```
@Override
public List<Artikel> findByNaamContains(String woord) {
    return manager.createNamedQuery("Artikel.findByNaamContains", Artikel.class)
        .setParameter("zoals", '%' + woord + '%').getResultList();
}
```

## 2.7 Prijsverhoging

### 2.7.1 ArtikelRepository

Extra method declaratie:

```
int prijsVerhoging(BigDecimal percentage);
```

### 2.7.2 JpaArtikelRepository

Extra method:

```
@Override
public int prijsVerhoging(BigDecimal percentage) {
    throw new IllegalArgumentException();
}
```

### 2.7.3 JpaArtikelRepositoryTest

Extra test:

```
@Test
public void prijsVerhoging() {
    int aantalAangepast = repository.prijsVerhoging(BigDecimal.TEN);
    assertEquals(super.countRowsInTable(ARTIKELS), aantalAangepast);
    BigDecimal nieuwePrijs = super.jdbcTemplate.queryForObject(
        "select verkoopprijs from artikels where id=?",
        BigDecimal.class, idVanTestArtikel());
    assertEquals(0, BigDecimal.valueOf(132).compareTo(nieuwePrijs));
}
```

### 2.7.4 orm.xml

```
<named-query name="Artikel.prijsverhoging">
  <query>
    update Artikel a set a.verkoopprijs = a.verkoopprijs * :factor
  </query>
</named-query>
```

### 2.7.5 JpaArtikelRepository

```
@Override
public int prijsVerhoging(BigDecimal percentage) {
    BigDecimal factor =
        BigDecimal.ONE.add(percentage.divide(BigDecimal.valueOf(100)));
    return manager.createNamedQuery("Artikel.prijsverhoging")
        .setParameter("factor", factor)
        .executeUpdate();
}
```

## 2.8 Food en non-food artikels

### 2.8.1 Artikel

```
package be.vdab.allesvoordekeuken.entities;
// enkele imports ...

@Entity
@Table(name = "artikels")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "soort")
public abstract class Artikel implements Serializable {
    ...
}
```

### 2.8.2 FoodArtikel

```
package be.vdab.allesvoordekeuken.entities;
// enkele imports ...

@Entity
@DiscriminatorValue("F")
public class FoodArtikel extends Artikel {
    private static final long serialVersionUID = 1L;
    private int houdbaarheid;
    // een geparametriseerde constructor en een protected default constructor
    // een getter voor houdbaarheid
}
```

### 2.8.3 NonFoodArtikel

```
package be.vdab.allesvoordekeuken.entities;
// enkele imports ...

@Entity
@DiscriminatorValue("NF")
public class NonFoodArtikel extends Artikel {
    private static final long serialVersionUID = 1L;
    private int garantie;
    // een geparametriseerde constructor en een protected default constructor
    // een getter voor garantie
}
```

### 2.8.4 insertArtikel.sql

```
insert into artikels(naam,aankoopprijs,verkoopprijs,houdbaarheid,soort)
values('testfood',100,120, 7, 'F');
insert into artikels(naam,aankoopprijs,verkoopprijs,garantie,soort)
values('testnonfood',100,120, 30, 'NF');
```

### 2.8.5 JpaArtikelRepositoryTest

```
package be.vdab.allesvoordekeuken.repositories;
// enkele imports

@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
@Import(JpaArtikelRepository.class)
@Sql("/insertArtikel.sql")
public class JpaArtikelRepositoryTest extends
AbstractTransactionalJUnit4SpringContextTests {
    private static final String ARTIKELS = "artikels";
    @Autowired
    private JpaArtikelRepository repository;
    private FoodArtikel foodArtikel;
    private NonFoodArtikel nonFoodArtikel;
```

```

@Before
public void before() {
    foodArtikel = new FoodArtikel("testfood2",BigDecimal.ONE,BigDecimal.TEN,7);
    nonFoodArtikel =
        new NonFoodArtikel("testnonfood2", BigDecimal.ONE, BigDecimal.TEN, 30);
}
private long idVanTestFoodArtikel() {
    return super.jdbcTemplate.queryForObject(
        "select id from artikels where naam='testfood'", Long.class);
}
private long idVanTestNonFoodArtikel() {
    return super.jdbcTemplate.queryForObject(
        "select id from artikels where naam='testnonfood'", Long.class);
}
@Test
public void readFoodArtikel() {
    FoodArtikel artikel =
        (FoodArtikel) repository.read(idVanTestFoodArtikel()).get();
    assertEquals("testfood", artikel.getNaam());
}
@Test
public void readNonFoodArtikel() {
    NonFoodArtikel artikel =
        (NonFoodArtikel) repository.read(idVanTestNonFoodArtikel()).get();
    assertEquals("testnonfood", artikel.getNaam());
}
@Test
public void readOnbestaandArtikel() {
    assertFalse(repository.read(-1).isPresent());
}
@Test
public void createFoodArtikel() {
    int aantalFoodArtikels = super.countRowsInTableWhere(ARTIKELS, "soort='F'");
    repository.create(foodArtikel);
    assertEquals(aantalFoodArtikels + 1,
        super.countRowsInTableWhere(ARTIKELS, "soort='F'"));
    assertEquals(1,
        super.countRowsInTableWhere(ARTIKELS, "id=" + foodArtikel.getId()));
}
@Test
public void createNonFoodArtikel() {
    int aantalNonFoodArtikels =
        super.countRowsInTableWhere(ARTIKELS, "soort='NF'");
    repository.create(nonFoodArtikel);
    assertEquals(aantalNonFoodArtikels + 1,
        super.countRowsInTableWhere(ARTIKELS, "soort='NF'"));
    assertEquals(1,
        super.countRowsInTableWhere(ARTIKELS, "id=" + nonFoodArtikel.getId()));
}
@Test
public void findBijNaamContains() {
    List<Artikel> artikels = repository.findByNaamContains("es");
    long aantalArtikels = super.jdbcTemplate.queryForObject(
        "select count(*) from artikels where naam like '%es'", Long.class);
    assertEquals(aantalArtikels, artikels.size());
    String vorigeNaam = "";
    for (Artikel artikel : artikels) {
        String naam = artikel.getNaam();
        assertTrue(naam.toLowerCase().contains("es"));
        assertTrue(naam.compareTo(vorigeNaam) >= 0);
    }
}

```

```

        vorigeNaam = naam;
    }
}
@Test
public void prijsVerhoging() {
    int aantalAangepast = repository.prijsVerhoging(BigDecimal.TEN);
    assertEquals(super.countRowsInTable("artikels"), aantalAangepast);
    BigDecimal nieuwePrijs = super.jdbcTemplate.queryForObject(
        "select verkoopprijs from artikels where id=?", BigDecimal.class,
        idVanTestFoodArtikel());
    assertEquals(0, BigDecimal.valueOf(132).compareTo(nieuwePrijs));
}
}

```

## 2.9 Kortingen

### 2.9.1 Korting

```

package be.vdab.allesvoordekeuken.valueobjects;
// enkele imports
@Embeddable
public class Korting implements Serializable {
    private static final long serialVersionUID = 1L;
    private int vanafAantal;
    private BigDecimal percentage;
    // een geparametriseerde constructor en een protected default constructor
    // getters voor vanafAantal en percentage
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Korting)) {
            return false;
        }
        Korting andereKorting = (Korting) object;
        return vanafAantal == andereKorting.vanafAantal;
    }
    @Override
    public int hashCode() {
        return vanafAantal;
    }
}

```

### 2.9.2 KortingTest

```

package be.vdab.allesvoordekeuken.valueobjects;
// enkele imports
public class KortingTest {
    private Korting korting1, nogEensKorting1, korting2;
    @Before
    public void before() {
        korting1 = new Korting(1, BigDecimal.ONE);
        nogEensKorting1 = new Korting(1, BigDecimal.ONE);
        korting2 = new Korting(2, BigDecimal.TEN);
    }
    @Test
    public void kortingenZijnGelijkAlsHunVanafAantallenGelijkZijn() {
        assertEquals(korting1, nogEensKorting1);
    }
    @Test
    public void kortingenZijnVerschillendAlsHunVanafAantallenVerschillen() {
        assertNotEquals(korting1, korting2);
    }
}

```

```

@Test
public void eenKortingIsNietGelijkAanNull() {
    assertNotEquals(korting1, null);
}

@Test
public void eenKortingIsNietGelijkAanEenAnderTypeObject() {
    assertNotEquals(korting1, "");
}

@Test
public void gelijkeKortingenGevenDezelfdeHashCode() {
    assertEquals(korting1.hashCode(), nogEensKorting1.hashCode());
}
}

```

### 2.9.3 Artikel: uitbreiding

```

@ElementCollection @OrderBy("vanafAantal")
@CollectionTable(name = "kortingen",
    joinColumns = @JoinColumn(name = "artikelid"))
private Set<Korting> kortingen;
public Set<Korting> getKortingen() {
    return Collections.unmodifiableSet(kortingen);
}

```

Extra opdracht in geparametriseerde constructor:

```
this.kortingen = new LinkedHashSet<>();
```

### 2.9.4 insertArtikel.sql

Extra opdracht:

```

insert into kortingen(artikelid,vanafAantal,percentage)
values ((select id from artikels where naam='testfood'), 1, 10);

```

### 2.9.5 JpaArtikelRepositoryTest

Extra method:

```

@Test
public void kortingenLezen() {
    Artikel artikel = repository.read(idVanTestFoodArtikel()).get();
    assertEquals(1, artikel.getKortingen().size());
    assertTrue(artikel.getKortingen().contains(new Korting(1, BigDecimal.TEN)));
}

```

## 2.10 Artikelgroepen

### 2.10.1 Artikelgroep

```

package be.vdab.allesvoordekeuken.entities;
// enkele imports

@Entity
@Table(name = "artikelgroepen")
public class ArtikelGroep implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String naam;
    @OneToMany(mappedBy = "artikelGroep")
    @OrderBy("naam")
    private Set<Artikel> artikels;
    // getters voor id en voor naam
    // geparametriseerde constructor met enkel naam, protected default constructor
    public Set<Artikel> getArtikels() {
        return Collections.unmodifiableSet(artikels);
    }
}

```

```

public boolean add(Artikel artikel) {
    if (artikel == null) {
        throw new NullPointerException();
    }
    boolean toegevoegd = artikels.add(artikel);
    ArtikelGroep oudeArtikelGroep = artikel.getArtikelGroep();
    if (oudeArtikelGroep != null && oudeArtikelGroep != this) {
        oudeArtikelGroep.artikels.remove(artikel);
    }
    if (oudeArtikelGroep != this) {
        artikel.setArtikelGroep(this);
    }
    return toegevoegd;
}
}

```

### 2.10.2 Artikel

Extra code:

```

@ManyToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "artikelgroepid")
private ArtikelGroep artikelGroep;
public ArtikelGroep getArtikelGroep() {
    return artikelGroep;
}
public void setArtikelGroep(ArtikelGroep artikelGroep) {
    if (artikelGroep == null) {
        throw new NullPointerException();
    }
    if (!artikelGroep.getArtikels().contains(this)) {
        artikelGroep.add(this);
    }
    this.artikelGroep = artikelGroep;
}
// equals en hashCode gebaseerd op naam

```

Aangepaste geparametriseerde constructor:

```

public Artikel(String naam, BigDecimal aankoopprijs, BigDecimal verkoopprijs,
    ArtikelGroep artikelGroep) {
    this.naam = naam;
    this.aankoopprijs = aankoopprijs;
    this.verkoopprijs = verkoopprijs;
    this.kortingen = new LinkedHashSet<>();
    setArtikelGroep(artikelGroep);
}

```

### 2.10.3 FoodArtikel

Aangepaste geparametriseerde constructor:

```

public FoodArtikel(String naam, BigDecimal aankoopprijs,
    BigDecimal verkoopprijs, int houdbaarheid, ArtikelGroep artikelGroep) {
    super(naam, aankoopprijs, verkoopprijs, artikelGroep);
    this.houdbaarheid = houdbaarheid;
}

```

### 2.10.4 NonFoodArtikel

Aangepaste geparametriseerde constructor:

```

public NonFoodArtikel(String naam, BigDecimal aankoopprijs,
    BigDecimal verkoopprijs, int garantie, ArtikelGroep artikelGroep) {
    super(naam, aankoopprijs, verkoopprijs, artikelGroep);
    this.garantie = garantie;
}

```



### 2.10.5 ArtikelGroepTest

```
package be.vdab.allesvoordekeuken.entities;
// enkele imports
public class ArtikelGroepTest {
    private ArtikelGroep groep1;
    private ArtikelGroep groep2;
    private Artikel artikel1;
    @Before
    public void before() {
        groep1 = new ArtikelGroep("test");
        groep2 = new ArtikelGroep("test2");
        artikel1 = new FoodArtikel("test",BigDecimal.ONE,BigDecimal.ONE,1,groep1);
    }
    @Test
    public void groep1MoetDeArtikelGroepZijnVanArtikel1() {
        assertEquals(groep1, artikel1.getArtikelGroep());
        assertEquals(1, groep1.getArtikels().size());
        assertTrue(groep1.getArtikels().contains(artikel1));
    }
    @Test
    public void artikel1VerhuistNaarGroep2() {
        assertTrue(groep2.add(artikel1));
        assertTrue(groep1.getArtikels().isEmpty());
        assertTrue(groep2.getArtikels().contains(artikel1));
        assertEquals(groep2, artikel1.getArtikelGroep());
    }
}
```

### 2.10.6 ArtikelTest

```
package be.vdab.allesvoordekeuken.entities;
// enkele imports
public class ArtikelTest {
    private Artikel artikel1;
    private ArtikelGroep groep1;
    private ArtikelGroep groep2;
    @Before
    public void before() {
        groep1 = new ArtikelGroep("test");
        groep2 = new ArtikelGroep("test2");
        artikel1 = new FoodArtikel("test",BigDecimal.ONE,BigDecimal.ONE,1,groep1);
    }
    @Test
    public void groep1EnArtikel1ZijnGoedVerbonden() {
        assertEquals(1, groep1.getArtikels().size());
        assertTrue(groep1.getArtikels().contains(artikel1));
        assertEquals(groep1, artikel1.getArtikelGroep());
    }
    @Test
    public void artikel1VerhuistNaarGroep2() {
        artikel1.setArtikelGroep(groep2);
        assertTrue(groep1.getArtikels().isEmpty());
        assertEquals(1, groep2.getArtikels().size());
        assertTrue(groep2.getArtikels().contains(artikel1));
        assertEquals(groep2, artikel1.getArtikelGroep());
    }
}
```

### 2.10.7 insertArtikelGroep.sql

```
insert into artikelgroepen(naam) values('test');
```

### 2.10.8 insertArtikel.sql

```
insert into
artikels(naam,aankoopprijs,verkoopprijs,houdbaarheid,soort,artikelgroepid)
values('testfood',100,120, 7, 'F',
(select id from artikelgroepen where naam='test'));
insert into
artikels(naam,aankoopprijs,verkoopprijs,garantie,soort,artikelgroepid)
values('testnonfood',100,120, 30, 'NF',
(select id from artikelgroepen where naam='test'));
insert into kortingen(artikelid,vanafAantal,percentage)
values ((select id from artikels where naam='testfood'), 1, 10);
```

### 2.10.9 JpaArtikelRepositoryTest

Extra opdracht voor @Sql("/insertArtikel.sql"):

@Sql("/insertArtikelGroep.sql")

Extra variabelen:

```
private ArtikelGroep groep;
@Autowired
private EntityManager manager;
```

Gewijzigde method before:

@Before

```
public void before() {
    groep = new ArtikelGroep("test");
    foodArtikel = new FoodArtikel("testfood", BigDecimal.ONE, BigDecimal.TEN, 7,
    groep);
    nonFoodArtikel = new NonFoodArtikel("testnonfood", BigDecimal.ONE,
    BigDecimal.TEN, 30, groep);
}
```

Extra opdracht vooraan in createFood:

```
manager.persist(artikelGroep);
```

Extra opdracht vooraan in createNonFood:

```
manager.persist(artikelGroep);
```

Extra method:

@Test

```
public void artikelGroepLazyLoaded() {
    Artikel artikel = repository.read(idVanTestFoodArtikel()).get();
    assertEquals("test", artikel.getArtikelGroep().getNaam());
}
```

## 2.11 Artikellijst

### 2.11.1 JpaArtikelRepositoryTest

Extra method na List<Artikel> artikels = repository.findByNaamContains("es");:

```
manager.clear();
```

Extra method als laatste in de for lus:

```
System.out.println(artikel.getNaam()+ ' '+artikel.getArtikelGroep().getNaam());
```

### 2.11.2 Artikel

Extra annotation:

```
@NamedEntityGraph(name = Artikel.MET_ARTIKELGROEP,
attributeNodes = @NamedAttributeNode("artikelGroep"))
```

Extra constante:

```
public static final String MET_ARTIKELGROEP = "Artikel.metArtikelGroep";
```

### 2.11.3 JpaArtikelRepository

```
@Override
public List<Artikel> findByNaamContains(String woord) {
    return manager.createNamedQuery("Artikel.findByNaamContains", Artikel.class)
        .setParameter("zoals", '%' + woord + '%')
        .setHint("javax.persistence.loadgraph",
            manager.createEntityGraph(Artikel.MET_ARTIKELGROEP))
        .getResultList();
}
```

## 2.12 Muziek

### 2.12.1 pom.xml

Extra dependencies:

```
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>jsp-api</artifactId>
  <version>[2.2,]</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```

### 2.12.2 application.properties

```
spring.datasource.url=jdbc:mysql://localhost/muziek?useSSL=false
spring.datasource.username=cursist
spring.datasource.password=cursist
spring.mvc.view.prefix:/WEB-INF/JSP/
spring.mvc.view.suffix:.jsp
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
spring.jpa.hibernate.naming.physical-strategy=\
org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

### 2.12.3 DataSourceTest

Zelfde als in theorie.

### 2.12.4 orm.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
    http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd"
  version="2.1">
  <named-query name='Album.findAll'>
    <query>
      select a from Album a
      order by a.naam
    </query>
  </named-query>
</entity-mappings>
```

### 2.12.5 Artiest

```
package be.vdab.muziek.entities;
// enkele imports ...
@Entity
@Table(name = "artiesten")
public class Artiest implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String naam;
    // getters voor id en naam, protected default constructor
}
```

### 2.12.6 Track

```
package be.vdab.muziek.valueobjects;
// enkele imports ...
@Embeddable
public class Track implements Serializable {
    private static final long serialVersionUID = 1L;
    private String naam;
    private BigDecimal tijd;
    // getters voor naam en tijd, protected default constructor
    // equals en hashCode gebaseerd op naam
}
```

### 2.12.7 Album

```
package be.vdab.muziek.entities;
// enkele imports ...
@Entity
@Table(name = "albums")
@NamedEntityGraph(name = Album.MET_ARTIEST,
    attributeNodes = @NamedAttributeNode("artiest") )
public class Album implements Serializable {
    private static final long serialVersionUID = 1L;
    public static final String MET_ARTIEST = "Album.metArtiest";
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String naam;
    @ManyToOne(optional = false, fetch = FetchType.LAZY)
    @JoinColumn(name = "artiestid")
    private Artiest artiest;
    @ElementCollection
    @CollectionTable(name="tracks", joinColumns=@JoinColumn(name = "albumid"))
    private Set<Track> tracks;
    // getters voor id, naam en artiest, protected default constructor
    public Set<Track> getTracks() {
        return Collections.unmodifiableSet(tracks);
    }
    public BigDecimal getTijd() {
        return tracks.stream().map(track->track.getTijd())
            .reduce(BigDecimal.ZERO, (vorigTotaal, huidigeWaarde)->
                vorigTotaal.add(huidigeWaarde));
    }
}
```

### 2.12.8 AlbumRepository

```
package be.vdab.muziek.repositories;
// enkele imports
public interface AlbumRepository {
    List<Album> findAll();
    Optional<Album> read(long id);
}
```

### 2.12.9 JpaAlbumRepository

```
package be.vdab.muziek.repositories;
// enkele imports
@Repository
class JpaAlbumRepository implements AlbumRepository {
    @Override
    public List<Album> findAll() {
        throw new UnsupportedOperationException();
    }
    @Override
    public Optional<Album> read(long id) {
        throw new UnsupportedOperationException();
    }
}
```

### 2.12.10 insertArtiest.sql

```
insert into artiesten(naam) values ('test');
```

### 2.12.11 insertAlbum.sql

```
insert into albums(naam,artiestid)
values ('test', (select id from artiesten where naam='test'));
insert into tracks(naam,tijd,albumid)
values ('test',10,(select id from albums where naam='test'));
```

### 2.12.12 JpaAlbumRepositoryTest

```
package be.vdab.muziek.repositories;
// enkele imports
@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
@Import(JpaAlbumRepository.class)
@Sql("/insertArtiest.sql")
@Sql("/insertAlbum.sql")
public class JpaAlbumRepositoryTest
    extends AbstractTransactionalJUnit4SpringContextTests {
    private static final String ALBUMS = "albums";
    @Autowired
    private JpaAlbumRepository repository;
    @Autowired
    private EntityManager manager;
    private long idVanTestAlbum() {
        return super.jdbcTemplate.queryForObject(
            "select id from albums where naam='test'", Long.class);
    }
    @Test
    public void read() {
        Album album = repository.read(idVanTestAlbum()).get();
        assertEquals("test", album.getNaam());
        assertEquals("test", album.getArtiest().getNaam());
        assertEquals(0, BigDecimal.TEN.compareTo(album.getTijd()));
    }
}
```

```

@Test
public void findAll() {
    List<Album> albums = repository.findAll();
    manager.clear();
    assertEquals(super.countRowsInTable(ALBUMS), albums.size());
    String vorigeNaam = "";
    for (Album album : albums) {
        assertTrue(album.getNaam().compareToIgnoreCase(vorigeNaam) >= 0);
        vorigeNaam = album.getNaam();
        System.out.println(album.getNaam() + ' ' + album.getArtiest().getNaam());
    }
}

```

### 2.12.13 JpaAlbumRepository

```

package be.vdab.muziek.repositories;
// enkele imports
@Repository
class JpaAlbumRepository implements AlbumRepository {
    private final EntityManager manager;
    JpaAlbumRepository(EntityManager manager) {
        this.manager = manager;
    }
    @Override
    public List<Album> findAll() {
        return manager.createNamedQuery("Album.findAll", Album.class)
            .setHint("javax.persistence.loadgraph",
                manager.createEntityGraph(Album.MET_ARTIEST))
            .getResultList();
    }
    @Override
    public Optional<Album> read(long id) {
        return Optional.ofNullable(manager.find(Album.class, id));
    }
}

```

### 2.12.14 AlbumService

```

package be.vdab.muziek.services;
// enkele imports
public interface AlbumService {
    List<Album> findAll();
    Optional<Album> read(long id);
}

```

### 2.12.15 DefaultAlbumService

```

package be.vdab.muziek.services;
// enkele imports ...
@Service
class DefaultAlbumService implements AlbumService {
    private final AlbumRepository albumRepository;
    DefaultAlbumService(AlbumRepository albumRepository) {
        this.albumRepository = albumRepository;
    }
    @Override
    public List<Album> findAll() {
        return albumRepository.findAll();
    }
    @Override
    public Optional<Album> read(long id) {
        return albumRepository.read(id);
    }
}

```

```

    }
}

```

### 2.12.16 IndexController

```

package be.vdab.muziek.web;
// enkele imports
@Controller
@RequestMapping("/")
class IndexController {
    private final static String VIEW = "index";
    private final AlbumService albumService;
    IndexController(AlbumService albumService) {
        this.albumService = albumService;
    }
    @GetMapping
    ModelAndView albums() {
        return new ModelAndView(VIEW, "albums", albumService.findAll());
    }
}

```

### 2.12.17 index.jsp

```

<%@ page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<%@taglib prefix='spring' uri='http://www.springframework.org/tags' %>
<!doctype html>
<html lang='nl'>
<head>
<title>Albums</title>
</head>
<body>
    <c:if test="${not empty param.fout}">
        <div>${param.fout}</div>
    </c:if>
    <h1>Albums</h1>
    <ul>
    <c:forEach var='album' items='${albums}'>
        <spring:url var='url' value='/albums/{id}'>
            <spring:param name='id' value='${album.id}'/>
        </spring:url>
        <li><a href='${url}'>${album.naam}</a> ${album.artiest.naam}</li>
    </c:forEach>
    </ul>
</body>
</html>

```

### 2.12.18 AlbumController

```

package be.vdab.muziek.web;
// enkele imports
@Controller
@RequestMapping("albums")
class AlbumController {
    private final static String VIEW="album";
    private final static String REDIRECT_ALBUM_NIET_GEVONDEN = "redirect:/";
    private final AlbumService albumService;
    AlbumController(AlbumService albumService) {
        this.albumService = albumService;
    }
}

```

```

@GetMapping("{id}")
ModelAndView album(@PathVariable long id, RedirectAttributes redirectAttributes) {
    Optional<Album> album = albumService.read(id);
    if (album.isPresent()) {
        return new ModelAndView(VIEW).addObject(album.get());
    }
    redirectAttributes.addAttribute("fout", "album niet gevonden");
    return new ModelAndView(REDIRECT_ALBUM_NIET_GEVONDEN);
}
}

```

### 2.12.19 album.jsp

```

<%@ page contentType='text/html' pageEncoding='UTF-8' session='false'%>
<%@taglib prefix='c' uri='http://java.sun.com/jsp/jstl/core'%>
<!doctype html>
<html lang='nl'>
<head>
<title>${album.naam} - ${album.artiest.naam}</title>
</head>
<body>
    <h1>${album.naam} - ${album.artiest.naam}</h1>
    <ul>
    <c:forEach var="track" items="${album.tracks}">
        <li>${track.naam} ${track.tijd}</li>
    </c:forEach>
    </ul>
    Totale tijd: ${album.tijd}
</body>
</html>

```



### 3 COLOFON

<b>Domeinexpertisemanager:</b>	Jean Smits
<b>Moduleverantwoordelijke:</b>	Hans Desmet
<b>Medewerkers:</b>	Hans Desmet
<b>Versie:</b>	27/4/2018
<b>Nummer dotatielijst:</b>	