Java

Les 7

jeroen.devos01@ap.be

AP HOGESCHOOL ANTWERPEN

AP.BE


IT'S MONDAY AGAIN
HERE WE GO!
makeameme.org

Lesdoelen

- JDBC
- ORM
- JPA
- Hibernate
- HQL

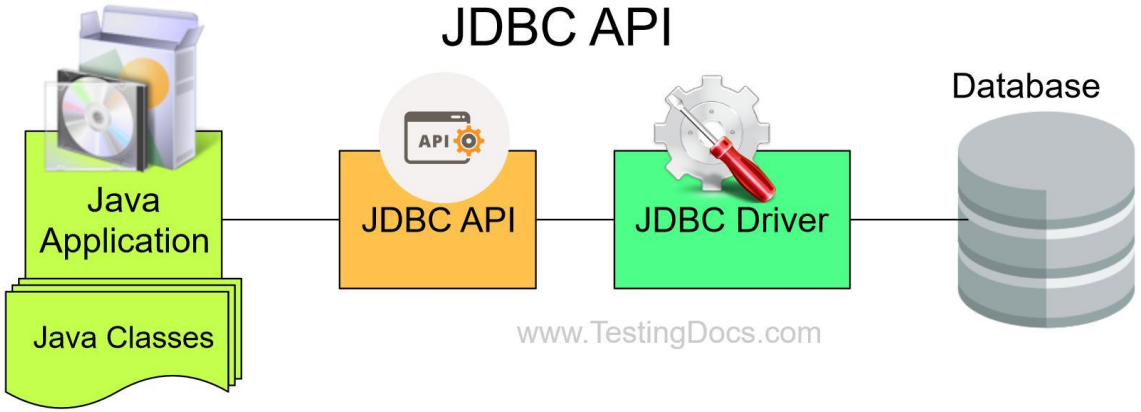
Advanced Java™

Theorie - JDBC

Java **D**atab**b**ase **C**onnectivity

Twee onderdelen:

- JDBC Driver : specifieke driver per database type, bv mysql driver
- JDBC API : generieke methods om database aan te spreken



JDBC - tekortkomingen

- Low level
- Heel veel code nodig
- Geen mapping tussen Java Objecten en database gegevens

Theorie - ORM

Object **R**elational **M**apping

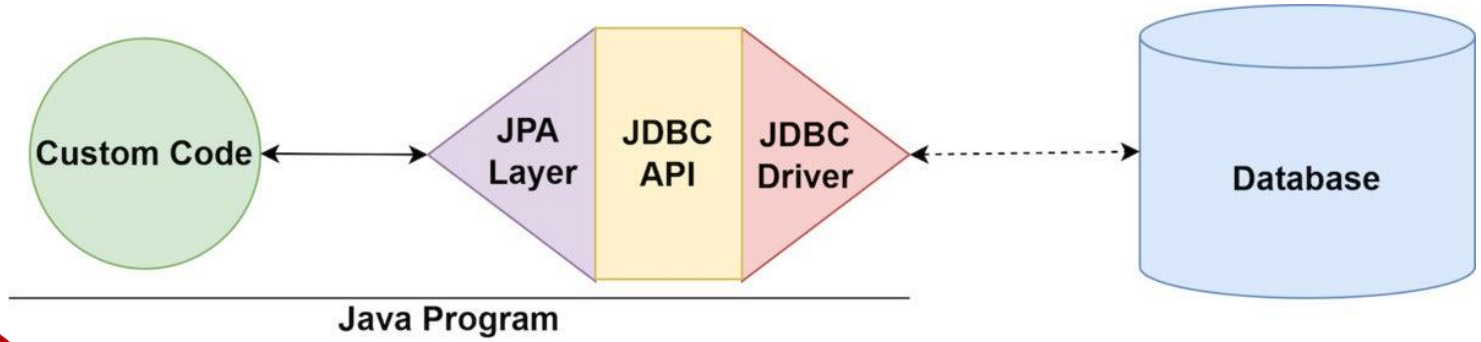
- Beschrijft hoe Objecten op een Database tabel moeten gemapped worden
- Gebruikt metadata zoals xml of annotation
- SQL wordt afgeschermd
- Taalonafhankelijk, wordt in Java en C# gebruikt

Theorie - JPA

Java **P**ersistence **A**PI

= specificatie, geen implementatie !

- Beschrijft ORM voor Java Objecten
- Legt een ‘laag’ rond JDBC

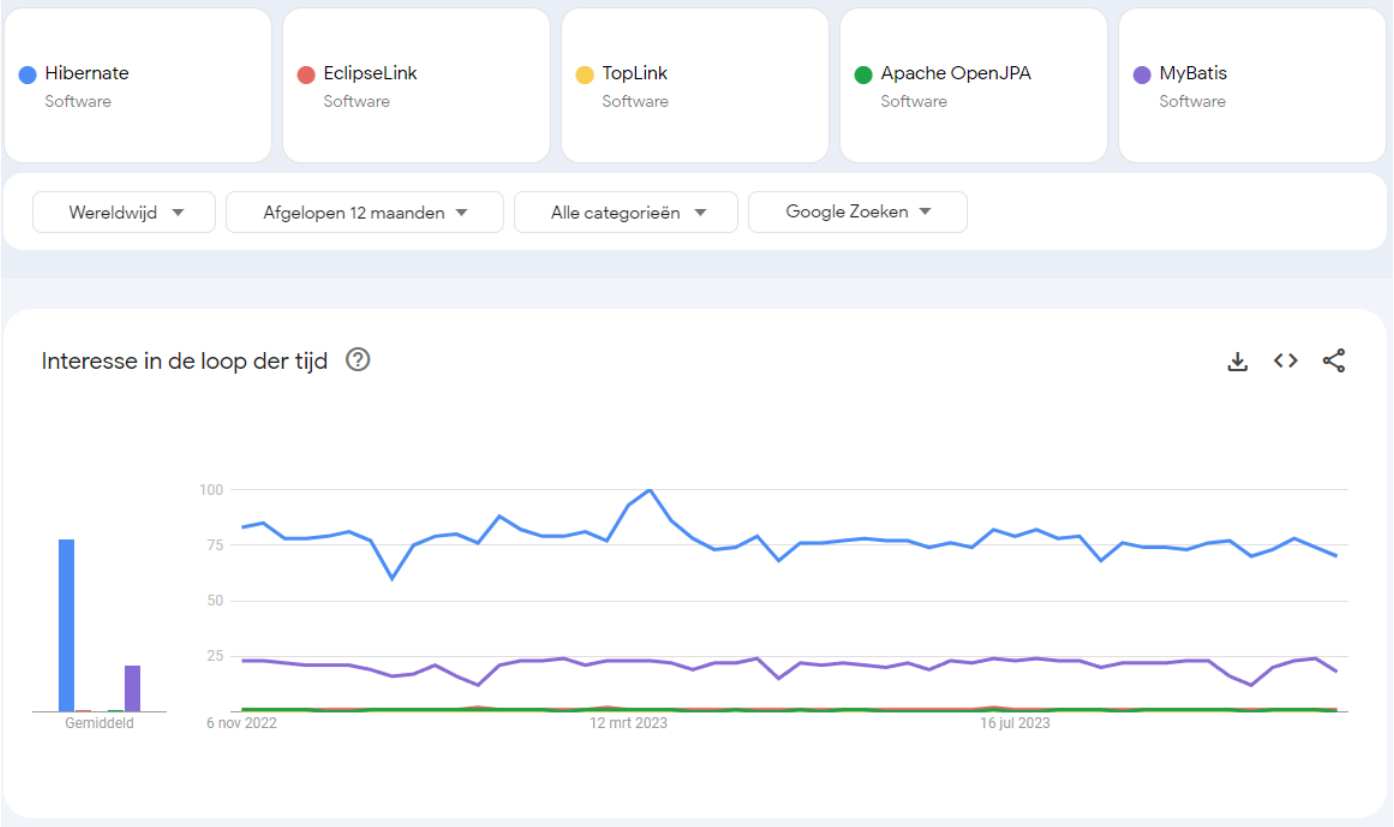


Theorie – Java ORM Frameworks

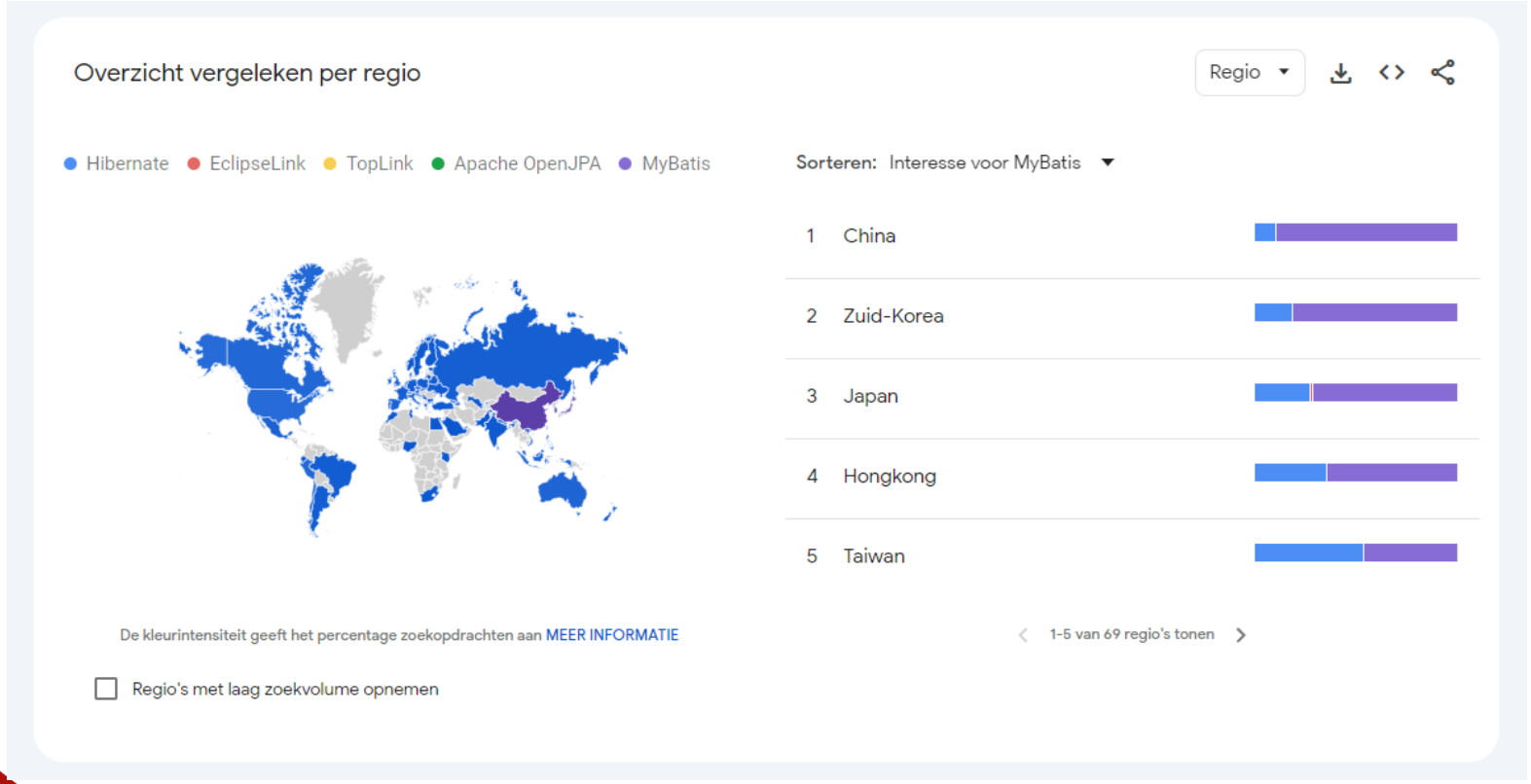
Bekende Java ORM Frameworks:

- Hibernate
- EclipseLink
- Oracle TopLink
- Apache OpenJPA
- MyBatis

Theorie – Java ORM Frameworks



Theorie – Java ORM Frameworks



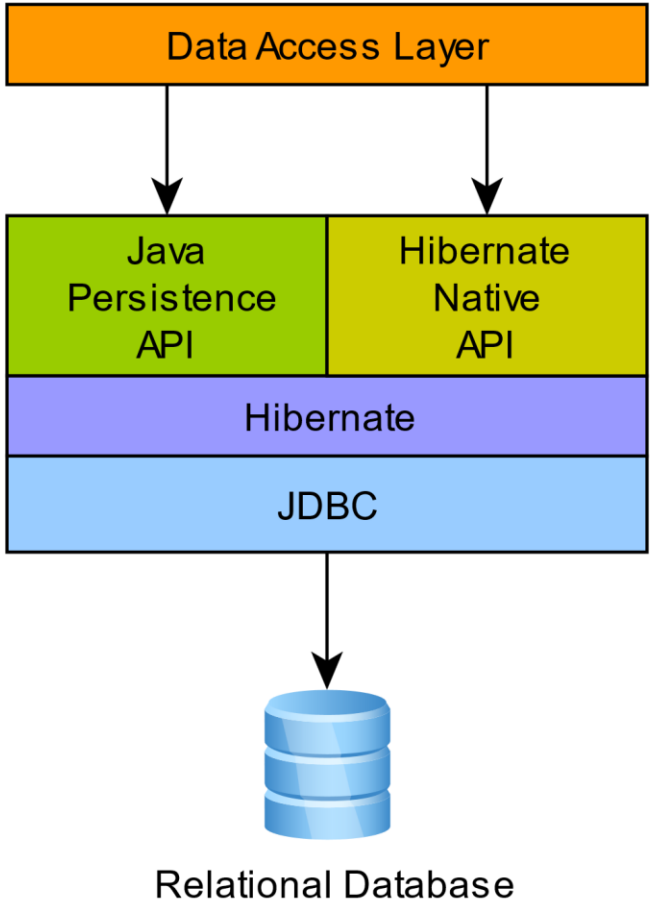
Hibernate

Hibernate

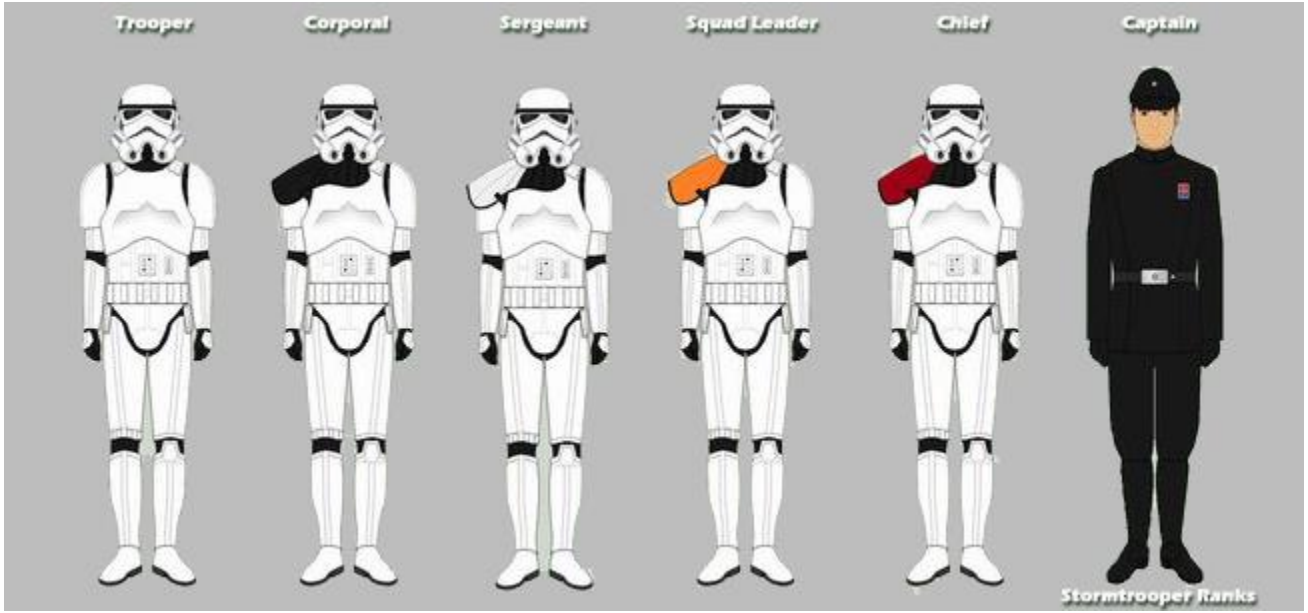
- Open source
- Heel populair – bijna standaard
- ORM
- Implementeert JPA
- Native API

https://hibernate.org/

Hibernate


Data Access Layer
Java Persistence API
Hibernate Native API
Hibernate
JDBC
Relational Database

Hands on with hibernate



Hibernate – Stappenplan

1. Maven dependencies toevoegen
   - hibernate
   - database
2. Hibernate database configuratie schrijven
3. Java Object mapping schrijven
4. SessionFactory schrijven
5. Data Access Object (DAO) schrijven

Hibernate – Stappenplan

1. **Maven dependencies toevoegen**
   - **hibernate**
   - **database**
2. Hibernate database configuratie schrijven
3. Java Object mapping schrijven
4. SessionFactory schrijven
5. Data Access Object (DAO) schrijven

Hibernate Maven dependency

```
<dependency>
   <groupId>org.hibernate.orm</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>6.3.1.Final</version>
</dependency>
```

- hibernate-core bevat de nodige libraries om hibernate te gebruiken.

Database MySQL Maven dependency

```
<dependency>
   <groupId>com.mysql</groupId>
   <artifactId>mysql-connector-j</artifactId>
   <version>8.0.33</version>
</dependency>
```

- mysql-connector is de driver om met een MySQL database te communiceren.
- MySQL moet nog wel apart geïnstalleerd worden.

Database H2 Maven dependency

```
<dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <version>2.2.224</version>
</dependency>
```

- H2 is een Java in-memory SQL database.
- Geen extra dependency op een geïnstalleerde database : ideaal voor testing (en toetsing).

Hibernate – Stappenplan

1. Maven dependencies toevoegen
   - hibernate
   - database
2. **Hibernate database configuratie schrijven**
3. Java Object mapping schrijven
4. SessionFactory schrijven
5. Data Access Object (DAO) schrijven

Hibernate database configuration

- Eenmalig aan te maken voor je database

- Twee mogelijkheden voor de configuratie:
  - Configuratie in XML bestand : hibernate.cfg.xml
  - Configuratie in Java code

- XML bestand komt in de /resources folder

MySQL XML database configuration

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property name="hibernate.dialect">
         org.hibernate.dialect.MySQLDialect
      </property>
      <property name="hibernate.connection.driver_class">
         com.mysql.jdbc.Driver
      </property>
      <property name="hibernate.connection.url">
         jdbc:mysql://localhost:3306/mydatabase
      </property>
      <property name="hibernate.connection.username">
         username
      </property>
      <property name="hibernate.connection.password">
         password
      </property>
   </session-factory>
</hibernate-configuration>
```

H2 XML database configuration

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property name="hibernate.dialect">
         org.hibernate.dialect.H2Dialect
      </property>
      <property name="hibernate.connection.driver_class">
         org.h2.Driver
      </property>
      <property name="hibernate.connection.url">
         jdbc:h2:./data/mydatabase
      </property>
      <property name="hibernate.connection.username">
         sa
      </property>
      <property name="hibernate.connection.password">

      </property>
   </session-factory>
</hibernate-configuration>
```

Default username = sa
Default password = “”

Hibernate basic configuration

- **hibernate.dialect** : elke database heeft een eigen SQL variant, deze property zorgt ervoor dat hibernate het juiste ‘dialect’ spreekt.
- **hibernate.connection.driver_class** : JDBC driver voor de database
- **hibernate.connection.url** : JDBC URL naar de database
- **hibernate.connection.username** : database gebruikersnaam
- **hibernate.connection.password** : database wachtwoord

Hibernate extra configuration

- **hibernate.connection.pool_size** : Aantal connecties in de connection pool. Zet voor development deze op 1.
- **hibernate.show_sql** : true/false : toont de SQL queries. Nuttig voor development!
- **hibernate.hbm2ddl.auto** : bepaalt het al dan niet aanmaken van het database schema. Zet voor development op ‘create’.
  - create : maak telkens bij opstart de database opnieuw aan
  - validate : valideer het schema bij opstart, doe geen aanpassingen
  - update : pas bij opstart het schema aan
  - none (default) : doe niets met schema bij opstart

MySQL code database configuration

```
try {
   Configuration configuration = new Configuration();

   Properties properties = new Properties();
   properties.put(Environment.DIALECT, "org.hibernate.dialect.MySQLDialect");
   properties.put(Environment.DRIVER, "com.mysql.jdbc.Driver");
   properties.put(Environment.URL, "jdbc:mysql://localhost:3306/mydatabase");
   properties.put(Environment.USER, "username");
   properties.put(Environment.PASS, "password");

   properties.put(Environment.POOL_SIZE, 1);
   properties.put(Environment.SHOW_SQL, "true");
   properties.put(Environment.HBM2DDL_AUTO, "create");

   configuration.setProperties(properties);
} catch (Exception e) {
   e.printStackTrace();
}
```

XML of code ??

- Persoonlijke voorkeur…
- Meestal XML

Hibernate – Stappenplan

1. Maven dependencies toevoegen
   - hibernate
   - database
2. Hibernate database configuratie schrijven
3. **Java Object mapping schrijven**
4. SessionFactory schrijven
5. Data Access Object (DAO) schrijven

Java Object Mapping

- Java Object koppelen aan een database (= Persistent Class)

- Twee mogelijkheden voor de configuratie
  - Configuratie in XML bestand : *classname*.hbm.xml
  - Configuratie in Java code dmv Java Annotiation

- Java Object aanpassen naar Persistent Class regels

Persistent Class regels

1. De class moet een default constructor hebben (en mag daarnaast nog extra constructors hebben).
2. De class moet een ID attribuut hebben.
3. De class attributen moeten private zijn.
4. De class attributen mogen niet final zijn.
5. De class attributen moeten een getter en setter hebben.

Persistent Class regels toepassen

```java
public class StormTrooper {
   private final String name;
   private final Rank rank;

   public StormTrooper(String name, Rank rank) {
      this.name = name;
      this.rank = rank;
   }

   public String getName() {
      return name;
   }

   public Rank getRank() {
      return rank;
   }
}
```

```java
public class StormTrooper {
   private int id;
   private String name;
   private Rank rank;

   public StormTrooper() {
   }

   public StormTrooper(String name, Rank rank) {
      this.name = name;
      this.rank = rank;
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }
```

Object XML mapping

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping SYSTEM
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="starwars.troopers.StormTrooper" table="STORMTROOPER">
      <id name="id“ column="id" type="int">
         <generator class="identity"/>
      </id>
      <property name="name" column="name" type="java.lang.String"/>
      <property name="rank" column="rank" type="org.hibernate.type.EnumType"/>
   </class>
</hibernate-mapping>
```

Object XML mapping

- tag class
  - attribute name : naam + package van je class
  - attribute table : tabelnaam
- tag id
  - attribute name : naam van het attribuut in je class
  - attribute column : kolomnaam
  - attribute type : data type van het attribuut
  - tag generator : ID generatie strategie
- tag property
  - attribute name : naam van het attribuut in je class
  - attribute column : kolomnaam
  - attribute type : data type van het attribuut

Object code mapping

```
@Entity
@Table(name = "STORMTROOPER")
public class StormTrooper implements Comparable<StormTrooper> {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id")
   private int id;
   @Column(name = "name")
   private String name;
   @Enumerated(EnumType.STRING)
   @Column(name = "rank")
   private Rank rank;

   public StormTrooper() {
   }

   public StormTrooper(String name, Rank rank) {
      this.name = name;
      this.rank = rank;
   }
```

Object code mapping

OPGELET : gebruik jakarta.persistence package voor imports !!!

- @Entity : marker annotation om aan te geven dat deze class op een tabel moet gemapped worden
- @Table : tabelnaam
- @Id : marker annotation om aan te geven dat het veld een ID is
- @GeneratedValue : ID generatie strategy
- @Column : kolomnaam
- @Enumerated : mapping van Enum naar kolom datatype

XML of code ??

- Persoonlijke voorkeur…
- Meestal code !!!

Mapping aan config doorgeven

- Het is niet voldoende om het Java Object te mappen, hibernate moet nog weten dat de mapping gebruikt mag worden.

- Twee mogelijkheden:
  - Toevoegen aan XML configuratie bestand
  - Toevoegen aan Java Code

Mapping : XML database configuration

- Optie 1 : Object mapping in XML bestand

```
<session-factory>
   <mapping resource="stormtrooper.hbm.xml"/>
</session-factory>
```

- Optie 2 : Object mapping in Java code

```
<session-factory>
   <mapping class="starwars.troopers.StormTrooper"/>
</session-factory>
```

Mapping : Code database configuration

- Optie 1 : Object mapping in XML bestand

```
Configuration configuration = new Configuration();
configuration.addResource(“stormtrooper.hbm.xml”);
```

- Optie 2 : Object mapping in Java code

```
Configuration configuration = new Configuration();
configuration.addAnnotatedClass(Stormtrooper.class);
```

XML of code ??

- Persoonlijke voorkeur…
- **Hybride** : XML + Code

Hybride configuration

```
Configuration configuration = new Configuration();
configuration.configure("hibernate.cfg.xml");
configuration.addAnnotatedClass(StormTrooper.class);
```

- Database configuratie in XML = STATISCH
- Object mapping in Java = DYNAMISCH
- Objecten toevoegen aan configuratie = DYNAMISCH

Hibernate – Stappenplan

1. Maven dependencies toevoegen
   - hibernate
   - database
2. Hibernate database configuratie schrijven
3. Java Object mapping schrijven
4. **SessionFactory schrijven**
5. Data Access Object (DAO) schrijven

SessionFactory

Communicatie met database = Session

Eénmalig moet je een SessionFactory aanmaken. Dit object is verantwoordelijk voor het toekennen van Session objecten.

HibernateUtil maakt SessionFactory

```
public class HibernateUtil {
   private static SessionFactory sessionFactory;

   public static SessionFactory getSessionFactory() {
      if (sessionFactory == null) {
         try {
            Configuration configuration = new Configuration();
            configuration.configure("hibernate.cfg.xml");
            configuration.addAnnotatedClass(StormTrooper.class);
            configuration.addAnnotatedClass(Squad.class);
            sessionFactory = configuration.buildSessionFactory();
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
      return sessionFactory;
   }
}
```

static SessionFactory

→ eenmalig aangemaakt en beschikbaar doorheen de toepassing

static getSessionFactory

- indien SessionFactory niet bestaat, aanmaken via config
- indien wel bestaat, static waarde teruggeven

SessionFactory gebruiken

```
Session session = null;
try {
   session = HibernateUtil.getSessionFactory().openSession();
   List<Squad> squads = session.createQuery("SELECT s from Squad s", Squad.class).list();
} finally {
   if(session != null) {
      session.close();
   }
}
```

1. openSession() via SessionFactory
2. Voer query uit op session
3. Sluit session in finally

SessionFactory– try with resources

```
try (Session session = HibernateUtil.getSessionFactory().openSession()) {
   List<Squad> squads = session.createQuery("SELECT s from Squad s", Squad.class).list();
}
```

Gebruik try with resources

- Zelfde als vorige slide !!
- finally block met session.close() wordt automatisch toegevoegd
- Voordelen:
  - Veel kortere code
  - Scoping: session is enkel gekend in de try block

Hibernate – Stappenplan

1. Maven dependencies toevoegen
   - hibernate
   - database
2. Hibernate database configuratie schrijven
3. Java Object mapping schrijven
4. SessionFactory schrijven
5. **Data Access Object (DAO) schrijven**

Data Access Object

Java Object dat de communicatie met de database voor één Entity op zich neemt.

Afschermen van database logica (queries)

Naamgeving class : Entity naam + DAO

DAO – SELECT list

```
public class SquadDAO {
   public List<Squad> getSquads() {
      try (Session session = HibernateUtil.getSessionFactory().openSession()) {
         String hql = “SELECT s FROM Squad s”;
         Query<Squad> query = session.createQuery(hql, Squad.class);
         return query.list();
      }
   }
}
```

List bevat 0 tot n elementen en is nooit null.

Gebruik HQL (Hibernate Query Language) om queries te schrijven.

DAO – find by id

```
public class SquadDAO {
   public Squad getSquadById(int id) {
      try (Session session = HibernateUtil.getSessionFactory().openSession()) {
         return session.get(Squad.class, id));
      }
   }
}
```

Object kan **null** zijn !!!

DAO – find by id with Optional

```
public class SquadDAO {
   public Optional<Squad> getSquadById(int id) {
      try (Session session = HibernateUtil.getSessionFactory().openSession()) {
         return Optional.ofNullable(session.get(Squad.class, id)));
      }
   }
}
```

Optional aanmaken

Optional.ofNullable → korte notatie voor :

if(squad == null) return Optional.empty() else return Optional.of(squad)

DAO – create

```
public class SquadDAO {
    public void createSquad(Squad squad) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.persist(squad);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
        }
    }
}
```

DAO – update

```
public class SquadDAO {
    public void updateSquad(Squad squad) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.merge(squad);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
        }
    }
}
```

DAO – delete

```
public class SquadDAO {
    public void updateSquad(Squad squad) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.remove(squad);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
        }
    }
}
```

HQL – Hibernate Query Language

HQL is net als SQL een query language.

Het verschil is:

- SQL werkt op tabellen en kolommen
- HQL werkt op objecten en attributen

Zelfde operatoren: SELECT / FROM / WHERE / ORDER BY / …

HQL SELECT - WHERE

```
SELECT s FROM StormTrooper s

SELECT s.name, s.rank FROM StormTrooper s

SELECT s FROM StormTrooper s WHERE s.rank = ‘SERGEANT’
```

- Gebruik de classname (StormTrooper) en niet de tabelnaam
- Gebruik steeds een alias (s)
- Gebruik attribuutnamen en niet de kolomnamen

Advanced hibernate mapping

Advanced Mapping ‘extends’

```
public class DarthVader extends StarWarsCharacter
```

En wat als een object overerft van een ander object?

Hoe wordt dit dan gemapt?

Hoe moet dit in de database komen?

Advanced Mapping ‘extends’ – optie 1

```
@MappedSuperclass
public abstract class StarWarsCharacter

@Entity
@Table(name=“DARTHVADER”)
public class DarthVader extends StarWarsCharacter
```

@MappedSuperclass:

De abstract class wordt zelf géén tabel, de subclasses bevatten alle velden van de abstract class + eigen velden.

Advanced Mapping ‘extends’ – optie 2

```
@Entity
@Table(name=“STARWARS_CHARACTER”)
@DiscriminatorColumn(name=“character_type”)
public abstract class StarWarsCharacter

@Entity
public class DarthVader extends StarWarsCharacter
```

Single Table:

Er wordt één tabel aangemaakt met alle attributen van zowel de abstract class als **alle subclasses**.

Er wordt daarnaast ook een extra kolom gemaakt : DiscriminatorColumn die de naam van de subclass bevat

Advanced Mapping : bidirectionele relatie

```
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = “squad_id”)
private Squad squad;
```

- @ManyToOne : marker om de relatie te activeren
  - FetchType : EAGER of LAZY
- @JoinColumn : naam van de Foreign Key kolom in de tabel

→ Zelfde notatie voor @OneToOne

Advanced Mapping : bidirectionele relatie

```
@OneToMany(fetch = FetchType.EAGER)
@JoinTable(name = “SQUAD_TROOPER”)
private Set<StormTrooper> troopers;
```

- @OneToMany : marker om de relatie te activeren
  - FetchType : EAGER of LAZY
- @JoinTable : naam van de koppeltabel

→ Zelfde notatie voor @ManyToMany


FOR TODAY
ANYWAY...