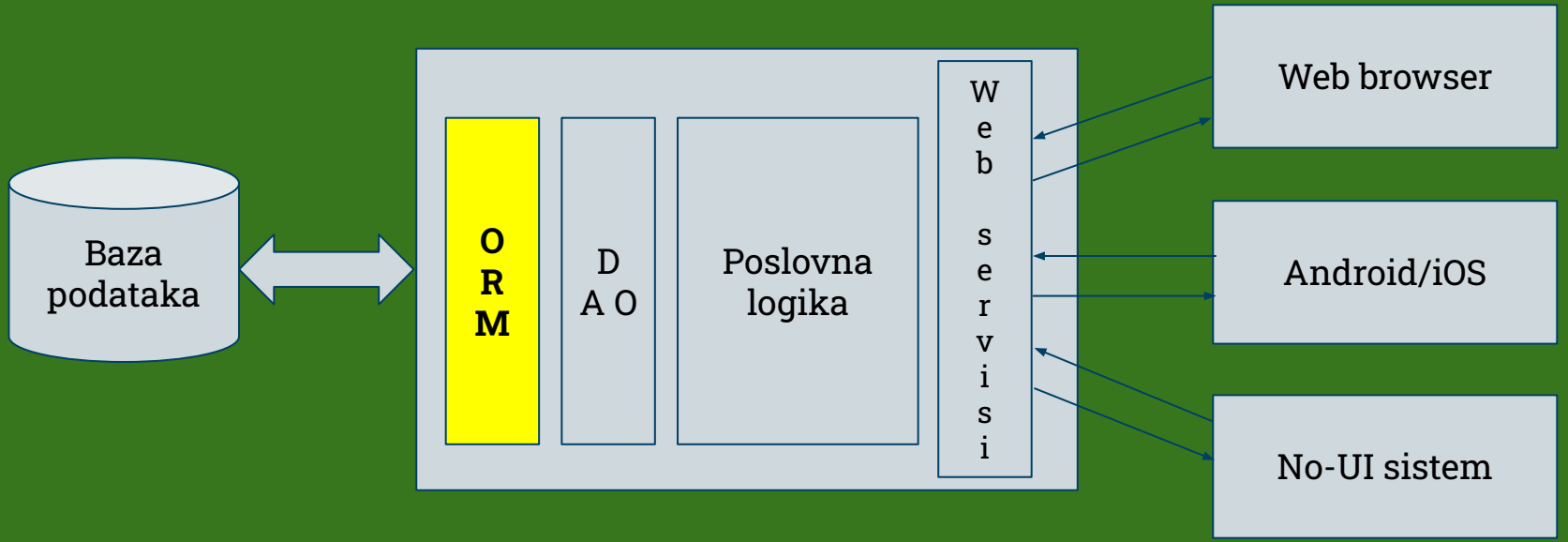
Java Persistence API

Sadržaj

- Objektno-relaciono mapiranje (ORM)
- Java Persistence API (JPA) *entity* klase
- JPA Query Language (JPQL)

Višeslojna arhitektura



Objektno-relaciono mapiranje

- Objektni model preslikati u relacioni model - *forward engineering*
- Relacioni model preslikati u objektni - *reverse engineering*
- Java Enterprise Edition (JEE) specifikacija za ORM je Java Persistence API (JPA)
- Više postojećih implementacija JPA specifikacije
  - Hibernate
  - EclipseLink
  - Apache OpenJPA

Java Persistence API

- Omogućava rad sa objektima
- Eliminiše potrebu za pisanjem SQL izraza i koda koji bi vršio preslikavanje iz jednog u drugi model
  - Čitanje podataka pomoću SQL upita i kreiranje objekata na osnovu pročitanih vrednosti
  - Ažuriranje podataka tako što se vrednosti osobina objekta koriste za izvršavanje SQL izraza
- JPA vrši preslikavanje korišćenjem metapodataka - Java anotacija
- JPQL - Java Persistence Query Language
  - statički upiti
  - dinamički upiti
- Omogućava kreiranje baze podataka na osnovu objektnog modela

JPA *entity* klase

- Plain Old Java Object (POJO) klase
- Anotacija `@Entity` ukazuje da se klasa preslikava u tabelu
- Hibernate podrazumevano koristi strategiju mapiranja naziva tako što *camelcase* transformiše u *snakecase* znak
- Svaki objekat klase predstavlja slog u odgovarajućoj tabeli
- Mora imati konstruktor bez argumenata
- Mora biti definisano obeležje koje predstavlja identifikator

Strategije mapiranja naziva

- Explicitna (`@Table` i `@Column`) ili implicitna strategija mapiranja naziva
- Hibernate ima dva koraka u strategiji imenovanja
  - logičko imenovanje (`spring.jpa.hibernate.naming.implicit-strategy`)
    - Camel Case u Snake Case
  - fizičko imenovanje (`spring.jpa.hibernate.naming.physical-strategy`)
    - Podrazumevano je isto kao logičko
- Moguće je definisati sopstvenu strategiju
  - dodati prefiks, sufiks i slično

Obeležja *entity* klase

- Perzistentna obeležja se preslikavaju u kolone
  - `@Column` - preslikavanje u kolonu sa odgovarajućim nazivom
    - strategija mapiranja naziva kao kod klasa i tabela
  - `@Id` - identifikaciono obeležje koje se preslikava u kolonu primarnog ključa
  - Ukoliko je u bazi podataka kompozitni ključ, tip identifikacionog obeležja će biti klasa čija se obeležja preslikavaju u odgovarajuće kolone primarnog ključa
- Tranzienta obeležja su izvedene vrednosti
  - `@Transient` - obeležje se ne preslikava u kolonu ali postoji u objektnom modelu
  - Primer: `firstName` i `lastName` su posebna obeležja ali je često potrebno prikazati vrednost `firstName + " " + lastName`

Generisanje vrednosti identifikatora

- Anotacija `@GeneratedValue` ukazuje da će vrednost identifikatora biti automatski generisana
- Četiri strategije generisanja
  - `GenerationType.IDENTITY`
  - `GenerationType.SEQUENCE`
  - `GenerationType.TABLE`
  - `GenerationType.AUTO`
- Primer:

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

Strategije generisanja vrednosti identifikatora

- `@GenerationType.IDENTITY`
  - najjednostavnija strategija
  - koristi autoinkrementiranu kolonu u bazi podataka
- `@GenerationType.SEQUENCE`
  - koristi sekvencu iz baze podataka
- `@GenerationType.TABLE`
  - koristi se tabela koja simulira sekvencu - prevaziđeno, retko se koristi
- `@GenerationType.AUTO`
  - JPA implementacija određuje način generisanja vrednosti identifikatora
  - Hibernate generiše id zavisno od dijalekta baze podataka

Primer JPA klase

```
@Entity
public class TStore {

  @Id
  @GeneratedValue
  (stategy=GenerationType.IDENTITY)
  private Integer id;

  @Column(nullable=false)
  private String name;

  public TStore() {
  }

  public Integer getId() {
    return this.id;
  }

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

  public String getName() {
    return this.name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Mapiranje veza

- `@OneToOne`, `@OneToMany`, `@ManyToOne` i `@ManyToMany`

```
@Entity
public class TOrder {
  …
  @OneToMany(mappedBy=”tOrder”)
  private List<TOrderItem> tOrderItems;

  @ManyToOne
  @JoinColumn(name=”customer_id”)
  private TCustomer tCustomer;
  …
}
```

Govori koja **promenljiva** u klasi `TOrderItem` referencira `TOrder`

Govori koja **kolona** u tabeli `T_ORDER` referencira primarni ključ u `T_CUSTOMER`

Načini učitavanja povezanih objekata

- Kada se definišu veze između *entity* klasa, moguće je upravljati učitavanjem zavisnih objekata
- Postoje dva tipa učitavanja (engl. *fecth type*):
  - LAZY (u prevodu “lenj”)
    - ne učitava se zavisni objekat dok se ne pozove get metoda
    - *default* za kolekcije i mape
  - EAGER (u prevodu “željan”)
    - odmah se učitava zavisni objekat
    - *default* za sve tipove osim kolekcija i mapa

Primer učitavanja povezanih objekata

```
@Entity
public class TOrder {
  …
  @OneToMany(mappedBy=”tOrder”, fetch=FetchType.EAGER)
  private List<TOrderItem> tOrderItems;

  @ManyToOne
  @JoinColumn(name=”customer_id”, fetch=FetchType.LAZY)
  private TCustomer tCustomer;
  …
}
```

Prilikom učitavanja porudžbine učitaće se i sve njene stavke

Prilikom učitavanja porudžbine neće se učitati podaci o kupcu dok se ne pozove metoda `getCustomer()`

Serijalizacija i `@JsonIgnore`

- Serijalizacija predstavlja transformaciju stanja objekta u trajni zapis
- Suprotan proces se zove deserijalizacija
- Kod postojanja dvosmerne reference potrebno je sprečiti beskonačno ugnežđavanje objekata

```
public class TCustomer {
  …
  @JsonIgnore
  public List<TOrder> getTOrders() {
    return this.tOrders;
  }
}
```

Klasa `EntityManager`

- Omogućava izvršavanje operacija nad bazom podataka
  - upiti
  - DML naredbe
  - upravljanje transakcijama
- U JEE aplikacija `EntityManager` objekat se automatski kreira
- U JSE aplikacijama mora se ručno kreirati i pozvati
- Sve *entity* klase čijim se objektima upravlja pripadaju jednom ***kontekstu perzistencije***
- Kontekst perzistencije se opisuje fajlom `persistence.xml` koji je smešten u podfolder `META-INF` unutar foldera izvornog koda

Primer fajla `persistence.xml`

```
<persistence xmlns=...>
  <persistence-unit name="jpa-example" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql:..." />
      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.password" value="mypassw" />
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="validate" />
    </properties>
  </persistence-unit>
</persistence>
```

JPQL

- JPA upitni jezik čija je sintaksa slična SQL
- Nastao na osnovu Hibernate Query Language-a (HQL) koji je nestandardni Hibernate upitni jezik
- Upiti se izvršavaju nad JPA objektima a ne direktno nad bazom podataka
- Podržava statičke i dinamičke (parametarske) upite
- Primer:

```
SELECT c
FROM TCustomer c
ORDER BY c.firstName, c.lastName
```

Izvršenje JPQL upita

```
HashMap<String, String> map = new HashMap<String, String>();
map.put("javax.persistence.jdbc.url", url);
map.put("javax.persistence.jdbc.user", usr);
map.put("javax.persistence.jdbc.password", pass);
EntityManagerFactory emf =
                Persistence.createEntityManagerFactory(“jpa-example“, map);
EntityManager em = emf.createEntityManager();
Query query = em.createQuery(“select e
                              from employee e”);
List<Employee> emps = query.getResultList();
for(Employee emp : emps)
  System.out.println("Employee name:" + emp.getName());
```

Upravljanje perzistencijom entiteta (1)

```
EntityManagerFactory emf = ...
EntityManager em = ...
em.getTransaction().begin();
Employee employee = new Employee();
employee.setId(1201);
employee.setFirstName("Marko");
employee.setLastName("Markovski");
employee.setSalary(40000);
em.persist(employee);
em.getTransaction().commit();
em.close();
emf.close();
```

Upravljanje perzistencijom entiteta (2)

- Modifikacija

```
em.getTransaction().begin();
Employee employee = em.find(Employee.class, 1201);
employee.setSalary(46000);
em.getTransaction().commit();
```

- Brisanje

```
em.getTransaction().begin();
Employee employee = em.find(Employee.class, 1201);
em.remove(employee);
em.getTransaction().commit();
```

Imenovani JPQL upiti

- U okviru entity klase moguće je predefinisati imenovane JPQL upite

```
@Entity
@NamedQueries({
  @NamedQuery(
    name = “Player.findAllUnordered”,
    query = “SELECT p FROM Player p”),
  @NamedQuery(
    name = “Player.findByAge”,
    query = “SELECT p FROM Player p WHERE p.age = :age”)
})
public class Player ...
```

Izvršenje imenovanog JPQL upita

- Statički upit:

```
EntityManager em = …
List players =
        em.createNamedQuery(“Player.findAllUnordered”)
            .getResultList();
```

- Parametarski upit:

```
EntityManager em = …
List players =
        em.createNamedQuery(“Player.findByAge”)
            .setParameter(“age”, 27)
            .getResultList();
```

Rezime

- JPA implementacija omogućava ORM u Java aplikacijama
- Java anotacije se koriste kao metapodaci za preslikavanje
- Postoje različite strategije generisanja identifikatora objekata
- Referencirani objekti mogu da se učitavaju na različite načine
- JPQL omogućava definisanje upita
- Klasa `EntityManager` omogućava izvršavanje upita, CRUD operacija i upravaljanje transakcijama