Razvoj Softvera

dr.sc. Emir Mešković

IX predavanje

Pregled

- Komunikacija s bazom podataka
- Objektno-relaciono mapiranje (ORM)
- Java Persistence API

Uvod

- Pohrana i dohvat podataka kao osnovni zahtjev većine aplikacija
- Najčešći oblik pohrane – relacijske baze podataka
- Potreba za povezivanjem aplikacije sa DBMS-om putem standardnog paketa koji omogućava
  - Funkcije za čuvanje i pribavljanje objekata za upravljanje transakcijama i oporavkom
  - Standardne funkcije baze podataka
- Java Database Conectivity

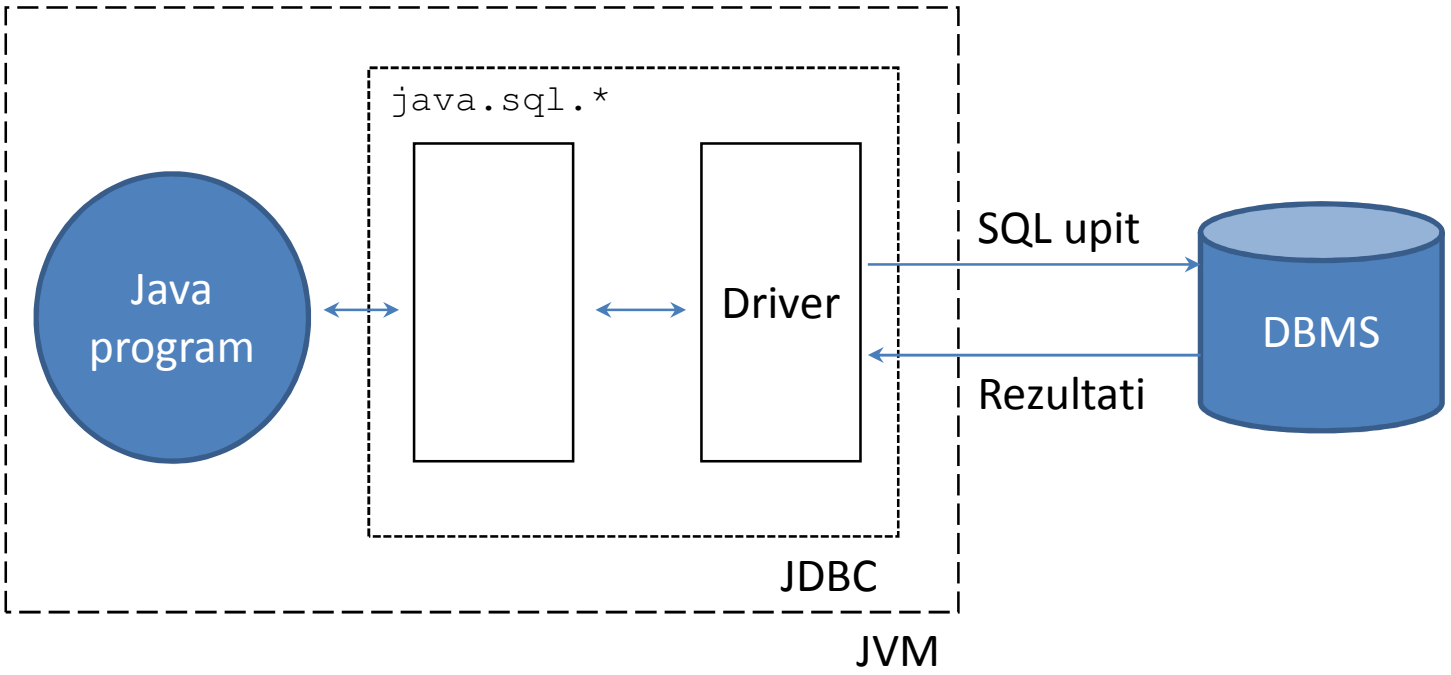
JDBC API

- Java specifična verzija Object Dabase Conectivity (ODBC) specifikacije za pristup relacijskim bazama podataka iz bilo kojeg jezika ili platforme
- Java programima omogućava punu interakciju sa bazama podataka
  - Uspostavljanje konekcije prema bazi podataka
  - Izvršavanje složenih SQL naredbi
  - Dohvat podataka iz baze podataka
  - Mijenjanje i brisanje podataka u bazi podataka
- Značajno se oslanja na SQL – programski interfejsi su portabilni ali SQL nije

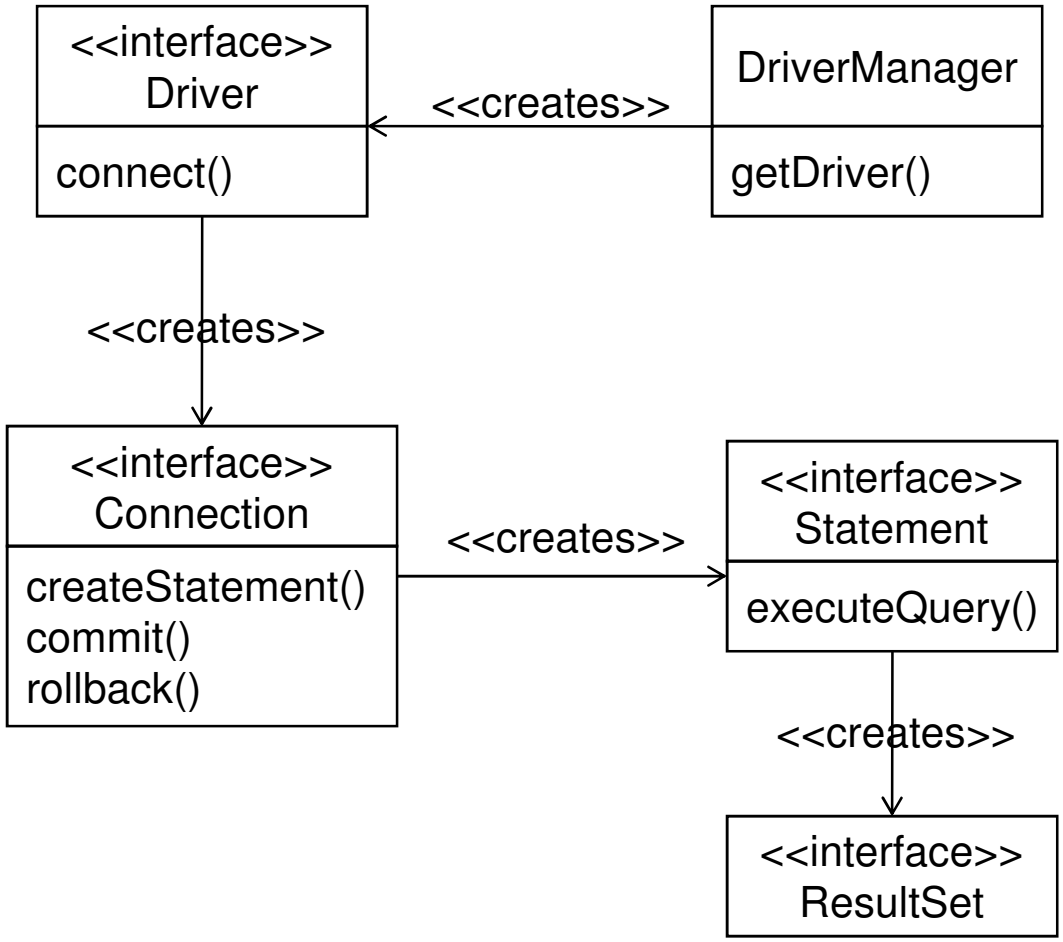
JDBC API

- Omogućava da se na jednostavan način radi sa različitim DBMS – bez obzira na kombinaciju platforme i DBMS
- JDBC drajveri ne dolaze uz JDBC API
  - Proizvođači DBMS kreiraju odgovarajuće drajvere
  - Svi drajveri se koriste na isti način
  - Instalacija – uključivanje jar arhiva u kojima se nalaze
- Pogodan za klijent-server aplikacije
  - Klijent – poslovna logika i upravljanje konekcijama i transakcijama
  - Server - DBMS

JDBC API


java.sql.*
Java program
Driver
SQL upit
Rezultati
DBMS
JDBC
JVM

JDBC API


<<interface>>
Driver
connect()
<<creates>>
DriverManager
getDriver()
<<creates>>
<<interface>>
Connection
createStatement()
commit()
rollback()
<<creates>>
<<interface>>
Statement
executeQuery()
<<creates>>
<<interface>>
ResultSet

JDBC API

- Konekcija na DBMS je predstavljena objektom tipa `Connection`

```
Connection con = DriverManager.getConnection("url",
"user", "password")
```

- U slučaju greške javlja se `SQLException`
- Za upotrebu SQL naredbi i rad sa relacijskim bazama podataka koristi se interfejs `Statement`
- Rezultati se vraćaju u obliku objekata klasa koje implementiraju `ResultSet` interfejs
  - Metode: `execute`, `executeUpdate`, `executeQuery`, `close`
  - Ako je SELECT naredba – reultat je `ResultSet` objekat

JDBC API


```
public static void main(String[] args) {
   ...
   try{
     //KORAK 1: Registracija JDBC drajvera
     Class.forName("com.mysql.jdbc.Driver");
     //KORAK 2: Otvaranje konekcije
     Connection conn = DriverManager.getConnection(DB_URL,USER,PASS);
     //KORAK 3: Izvršavanje upita
     String sql1 = "SELECT id, name, salary FROM Employees“
     String sql2 = "SELECT salary FROM Employees WHERE id = ? AND
     name = ?"
     Statement stmt = conn.createStatement();
     ResultSet rs1 = stmt.executeQuery(sql1);
     PreparedStatement pstmt = conn.prepareStatement(sql2);
     pstmt.setString(1, ID);
     pstmt.setString(2, NAME);
     ResultSet rs2 = pstmt.executeQuery();
     ...
   }...
```

JDBC API

- Rezultujući skup podataka u tabelarnoj formi
- Mogu se dobiti podaci o nazivima kolona, tipu podataka i vrijednostima
- `ResultSet` sadrži kursor koji pokazuje na poziciju tekućeg reda podataka u rezultujućem skupu
- Metoda `next` pomjera kursor na sljedeću poziciju

```
while(rs.next())
{
  int empId= rs.getInt("id");
  String empName= rs.getString("name");
  float empSalary = rs.getFloat("salary");
  ...
}
```

Objektno-relaciono mapiranje

- Relacijske baze (RDBMS)
  - Obezbjeđuju pohranjivanje podataka koji se organiziraju u redovima tabela koje su povezane putem ključeva (primarni/strani)
- Softver projekti
  - Dizajn i implementacija u objektno orijentiranim jezicima
  - Manipulacija podacima zapisanim u objektima koji egzistiraju u RAM-u
  - Organiziraju se oko servisa koje pružaju RDBMS
- Objektno-relaciono mapiranje
  - Konverzija između nekompatibilnih domena
  - Pruža programeru transparentan pristup RDBMS-u direktno iz OO domena
  - Dizajn uzorci koji tretiraju ovu materiju:
    - Active Record
    - Data Mapper

Java Persistance API

- Java standard koji između ostalog definira
  - Način na koji se Java objekti zapisuju u relacione baze
  - API za CRUD (Create, Read, Update, Delete) opreacije persistentnih Java objekata
  - Jezik za pretraživanje (JPQL - Java Persistance Query Language)
- Osobine
  - POJO (plain old java object) persistentnost
  - ORM se obavlja u potpunosti putem metapodataka
  - Kod bez promjena radi sa različitim RDBMS-ovima
  - "*Convention over configuration*" princip dizajna JPA minimizira popratnu konfiguraciju uz programski kod i čini je potrebnom samo za slučaj kada default-ne postavke nisu adekvatne

Entity

- Entity (entitet)
  - Jedinica koja ima stanje i odnose
  - Glavne osobine:
    - Persistability
      - Stanje i odnosi se mogu pohraniti (obično u bazu) radi kasnijeg pristupa
    - Identity
      - Ostvaruje se putem posebnog unikatnog identifikatora (ključ u bazi)
    - Transactionality
      - Kreiranja, promjene i brisanja se događaju unutar transakcija
      - Transakcije su atomske operacije

Entiteti i JPA

- Sa JPA stanovišta
  - Entitet je POJO čije se stanje dijelimično ili u potpunosti može mapirati u bazu
  - Mapiranje se obavlja u skladu sa metapodacima koji se zapisuju:
    - u kodu klase od koje je napravljen objekat i to putem anotacija iz `javax.persistence` paketa ili
    - unutar posebnog XML fajla

Obična klasa


```
package model;

public class Employee {

  private int id;
  private String name;
  private long salary;

  public Employee() {}
  public Employee(int id) { this.id = id; }

  public int getId() { return id; }
  public void setId(int id) { this.id = id; }
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public long getSalary() { return salary; }
  public void setSalary(long salary) { this.salary = salary; }

  @Override
  public String toString() {
    return getId()+" "+getName()+" "+getSalary();
  }
}
```

Entitet


```
package model;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Employee {
  @Id
  private int id;
  private String name;
  private long salary;

  public Employee() {}
  public Employee(int id) { this.id = id; }

  public int getId() { return id; }
  public void setId(int id) { this.id = id; }
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public long getSalary() { return salary; }
  public void setSalary(long salary) { this.salary = salary; }

  @Override
  public String toString() {
    return getId()+" "+getName()+" "+getSalary();
  }
}
```

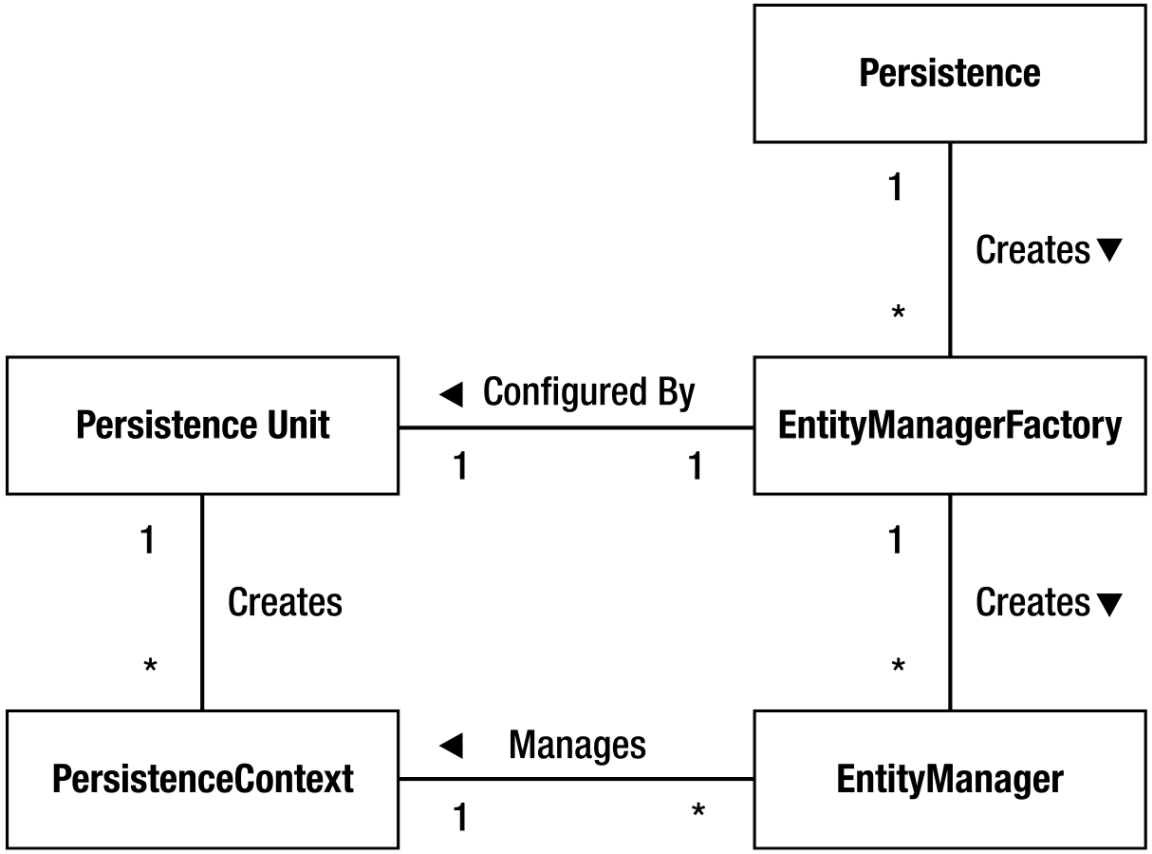
Persistance Context

- Cilj je odrađivati operacije u RDBMS-u što rijeđe i to putem odrađivanja operacija nad entitetima unutar posebnog konteksta koji:
  - čuva reference na entitete koji su u memoriji a čije se stanje mijenjalo ili koji se tek trebaju unijeti u bazu
  - obavlja ekvivalentne RDBMS operacije nad entitetima u memoriji a u RDBMS-u samo kada je to neophodno
- Kompletno menadžiran i programeru dostupan putem instance klase `EntityManager` koja prije svega vodi računa o svim objektima unutar konteksta i pohranjuje ih u bazu kada je to potrebno.

EntityManager

- Konfigurira se:
  - da može da vodi računa samo o entitetima određenog tipa
  - da ostvaruje vezu sa tačno određenom bazom podataka unutar određenog RDBMS-a
  - putem *persistance unit*-a specificiranog unutar `persistance.xml` fajla snimljenog u `META-INF` folderu projekta
- Kreira se putem Factory Design pattern-a implementiranog putem klase `EntityManagerFactory`

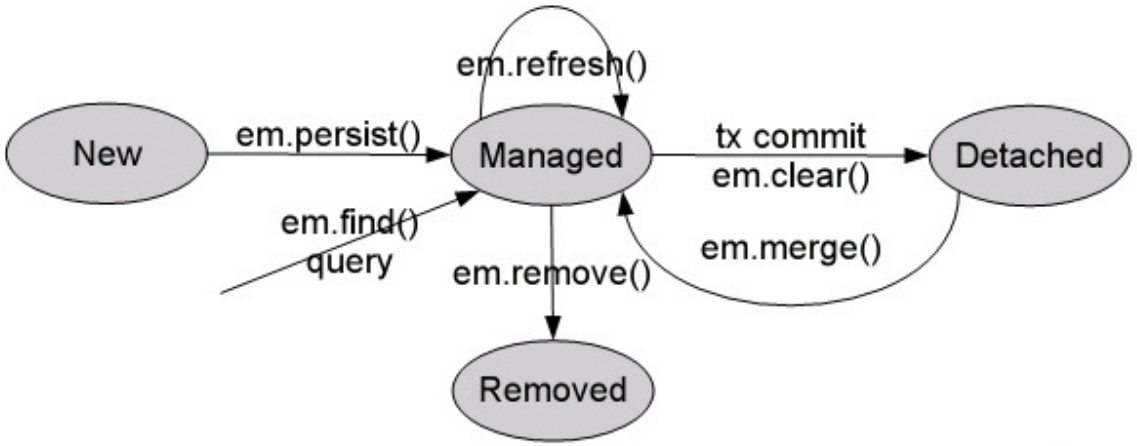
JPA koncepti i njihove relacije




```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("NekiDefiniraniPU");
EntityManager em = emf.createEntityManager();
```

Dijagram stanja entiteta spram PC-a

- `EntityManager` metodi za kontrolu stanja entiteta:
  - `persist`() - dodaje entitet u bazu
  - `remove`() - uklanja entiet iz baze
  - `merge`() - sinhronizira stanje entiteta i vraća mendžiranu kopiju
  - `refresh`() - ponovno učitava stanje iz baze
- `persist`, `remove` i `merge` odgađaju se dok `EntityManager` ne odluči da obavi operacije tj
  - do `commit` operacije neke transakcije ili
  - do neke query operacije (izuzimajući `find`)



Persistence Unit primjer definicije


```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">

  <persistence-unit name="TestPU" transaction-type="RESOURCE_LOCAL">
    <class>model.Employee</class>
    <properties>
      <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.EmbeddedDriver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:derby:./empDB;create=true" />
      <property name="javax.persistence.jdbc.user" value="test" />
      <property name="javax.persistence.jdbc.password" value="test" />

      <property name="eclipselink.ddl-generation" value="create-tables" />
    </properties>
  </persistence-unit>

</persistence>
```

Primjer upotrebe JPA koncepata


```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import model.Employee;
import java.util.List;

public class Main {
  private static final String PERSISTENCE_UNIT_NAME = "TestPU";
  private static EntityManagerFactory emf;

  public static void main(String[] args) {
    emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
    EntityManager em = emf.createEntityManager();

    em.getTransaction().begin();
    Employee udarnik = new Employee(2);
    udarnik.setName("Anoniman");
    udarnik.setSalary(500);
    em.persist(udarnik);
    em.getTransaction().commit();

    Query q = em.createQuery("select e from Employee e",Employee.class);
    List<Employee> empList = q.getResultList();

    for (Employee radnik : empList) {
      System.out.println(radnik);
    }

    System.out.println("Ukupan broj: " + empList.size());
    em.close();
    emf.close();
  }
}
```