# Programska potpora komunikacijskim sustavima

Dvanaesto predavanje, 7. lipnja 2023.

Izv. prof. dr. sc. Krešimir Pripužić

Doc. dr. sc. Josip Vuković

## Objektno relacijsko mapiranje (ORM)


Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

# Sadržaj predavanja

- Uvod
- Objektno-relacijsko mapiranje - ORM
- Java Persistance API - JPA
- Hibernate
- Primjeri korištenja
- Sažetak

# Uvod

- Aplikacije koriste podatke
  - Potrebno brzo i efikasno spremanje i korištenje podataka
- Perzistencija podataka
- Što odabrati?
  - Tekstualne datoteke
  - SQL
  - NoSQL
  - Serijalizirani objekti

# Objektno-relacijsko mapiranje (ORM)

- Povezuje dva (logički) „nekompatibilna” sustava
- Stvara se sloj između relacijskih baza podataka (Oracle, MySQL, H2, MongoDB, PostgreSQL, … ) i objektno orijentiranih programskih jezika (Java, C#, Python, …)
- Omogućuje programeru „automatiziran” prijenos podataka iz objekata u bazu i obrnuto

# JDBC (*Java Database Connectivity*)

- Javin API za spajanje na bazu podataka
- Svaka baza ima svoj API
- Problem u različitoj sintaksi SQL-a ovisno o bazi podataka
- Programer mora sam voditi računa o osnovnim CRUD (*create, read, update, delete*) operacijama nad bazom podataka

# Dodavanje potrebne ovisnosti u pom.xml

```
<!--
https://mvnrepository.com/artifact/com.h2database/h2
-->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.210</version>
</dependency>
```

# Primjer korištenja JDBC-a [1]

```
public class JdbcMain {
    static final String DB_URL = "jdbc:h2:~/students";
    //static final String DB_URL = "jdbc:mysql://localhost/students";
    static final String USER = "sa";
    static final String PASS = "passwd";

    public static void main(String[] args) {
        try ( Connection connection = DriverManager.getConnection(DB_URL, USER, PASS);  Statement statement =
connection.createStatement()) {
            //create a table
            String sql = "CREATE TABLE Student " + "(id INT PRIMARY KEY, first_name VARCHAR(255), last_name
VARCHAR(255));";
            //String sql = "CREATE TABLE Student " + "(id INTEGER not NULL, " + " first_name VARCHAR(255), " + "
last_name VARCHAR(255)" + " PRIMARY KEY ( id ))";
            statement.executeUpdate(sql);

            //insert a student
            sql = "INSERT INTO Student VALUES (1, 'Ante', 'Antić')";
            //sql = "INSERT INTO(id, first_name, last_name) Student VALUES (1, 'Ante', 'Antić')";
            statement.executeUpdate(sql);

            //print all students
            printAllStudents(statement);
```

# Primjer korištenja JDBC-a [1]

```
            //delete a student
            sql = "DELETE FROM Student WHERE ID=1";
            statement.executeUpdate(sql);

            //print all students
            printAllStudents(statement);

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void printAllStudents(final Statement statement) throws SQLException {
        String sql = "SELECT * FROM Student";
        ResultSet resultSet = statement.executeQuery(sql);
        while (resultSet.next()) {
            for (int i = 1; i <= resultSet.getMetaData().getColumnCount(); i++) {
                System.out.print(resultSet.getString(i) + " ");
            }
            System.out.println("");
        }
    }
}
```

# JPA (Java Persistence API)

- Specifikacija koja opisuje objektno-relacijsko mapiranje
- Postoji više implementacija koje podržavaju JPA
  - Hibernate
  - EclipseLink
  - Apache Open JPA
  - TopLink
  - …
- Programer se bavi samo objektima, a ne mora SQL-om za rad s bazom podataka

# JPA (Java Persistence API)

- Mapiranje podataka između baze i objektno orijentiranog jezika koristeći informacije sadržane u *metapodatcima*
  - *Metapodatci* mogu biti opisani u konfiguracijskom XML-u ili koristeći anotacije u samom programskom kodu ili kombinacijom anotacija i XML-a
  - XML konfiguracija uvijek nadjačava anotacije
- Koristi se jezik sličan SQL-u za slanje „direktnih” upita u bazu (iz programskog koda) prilikom npr. testiranja

# Hibernate

- Nije potrebno nasljeđivanje apstraktnih klasa/implementiranje sučelja
- POJO objekti „ne znaju” da će biti spremljeni u bazu podataka
- Relacije među klasama koje su podržane su:
  - *one-to-one*
  - *one-to-many*
  - *many-to-many*
- Podržana je refleksivnost (*one-to-many* relacija s objektima istog tipa kao i objekt nad kojim se provodi relacija)
- Za vrijeme *runtime-a* stvaraju se SQL upiti na temelju anotacija/XML-a

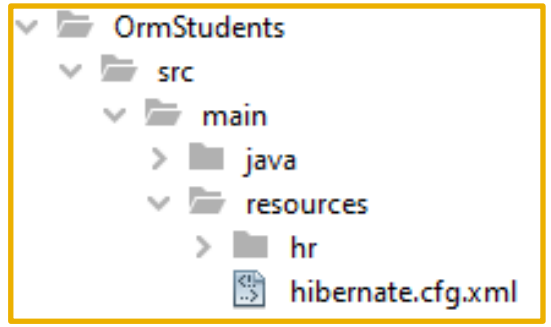
# Dodavanje dodatne ovisnosti u pom.xml

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.0.2.Final</version>
    <type>pom</type>
</dependency>
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.210</version>
</dependency>
```

# Klasa HibernateUtil

```
public class HibernateUtil {
    private static SessionFactory factory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            if (factory == null) {
                StandardServiceRegistry registry = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
                Metadata metadata = new MetadataSources(registry).getMetadataBuilder().build();
                factory = metadata.getSessionFactoryBuilder().build();
            }
            return factory;
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return factory;
    }

    public static void shutdown() {
        getSessionFactory().close();
    }
}
```

# Konfiguracijska datoteka hibernate.cfg.xml




```
<?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.connection.driver_class">org.h2.Driver</property>
        <property name="hibernate.connection.url">jdbc:h2:~/students</property>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password">passwd</property>
        <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
        <property name="hbm2ddl.auto">create</property>
        <mapping resource="hr/fer/zkist/ppks/students/student.hbm.xml"/>
<!--         <mapping class="hr.fer.zkist.ppks.students.Student"/>-->
    </session-factory>
</hibernate-configuration>
```

# Klasa Student [1]

```
public class Student {
    private String firstName;
    private String lastName;
    private String jmbag;

    public Student(String firstName, String lastName, String jmbag) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.jmbag = jmbag;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
```

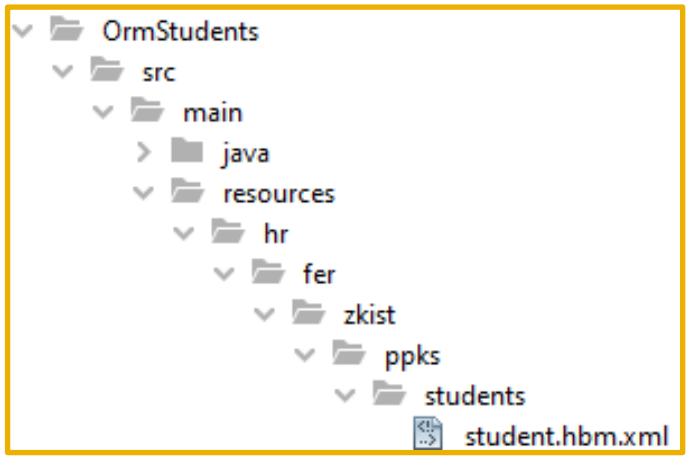
# Klasa Student [2]

```
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getJmbag() {
        return jmbag;
    }

    public void setJmbag(String jmbag) {
        this.jmbag = jmbag;
    }

    @Override
    public String toString() {
        return "Student{" + "firstName=" + firstName + ", lastName=" + lastName + ", jmbag=" + jmbag + '}';
    }
}
```

# Datoteka Student.hbm.xml

```
<hibernate-mapping>
    <class name="hr.fer.zkist.ppks.students.Student" table="Student">
        <id name="jmbag" column="id">
            <generator class="assigned" />
        </id>
        <property name="firstName" column="first_name" />
        <property name="lastName" column="last_name" />
    </class>
</hibernate-mapping>
```

| Student | | |
|---|---|---|
| id | first_name | last_name |
| | | |



# Primjer korištenja – klasa `OrmMain`

```
Session session = HibernateUtil.getSessionFactory().openSession();

//add a student
session.beginTransaction();
Student student = new Student("Ante", "Antić", "1");
session.persist(student);
session.getTransaction().commit();

//print all students
List<Student> list = session.createQuery("FROM Student", Student.class).list();
System.out.println(list);

//delete a student
System.out.println("deleting");
session.beginTransaction();
session.remove(session.get(Student.class, "1"));
session.getTransaction().commit();

//print all students
list = session.createQuery("FROM Student", Student.class).list();
System.out.println(list);

HibernateUtil.shutdown();
```

# Klasa `Student` s anotacijama (umjesto `Student.hbm.xml`)

```
@Entity
@Table(name = "Student")
public class Student {

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Id
    @Column(name = "Id", unique = true, nullable = false)
    private String jmbag;

    ...
}
```

# UML dijagram klasa primjera *one-to-many*

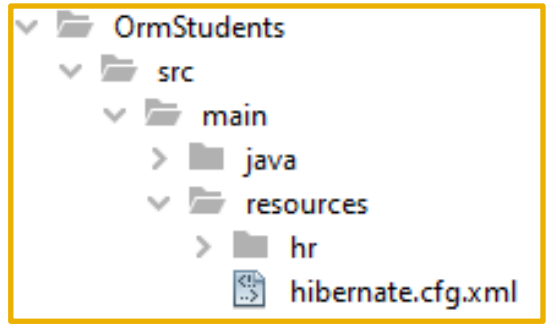
# Klasa `Student`

```java
@Entity
@Table(name = "Student")
public class Student {
    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Id
    @Column(name = "id", unique = true, nullable = false)
    private String jmbag;

    @OneToMany(cascade=CascadeType.PERSIST)
    @JoinColumn(name="student_id")
    private List<Email> emails;

    public Student(String firstName, String lastName, String jmbag, List<Email> emails) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.jmbag = jmbag;
        this.emails = emails;
    }

    ...
}
```

# Klasa Email

```java
@Entity
@Table(name = "Email")
public class Email {

    @Id
    @Column(name = "email")
    private String email;

    public Email(String email) {
        this.email = email;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Email{" + "email=" + email + '}';
    }
}
```

# Konfiguracijska datoteka hibernate.cfg.xml




```
<?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.connection.driver_class">org.h2.Driver</property>
        <property name="hibernate.connection.url">jdbc:h2:~/students</property>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password">passwd</property>
        <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
        <property name="hbm2ddl.auto">create</property>
        <!--<mapping class="hr.fer.zkist.ppks.students.Student"/>-->
        <mapping class="hr.fer.zkist.ppks.orm.one_to_n.Student"/>
        <mapping class="hr.fer.zkist.ppks.orm.one_to_n.Email"/>
    </session-factory>
</hibernate-configuration>
```

# Primjer korištenja – klasa `OrmEmailsMain`

```
public class OrmEmailsMain {

    public static void main(String[] args) {
      Session session = HibernateUtil.getSessionFactory().openSession();

      //add a student
      session.beginTransaction();

      Email e1 = new Email("ivo.ivic@gmail.com");
      Email e2 = new Email("ivo.ivic@yahoo.com");

      List<Email> emails = new ArrayList<>();
      emails.add(e1);
      emails.add(e2);

      Student student = new Student("Ante", "Antić", "1", emails);
      session.persist(student);
      session.getTransaction().commit();

      ...
    }
}
```

# Sadržaj baze podataka

| Email | |
|---|---|
| email | student_id |
| ivo.ivic@gmail.com | 1 |
| ivo.ivic@yahoo.com | 1 |

| Student | | |
|---|---|---|
| id | first_name | last_name |
| 1 | Ivo | Ivic |

# Sažetak

- Danas nema aplikacije bez podataka
- Aplikacijski kod bi trebao biti „neovisan” o korištenim platformama
- JPA + Hibernate je koristan alat koji olakšava programerima pohranu i korištenje podataka iz SQL baza podataka bez pisanja SQL naredbi u kodu
- Automatizirana manipulacija podatcima u „pozadini” tijekom izvođenja programa
- Jednostavna zamjena baza podataka kroz konfiguracijsku datoteku bez ponovnog prevođenja samog programskog koda


# Spring Boot & Hibernate

SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

# Primjer RESTful web-usluge

| resurs | podržane metode | šalje | svrha |
|---|---|---|---|
| /persons | GET | | vraća listu osoba |
| | POST | osoba | stvara novu osobu |
| /persons/{id} | GET | | vraća osobu s ID-om |
| | DELETE | | briše osobu |
| | PUT | osoba | mijenja podatke o osobi |

# Pohrana resursa u REST usluzi

- Originalna usluga je pohranjivala osobe u memoriji u listi
- Poboljšana aplikacija će koristiti Hibernate za pohranu osoba u bazi podataka H2

# Klasa Person s anotacijama

```
@Entity
@Table(name = "Person")
@JsonIgnoreProperties({"hibernateLazyInitializer","handler"})
public class Person {

    @Id
    @Column(name = "id", unique = true, nullable = false)
    private final long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "phone")
    private String phone;

    @Column(name = "room")
    private String room;

    public Person() {
        this.id = 0;
    }

    ...
}
```

# Sučelje PersonRepository

```
@Repository
public interface PersonRepository extends JpaRepository<Person, Long> {
}
```

# Klasa PersonService [1]

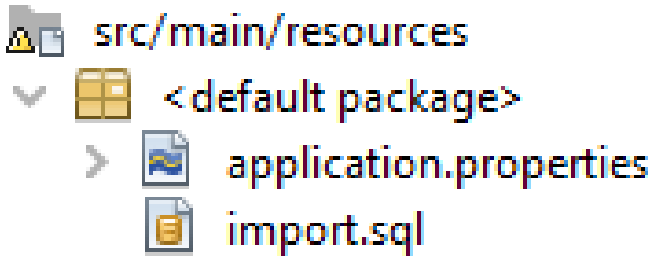
```
@Service
public class PersonService implements PersonInterface {

    @Autowired
    private PersonRepository personRepository;

    @Override
    public List<Person> get() {
        List<Person> persons = personRepository.findAll();
        Collections.sort(persons, Comparator.comparing(Person::getId));
        return persons;
    }

    @Override
    public void insert(Person person) {
        personRepository.save(person);
    }
```

# Klasa `PersonService` [2]

```
    @Override
    public Person get(long id) {
        return personRepository.getById(id);
    }

    @Override
    public boolean delete(long id) {
        boolean found = personRepository.existsById(id);
        personRepository.deleteById(id);
        return found;
    }

    @Override
    public boolean update(Person person, long id) {
        boolean found = personRepository.existsById(id);
        personRepository.save(person);
        return found;
    }
}
```

# Dodavanje osoba u bazu podataka H2 pri pokretanju korištenjem datoteke `import.sql`

```
INSERT INTO Person values(0, 'Goran', 'Delač', '01/6129-549', 'D-339-2');
INSERT INTO Person values(1, 'Marin', 'Šilić', '01/6129-549', 'D-339-2');
INSERT INTO Person values(2, 'Marin', 'Vuković', '01/6129-658', 'C07-04');
INSERT INTO Person values(3, 'Krešimir', 'Pripužić', '01/6129-745', 'C08-17');
```



# Dodavanje potrebnih ovisnosti u pom.xml

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	 <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
	<groupId>com.h2database</groupId>
	<artifactId>h2</artifactId>
</dependency>
```