Basic CRUD

ORM, Hibernate and CRUD

SoftUni Team

Technical Trainers

Software University

SoftUni Foundation

Software University

http://softuni.bg

Table of Contents

Have a Question?

SoftUni Foundation

sli.do

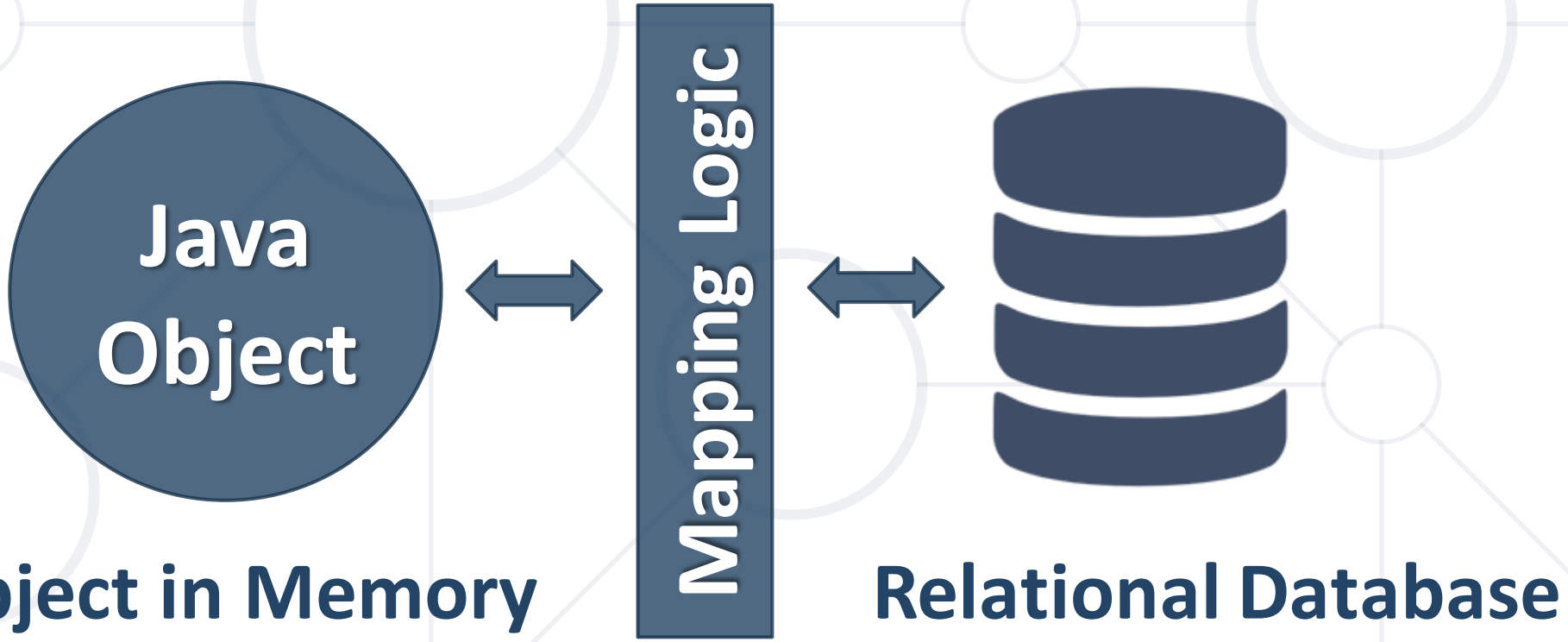
#tech-java

Object-Relational Mapping

ORM Concepts and Features

ORM Overview

- **ORM Frameworks** maps OOP **classes** to database **tables**



ORM Frameworks - Features

- Java **classes** are mapped to DB **tables**
  - DB **relationships** are mapped to class **associations**
- ORM provides API for **CRUD** operations
  - **List** objects / query database
  - **Create** new object
  - **Update** existing object
  - **Delete** existing object

CRUD operations execute **SQL commands** in the DB

- ORM provides **schema synchronization** (DB migrations)

JPA

Java Persistence API

JPA Overview

- Database persistence technology for Java (**official standard**)
- Object-relational mapping (**ORM**) technology
- Operates with **POJO** entities with **annotations** or **XML** mappings
- Implemented by many ORM engines: **Hibernate**, **EclipseLink**, …

Entities in JPA

- A JPA entity is just a **POJO class**
  - Abstract or concrete **top level** Java class
  - Non-final fields/properties, no-arguments constructor
  - Direct field or property-based access
- Getter/setter **can contain logic** (e.g. validation)

Hibernate Framework

Mapping Java Classes to Database Tables

Hibernate Framework

- Hibernate is a **Java ORM framework**
- Using **Java Annotations**
- Implements **JPA Standard**
- Mapping an object-oriented model to a relational database
- Maintain the database schema

Hibernate Configuration

SoftUni Foundation

pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
// Continues on the next slide
```

Hibernate Configuration (2)

SoftUni Foundation

pom.xml

```
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Hibernate Configuration (3)

SoftUni Foundation

application.properties

```
spring.datasource.driverClassName = com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/spring_demo?
                                useSSL=false&createDatabaseIfNotExist=true
spring.datasource.username = root
spring.datasource.password =
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect
                                                    .MySQL57InnoDBDialect
spring.jpa.properties.hibernate.format_sql=TRUE
spring.jpa.hibernate.ddl-auto=update
```

Annotations

- **@Entity** - Declares the class as an entity or a table
- **@Table** - Declares table name
- **@Id** - Specifies the property, use for identity of the class
  - **@GeneratedValue** - specifies how the identity attribute can be initialized
- **@Transient** - Specifies the property that is not persistent
- **@Column** - Specifies the column attribute for the persistence property

Hibernate Implementation

SoftUni Foundation

- Entity = = Java Class + Annotation

```
@Entity
@Table(name = "students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;
  @Column
  private String name;
}
```

```
import javax.persistence.Entity;
import javax.persistence.Table;
```

JPA Repository

SoftUni Foundation

- Spring Data JPA provides a built-In repository

```
public interface StudentRepository extends
                    JpaRepository<Student, Integer> { … }
```

- Provides bunch of generic CRUD methods

```
studentReposity.findAll();       //Finds all records
studentReposity.findById();      //Finds record by id
studentReposity.saveAndFlush(); //Inserts an entity
studentReposity.delete();        //Removes an entity
```

Saving an Entity

SoftUni Foundation

- First, we need a html form

```
<form th:action="@{/create}" th:method="POST">
    <div>
        <label>Name: </label>
        <input type="text" name="name"/>
    </div>
    <input type="submit">
</form>
```

- Render the view

```
@GetMapping("/create")
public ModelAndView create(ModelAndView modelAndView) {
  modelAndView.setViewName("create");
  return modelAndView; }
```

Saving an Entity (2)

SoftUni Foundation

- Mappin the data to an object

```
@PostMapping("/create")
public String createProcess(Student student) {
    this.studentRepository.saveAndFlush(student);
    return "redirect:/";
}
```

- As response, redirect to "/"
  - "/" is equivalent to home page

Find an Entity

SoftUni Foundation

- You can find easily an entity by id

```
@GetMapping("/details/{id}")
public ModelAndView details(@PathVariable(value = "id") Integer id,
                            ModelAndView modelAndView) {
    Student student = this.studentRepository.findById(id).get();
    modelAndView.setViewName("details");
    modelAndView.addObject("student", student);
    return modelAndView;
}
```

Implement the view

Editing an Entity

SoftUni Foundation

- First, we need to find the entity by id and pass it to the view

```
@GetMapping("/edit/{id}")
public ModelAndView edit(@PathVariable(value = "id") Integer id,
                         ModelAndView modelAndView) {
    Student student = this.studentRepository.findById(id).get();
    modelAndView.setViewName("edit");
    modelAndView.addObject("student", student);
    return modelAndView;
}
```

Editing an Entity (2)

SoftUni Foundation

- To update the entity, process the post request

```
@PostMapping("/edit/{id}")
public String edit(Student student) {
    this.studentRepository.saveAndFlush(student);
    return "redirect:/";
}
```

- **saveAndFlush** updates the entity
- As a response, redirect to **"/"** – the **index page**

Deleting an Entity

SoftUni Foundation

- Very similar to edit operation
- First, find an entity by id and pass It to the view

```
@GetMapping("/delete/{id}")
public ModelAndView delete(@PathVariable(value = "id") Integer id,
                           ModelAndView modelAndView) {
    Student student = this.studentRepository.findById(id).get();
    modelAndView.setViewName("delete");
    modelAndView.addObject("student", student);
    return modelAndView;
}
```

Deleting an Entity (2)

SoftUni Foundation

- To delete an entity, process the post request

```
@PostMapping("/delete/{id}")
public String delete(@PathVariable(value = "id") Integer id) {
    this.studentRepository.deleteById(id);
    this.studentRepository.flush();
    return "redirect:/";
}
```

- **deleteById** – deletes an entity by id
- **flush** – changes will be saved **immediately**

Summary

SoftUni Foundation

- ORM is used to **map objects** to **database tables**
- Java Persistence API is an **official standard** for Java ORM's
- Hibernate is a widely used **ORM**
  - Implements **JPA**
- Implementing **CRUD** operations

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SoftwareGroup
doing it right

NETPEAK

SmartIT

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

SoftUni Organizational Partners

SoftUni Foundation

Trainings @ Software University (SoftUni)

SoftUni Foundation

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license