Auto Mapping Objects DTO

Auto Mapping – DTOs and Domain Objects, Model Mapper

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.org/

Table of Contents

Questions

Software University

sli.do

#

Data Transfer Objects

Transmitting Aggregated Data from Entities

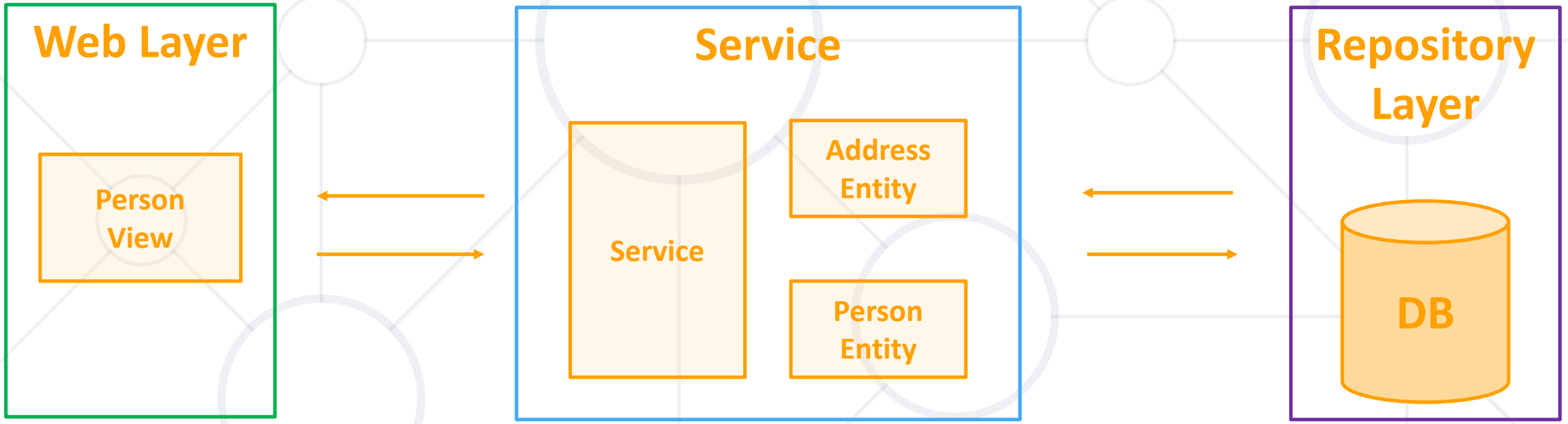
Data Transfer Object Concept

- Domain objects are mapped to view models – **DTO**s
  - A DTO is a **container class**
  - Exposes only properties, **not methods**
- In **simple** applications, domain objects can be used in the meaning of DTOs
  - Otherwise, we accomplish nothing but **object replication**

Entity Usage

Information is passed in the form of DTO

Information is passed by domain objects(entities)



Information is aggregated and entities are mapped to corresponding DTOs

DTO Usage

Software University

Employee.java

```
@Entity
@Table(name = "employees")
public class Employee {
    //…
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "salary")
    private BigDecimal salary;
    @ManyToOne
    @JoinColumn(name = "address_id")
    private Address address;
    //…}
```

Address.java

```
@Entity
@Table(name = "addresses")
public class Address {
    //…
    @Column
    private String city;
    //…
}
```

EmployeeDTO.java

```
public class EmployeeDto {

    private String firstName;
    private BigDecimal salary;
    private String addressCity;
}
```

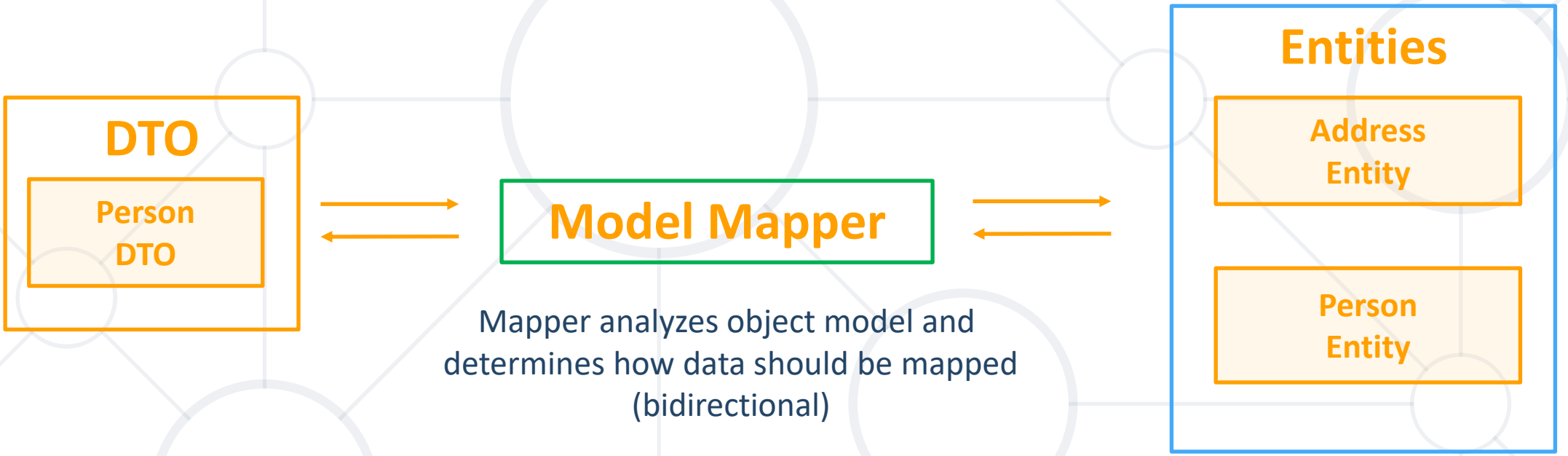
Model Mapping

Converting Entity Objects to DTOs

Model Mapping

- We often want to map data between **objects with similar structure**
- Model mapping is an easy way to **convert one model to another**
- Separate **models** must **remain segregated**
- We can **map entity objects to DTOs** using ModelMapper
- Uses **conventions** to determine how properties and values are mapped to each other

Model Mapper



Mapper analyzes object model and determines how data should be mapped (bidirectional)

Adding Model Mapper

- Add as maven dependency:

pom.xml

```
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>2.4.2</version>
</dependency>
```

- Create object:

ConsoleRunner.java

```
ModelMapper modelMapper = new ModelMapper();
EmployeeDto employeeDto = modelMapper.map(employee, EmployeeDto.class);
```

Source of information

Destination object(DTO)

Simple Mapping Entity to DTO

Software University

EmployeeDto.java

```
public class EmployeeDto {

    private String firstName;
    private BigDecimal salary;
    private String addressCity;
}
```

Address.java

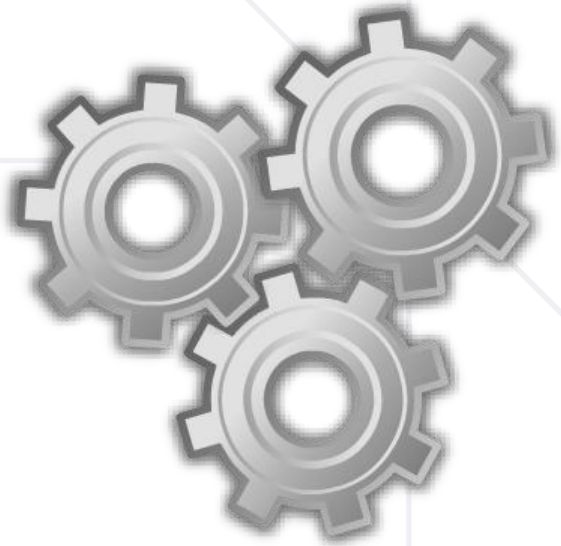
```
@Entity
@Table(name = "addresses")
public class Address {
    //…
    @Column
    private String city;
    //…
}
```

Employee.java

```
@Entity
@Table(name = "employees")
public class Employee {
    //…
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "salary")
    private BigDecimal salary;
    @ManyToOne
    @JoinColumn(name = “address_id")
    private Adress address;
    //…}
```

Model Mapping

- ModelMapper uses **conventions** to map objects
  - Sometimes fields differ and mapping **won't be done properly**
  - In this case some manual mapping is needed



Explicit Mapping DTO to Entity (1)

Software University

EmployeeDto.java

```
public class EmployeeDto {

    private String firstName;

    private BigDecimal salary;

    private String addressCity;
}
```

Employee.java

```
@Entity
@Table(name = "employees")
public class Employee {
    //…
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "salary")
    private BigDecimal salary;
    @ManyToOne
    @JoinColumn(name = “address_id")
    private Adress address;
    //…}
```

Address.java

```
@Entity
@Table(name = "addresses")
public class Address {
    //…
    @Basic
    private City city;
    //…
}
```

City.java

```
@Entity
@Table(name = "cities")
public class Address {
    //…
    @Basic
    private String name;
    //…
}
```

Explicit Mapping DTO to Entity (2)

ConsoleRunner.java

```
ModelMapper modelMapper = new ModelMapper();
PropertyMap<EmployeeDto, Employee> employeeMap = new PropertyMap<EmployeeDto, Employee>()
{
        @Override
        protected void configure() {
            map().setFirstName(source.getName());
            // Add mappings for other fields
            map().setAddressCity(source.getAddress().getCity().getName());
        }
};

modelMapper.addMappings(employeeMap).map(employeeDto,employee);
```

Explicit Mapping DTO to Entity – Java 8

Software University

ConsoleRunner.java

```
ModelMapper modelMapper = new ModelMapper();
TypeMap<EmployeeDto, Employee> typeMap = mapper.createTypeMap(
EmployeeDto.class, Employee.class);
typeMap.addMappings(m -> m.map(src -> src.getName(),
Employee::setFirtsName));
typeMap.map(employeeDto);
```

Validation

ConsoleRunner.java

```
ModelMapper modelMapper = new ModelMapper();
modelMapper.createTypeMap(EmployeeDto.class, Employee.class);
modelMapper.validate();
```

Source

Destination

Exception

```
1) Unmapped destination properties found in TypeMap[EmployeeDto -> Employee]:

        com.persons.domain.entities.Employee.setAddress()
        com.persons.domain.entities.Employee.setId()
        com.persons.domain.entities.Employee.setBirthday()
```

Skipping Properties

Software University

ConsoleRunner.java

```
ModelMapper modelMapper = new ModelMapper();
PropertyMap<EmployeeDto, Employee> employeeMap = new PropertyMap<EmployeeDto, Employee>()
{
            @Override
            protected void configure() {
                skip().setSalary(null);
            }
        };

modelMapper.addMappings(employeeMap).map(employeeDto,employee);
```

Skip Salary

ConsoleRunner.java – Java 8

```
typeMap.addMappings(mapper -> mapper.skip(Employee::setSalary));
typeMap.map(employeeDto);
```

Converting Properties – Java 7

Terminal.java

```
ModelMapper modelMapper = new ModelMapper();
Converter<String, String> stringConverter = new AbstractConverter<String, String>() {
            @Override
            protected String convert(String s) {
                return s == null ? null : s.toUpperCase();
            }
        };

PropertyMap<EmployeeDto, Employee> employeeMap = new PropertyMap<EmployeeDto, Employee>()
{
            @Override
            protected void configure() {
                using(stringConverter).map().setFirstName(source.getName());
            }
        };

modelMapper.addMappings(employeeMap).map(employeeDto,employee);
```

Convert Strings to Upper Case

Use Convertion

Converting Properties – Java 8

ConsoleRunner.java

```
ModelMapper modelMapper = new ModelMapper();
Converter<String, String> toUppercase = ctx -> ctx.getSource() == null ? null :         c
tx.getSource().toUpperCase();
TypeMap<EmployeeDto, Employee> typeMap = mapper.createTypeMap(EmployeeDto.class, Employee
.class).addMappings(mapper -> mapper.using(toUppercase).map(EmployeeDto::getName, Employe
e::setFirstName));
typeMap.map(employeeDto);
```

Summary

- We should **not expose full data** about our entities
  - Present only those which should be visible to the outside world
- Mapping is **done with ModelMapper**
  - Allows us to **map all or single fields**
  - Allows us to **convert field values**

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

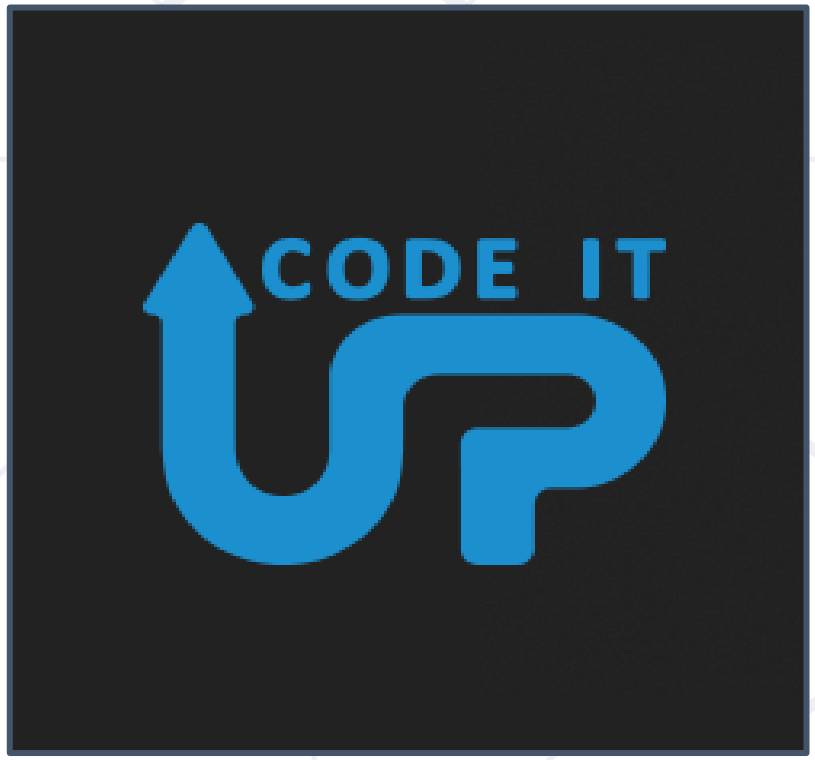
SoftUni Kids

SoftUni Diamond Partners

Software University

Educational Partners

Software University



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- SoftUni Global
  - softuni.org

Software University

SoftUni Global

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg
- © SoftUni Global – https://softuni.org