Auto Mapping Objects DTO

Auto Mapping – DTOs and domain objects, ModelMapper

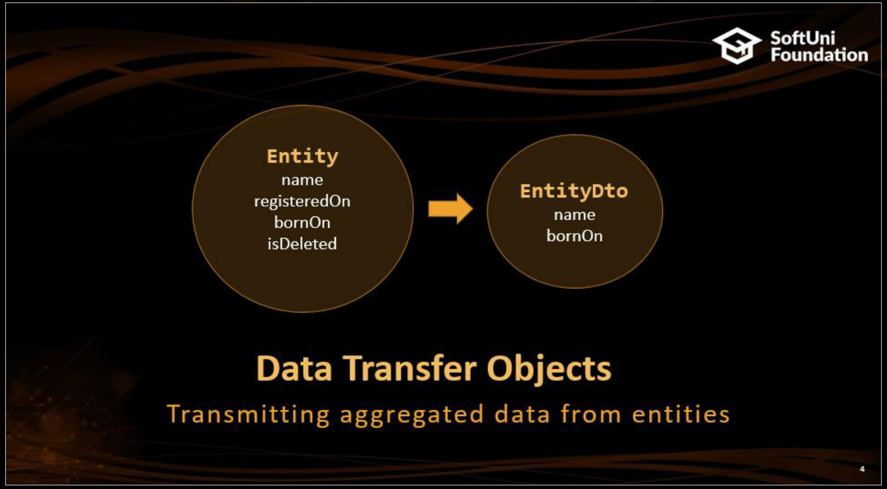
SoftUni Team

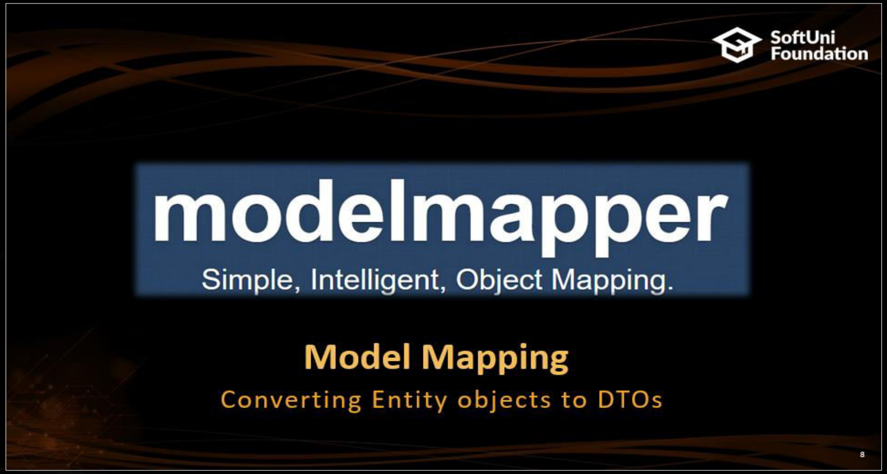
Technical Trainers

Software University

http://softuni.bg

Table of Contents



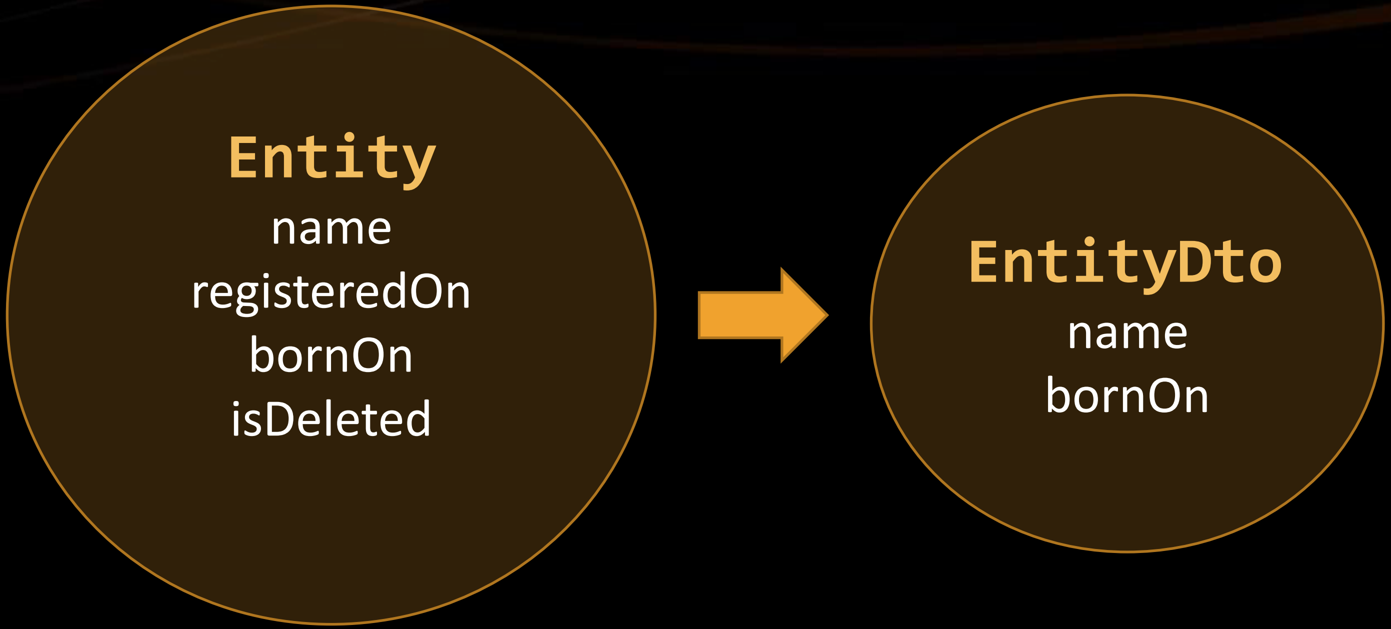


Questions

SoftUni Foundation

sli.do

#JavaDB



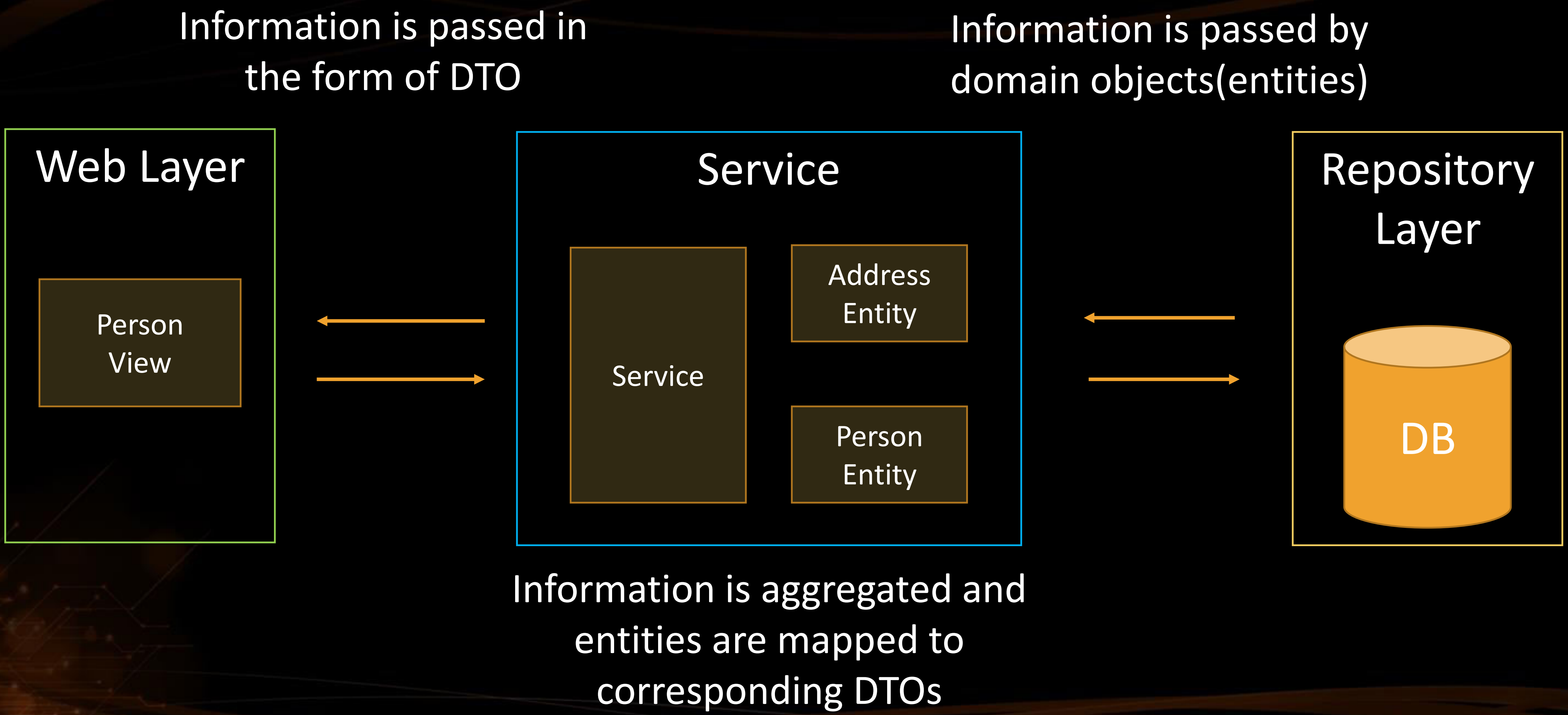
Data Transfer Objects

Transmitting aggregated data from entities

Data Transfer Object Concept

- In complex applications we do not want to expose unnecessary data in the display layer
- Domain objects are mapped to view models – DTOs
  - A DTO is nothing more than 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)
Web Layer
Person View
Service
Service
Address Entity
Person Entity
Repository Layer
DB
Information is aggregated and entities are mapped to corresponding DTOs

DTO Usage

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 {
    //…
    @Basic
    private String city;
    //…
}
```

EmployeeDTO.java

```
public class EmployeeDto {

    private String firstName;

    private BigDecimal salary;

    private String addressCity;
}
```

SoftUni Foundation

modelmapper

Simple, Intelligent, Object Mapping.

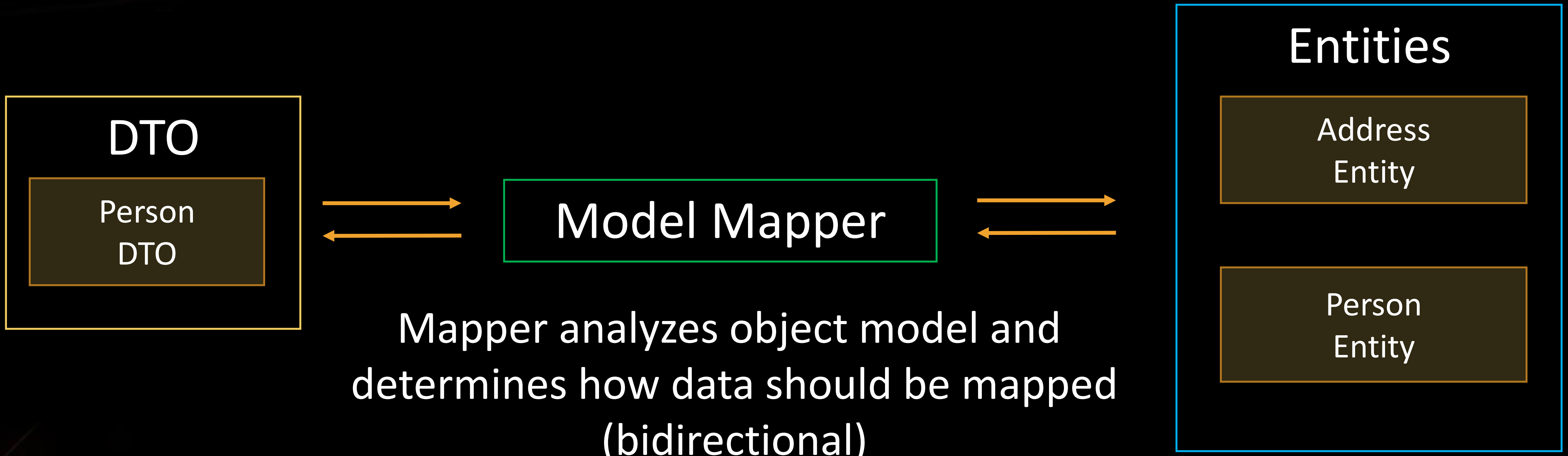
Model Mapping

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 entities 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 ModelMapper

- Add as maven dependency:

pom.xml

```
<dependency>
            <groupId>org.modelmapper</groupId>
            <artifactId>modelmapper</artifactId>
            <version>1.1.0</version>
</dependency>
```

- Create object:

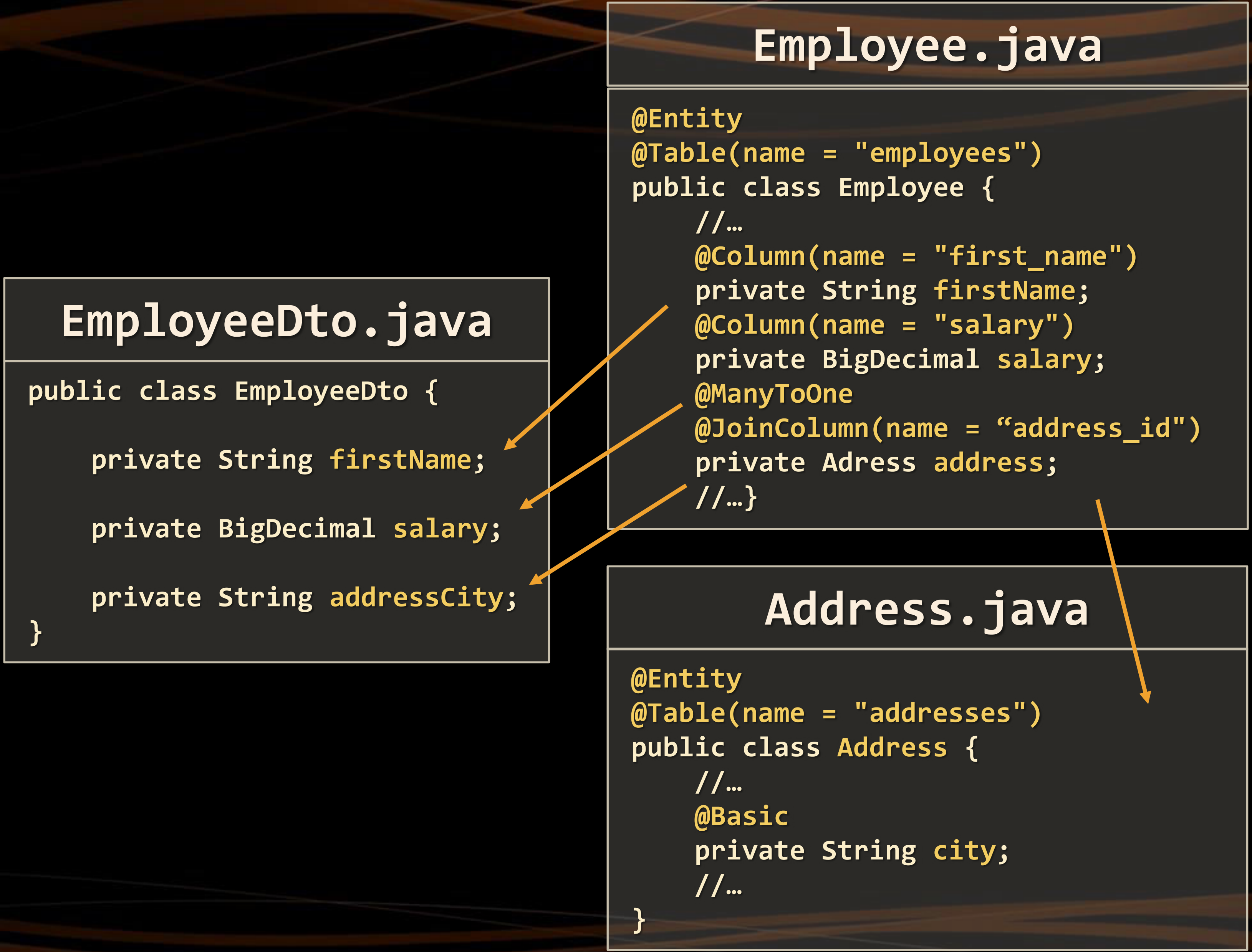
ConsoleRunner.java

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

Source of information

Destination object(DTO)

Simple Mapping Entity to DTO



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;
    //…}
```

EmployeeDto.java

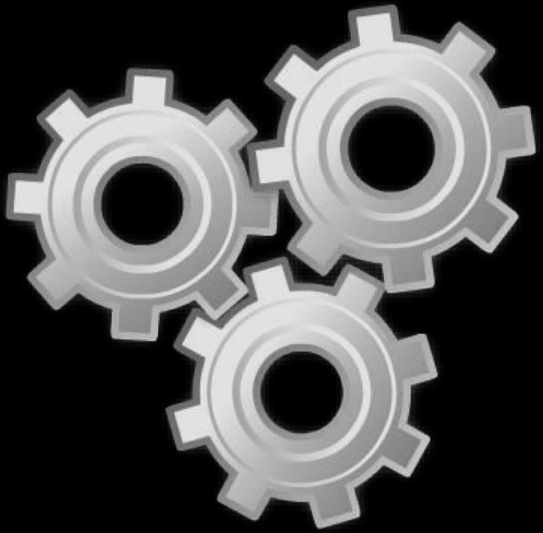
```
public class EmployeeDto {

    private String firstName;

    private BigDecimal salary;

    private String addressCity;
}
```

Address.java

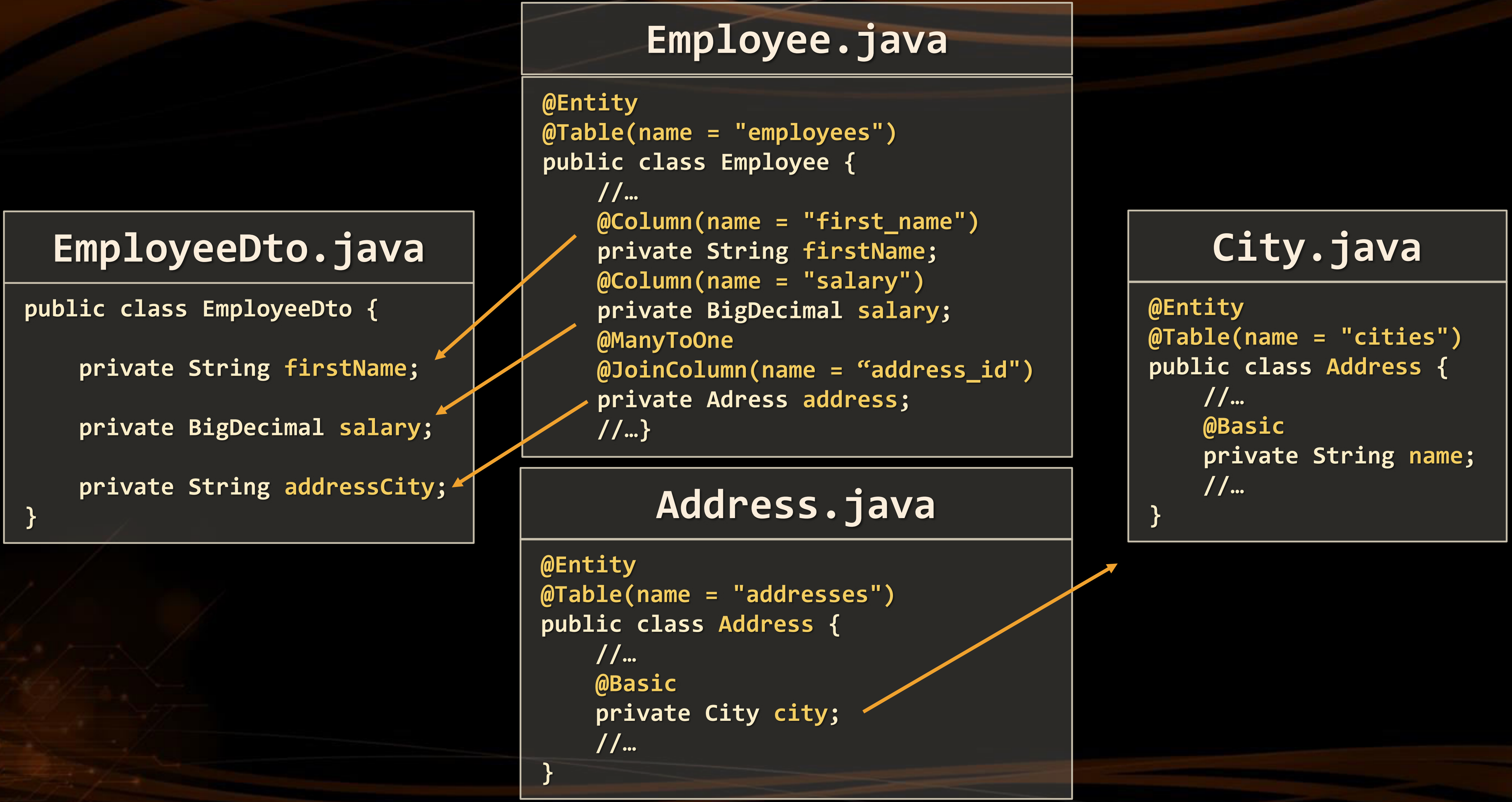
```
@Entity
@Table(name = "addresses")
public class Address {
    //…
    @Basic
    private String city;
    //…
}
```

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



EmployeeDto.java

```java
public class EmployeeDto {

    private String firstName;

    private BigDecimal salary;

    private String addressCity;
}
```

Employee.java

```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

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

City.java

```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

ConsoleRunner.java (ModelMappper v1.1.0)

```
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

```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

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 :
          ctx.getSource().toUppercase();
TypeMap<EmployeeDto, Employee> typeMap = mapper.createTypeMap(EmployeeDto.class,
Employee.class).addMappings(mapper -> mapper.using(toUppercase).map(EmployeeDto::getName,
Employee::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 easily done with ModelMapper
  - Allows us to map all or single fields
  - Allows us to convert field values

Auto Mapping Objects DTO

Questions?

https://softuni.bg/courses/databases-advanced-hibernate

License

Free Trainings @ Software University

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