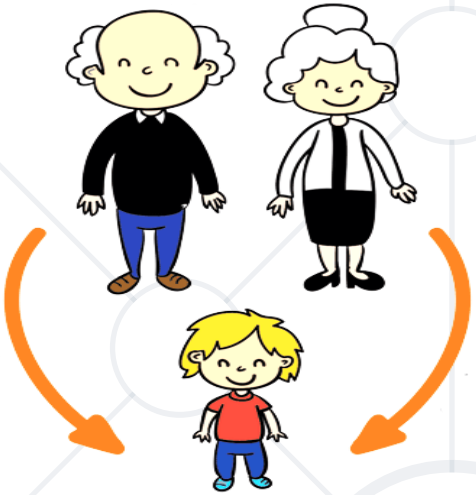
Hibernate (JPA) Code First Entity Relations

Advanced Mapping



SoftUni Team

Technical Trainers

Software University

Software University

http://softuni.bg

Table of Content

Questions

SoftUni Foundation

sli.do

#JavaDb

Java Persistence API Inheritance

Fundamental Inheritance Concepts

Inheritance

SoftUni Foundation

- Inheritance is a fundamental concept in most programming languages
  - SQL does not support this kind of relationships
- Implemented by any JPA framework by **inheriting** and **mapping Entities**

JPA Inheritance Strategies

SoftUni Foundation

- Implemented by the `javax.persistence.Inheritance` annotation
- The following mapping strategies are used to map the entity data to the underlying database:
  - A single **table per class** hierarchy
  - A table per **concrete entity class**
  - "**Join**" strategy – mapping common fields in a single table

Table Per Class Strategy

SoftUni Foundation

- **Table creation for each entity**
  - A table defined for each concrete class in the inheritance
  - Allows inheritance to be used in the object model, when it does not exist in the data model
- Querying root or branch classes can be very difficult and **inefficient**

Table Per Class Strategy: Example

SoftUni Foundation

Vehicle.java

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Vehicle {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    private int id;
    @Basic
    private String model;
    protected Vehicle() {}
    protected Vehicle(String model) {
        this.model = model;
    }
}
```

Inheritance type

A table generator is used for each table

Table Per Class Strategy: Example (2)

SoftUni Foundation

Bike.java

```
@Entity
@Table(name = "bikes")
public class Bike extends Vehicle {
    private final static String model = "BIKE";
    public Bike(){
        super(model);
    }
}
```

Table Name

Car.java

```
@Entity
@Table(name = "cars")
public class Car extends Vehicle {
    private final static String model = "CAR";
    public Car(){
        super(model);
    }
}
```

Table Name

Table Per Class Strategy: Example (3)

SoftUni Foundation

Main.java

```
..
Vehicle bike = new Bike();
Vehicle car = new Car();

em.persist(bike);
em.persist(car);
```

- Result:

bikes	
id	type
1	"BIKE"

cars	
id	type
1	"CAR"

Table Per Class Strategy: Conclusion

- **Disadvantages**:
  - Repeating information in each table
  - Changes in super class involves changes in all subclass tables
  - No foreign keys involved (unrelated tables)
- **Advantages**:
  - No NULL values – no unneeded fields
  - Simple style to implement inheritance mapping

Table Per Class: Joined

- Table is defined for each class in the inheritance hierarchy
  - Storing of that class **only the local attributes**
  - Each table must store object's **primary key**

Table Per Class Strategy: Example

SoftUni Foundation

Vehicle.java

```
@Entity
@Table(name = "vehicles")
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Vehicle {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    private int id;

    @Basic
    private String model;

    protected Vehicle() {}
    protected Vehicle(String model) {
        this.model = model;
    }
}
```

Inheritance type

A table generator is used for each table

Table Per Class Strategy: Example (2)

SoftUni Foundation

TransportationVehicle.java

```
@MappedSuperclass
public abstract class TransportationVehicle extends Vehicle {
    private int loadCapacity;

    // Getters and setters
}
```

Table Per Class Strategy: Example (2)

SoftUni Foundation

PassengerVehicle.java

```
@MappedSuperclass
public abstract class PassengerVehicle extends Vehicle {
    private int noOfpassengers;

    public PassengerVehicle(String model) {
        super(model);
    }

    // Getters and setters
}
```

Table Per Class Strategy: Example (3)

SoftUni Foundation

Truck.java

```
@Entity
public class Truck extends TransportationVehicle
{
    private final static String model = "CAR";
    private int noOfContainers;
    // Getters and setters
}
```

Car.java

```
@Entity
public class Car extends PassengerVehicle {
    private final static String model = "CAR";
    public Car(){
        super(model);
    }
}
```

Results - Joined Strategy

- After persist:

vehicles	
id	model
1	CAR
2	TRUCK

cars	
id	noOfPassengers
1	2

trucks		
id	noOfContainers	loadCapacity
1	2	5

Results - Joined Strategy

- **Disadvantages**:
  - Multiple JOINS - for deep hierarchies it may give poor performance
- **Advantages**:
  - No NULL values
  - No repeating information
  - Foreign keys involved
  - Reduced changes in schema on superclass changes

Table Per Class: Single Table

SoftUni Foundation

- **Simplest** and typically the best performing and best solution
  - A single table is used to store all of the instances of the **entire inheritance hierarchy**
  - A column for every attribute of every class
  - A **discriminator column** is used to determine to which class the particular row belongs to

Table Per Class strategy: Example

SoftUni Foundation

Vehicle.java

```
@Entity
@Table(name = "vehicles")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "type")
public abstract class Vehicle {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    private int id;
    @Basic
    private String model;
    protected Vehicle() {}
    protected Vehicle(String model) {
        this.model = model;
    }
}
```

Inheritance type

A table generator is used for each table

Table Per Class strategy: Example (2)

SoftUni Foundation

TransportationVehicle.java

```
@MappedSuperclass
public abstract class TransportationVehicle extends
Vehicle {
    private int loadCapacity;

    // Getters and setters
}
```

Table Per Class strategy: Example (2)

SoftUni Foundation

PassengerVehicle.java

```
@MappedSuperclass
public abstract class PassengerVehicle extends Vehicle {

    private int noOfpassengers;

    public PassengerVehicle(String model) {
        super(model);
    }

    // Getters and setters
}
```

Table Per Class strategy: Example (3)

SoftUni Foundation

Truck.java

```
@Entity
@DiscriminatorValue(values = "truck")
public class Truck extends TransportationVehicle
{
    private final static String model = "TRUCK";
    private int noOfContainers;
    // Getters and setters
}
```

Car.java

```
@Entity
@DiscriminatorValue(values = "car")
public class Car extends PassengerVehicle {
    private final static String model = "CAR";
    public Car(){
        super(model);
    }
}
```

Results - Joined strategy

- After persist:

vehicles				
id	type	loadCapacity	noOfPassengers	noOfContainers
1	truck	…	…	…
2	car	…	…	…

Discriminator column

Table Relations

One-to-One, One-to-Many, Many-to-Many

Database Relationships

SoftUni Foundation

- There are several types of database relationships:
  - **One to One** Relationships
  - **One to Many** and **Many to One** Relationships
  - **Many to Many** Relationships
  - **Self Referencing** Relationships

One-To-One - Unidirectional

One-To-One - Unidirectional

SoftUni Foundation

BasicShampoo.java

```
@Entity
@Table(name = "shampoos")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class BasicShampoo implements Shampoo {

//…

    @OneToOne(optional = false)
    @JoinColumn(name = "label_id",
    referencedColumnName = "id")
    private BasicLabel label;

//…
}
```

One-To-One relationship

Runtime evaluation

Column name in table labels

Column name in table shampoos

One-To-One - Bidirectional

One-To-One - Bidirectional

SoftUni Foundation

BasicLabel.java

```
@Entity
@Table(name = "labels")
public class BasicLabel implements Label{
//…

    @OneToOne(mappedBy = "label",
    targetEntity = BasicShampoo.class)
    private BasicShampoo basicShampoo;

//…
}
```

Field in entity BasicShampoo

Entity for the mapping

Many-To-One - Unidirectional

Many-To-One - Unidirectional

SoftUni Foundation

BasicShampoo.java

```
@Entity
@Table(name = "shampoos")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class BasicShampoo implements Shampoo {

//…
    @ManyToOne(optional = false)
    @JoinColumn(name = "batch_id", referencedColumnName = "id")
    private ProductionBatch batch;
//…
}
```

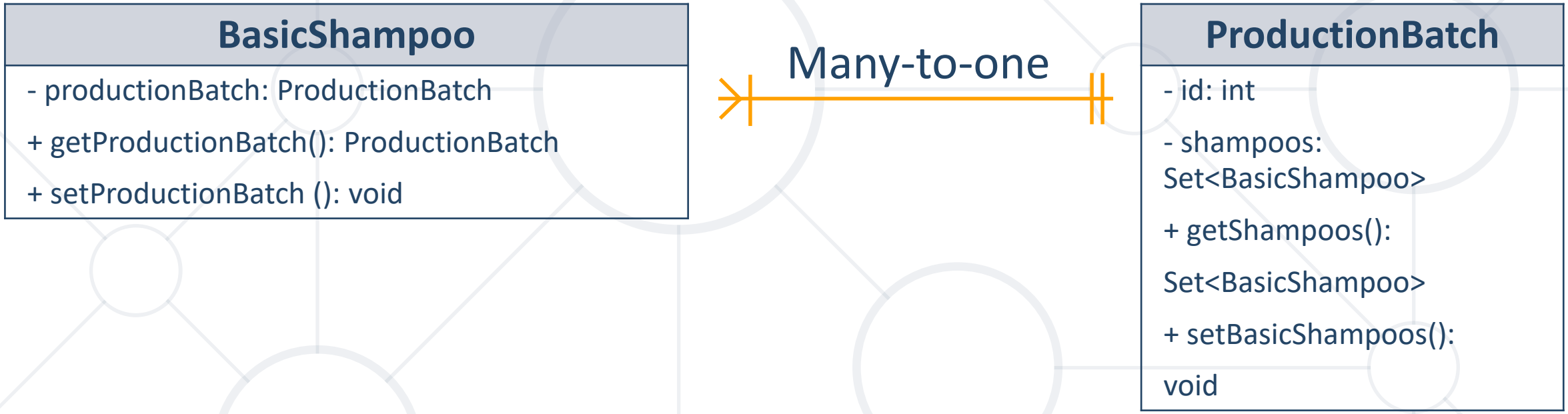
Many-To-One relationship

Runtime evaluation

Column name in table shampoos

Column name in table batches

One-To-Many - Bidirectional



One-To-Many - Bidirectional

SoftUni Foundation

ProductionBatch.java

```
@Entity
@Table(name = "batches")
public class ProductionBatch implements Batch {
//…
    @OneToMany(mappedBy = "batch", targetEntity = BasicShampoo.class,
            fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private Set<Shampoo> shampoos;

//…
}
```

Field in entity BasicShampoo

Entity for the mapping

Fetching type

Cascade type

Many-To-Many - Unidirectional

SoftUni Foundation

BasicShampoo.java

```
@Entity
@Table(name = "shampoos")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class BasicShampoo implements Shampoo {

//…
    @ManyToMany
    @JoinTable(name = "shampoos_ingredients",
    joinColumns = @JoinColumn(name = "shampoo_id", referencedColumnName = "id"),
    inverseJoinColumns = @JoinColumn(name = "ingredient_id", referencedColumnName
= "id"))
    private Set<BasicIngredient> ingredients;
//…
}
```

Many-To-Many relationship

Mapping table

Column in shampoos

Column in ingredients

Column in mapping table

Many-To-Many - Bidirectional

SoftUni Foundation

BasicIngredient.java

```
@Entity
@Table(name = "ingredients")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "type", discriminatorType = Discri
minatorType.STRING)
public abstract class BasicIngredient implements Ingredient {
//…

    @ManyToMany(mappedBy = "ingredients", targetEntity = Basic
Shampoo.class)
    private Set<BasicShampoo> shampoos;

//…
}
```

Field in entity BasicShampoo

Entity for the mapping

Lazy Loading - Fetch Types

- Fetching – retrieve objects from the database
  - Fetched entities are stored in the **Persistence Context** as cache
- Retrieval of an entity object might cause automatic retrieval of **additional** entity objects

Fetching Strategies

SoftUni Foundation

- Fetching Strategies
  - EAGER – retrieves all entity objects reachable through fetched entity
    - Can cause **slowdown** when used with a big data source
  - **LAZY** – retrieves all reachable entity objects **only when fetched entity's getter method is called**

```
University university = em.find((long) 1); // this.students = null

// The collection holding the students is populated when the getter is called
university.getStudents();
```

Cascading

SoftUni Foundation

- JPA translates **entity state transitions** to database **DML** statements
  - This behavior is configured through the **CascadeType** mappings
- **`CascadeType.PERSIST`**: means that save() or persist() operations cascade to related entities
- **`CascadeType.MERGE`**: means that related entities are merged int o managed state when the owning entity is merged
- **`CascadeType.REFRESH`**: does the same thing for the refresh() operation

Cascading (2)

- **CascadeType.REMOVE**: removes all related entities association with this setting when the owning entity is deleted
- **CascadeType.DETACH**: detaches all related entities if a "manual detach" occurs
- **CascadeType.ALL**: is shorthand for all of the above cascade operations

Summary

- Relational databases don't support inheritance
- It is implemented by JPA:
  - SINGLE_TABLE
  - TABLE_PER_CLASS
  - JOINED
- Table relations are Un/Bidirectional
- One-to-One
- Many-to-One
- Many-to-Many

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

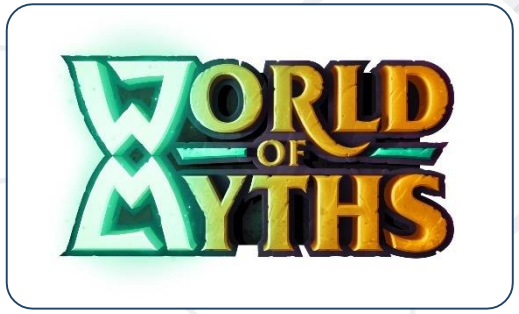
LIEBHERR

æternity

codeXio

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