MONASH University

MONASH INFORMATION TECHNOLOGY

FIT5192 Lecture 7: Advanced Application Java Persistence – Part I

GROUP OF EIGHT AUSTRALIA

This Lecture

- More advanced ORM

MONASH University

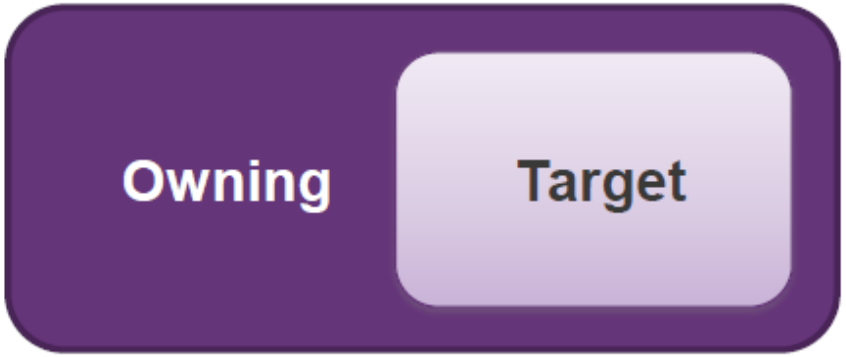
Object Relational Mapping (ORM)

Object-Relational Mapping

- Mapping data in object-oriented model to relational structure and vice versa.
- We only cover the most elementary mapping in the Intro to Java Persistence API
- This lecture, we will look at the mapping of:
  - Composition
  - Collection
  - Cardinality
  - Inheritance

Composition (1)

- A very common design in OO paradigm
- Two objects have a composition relationship, when one of them only exists as an intrinsic part of another.



- That means, the lifetime of the target object depends on that of the owning object.

Composition (2)

- When being mapped to a database, a target entity does not have its own persistent identity. It is stored as part of the owning entity and shares the identity of the owning entity.
- Mapped using
  - `@Embeddable`: on the target side
  - `@Embedded`: on the owning side

Composition Mapping

Indicates the objects of this class can be included as a target object

```
@Embeddable
@Access(AccessType.PROPERTY)
public class PhoneNumber implements Serializable {

    private String countryCode;
    private String areaCode;
    private String phoneNumber;
```

```
@Entity
public class Staff implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "staff_id")
    private int staffId;
    private String name;
    @Embedded
    private PhoneNumber contactNumber;
```

Mark the attribute as a target object

```
Staff staff1 = new Staff("Eddie Leung", new PhoneNumber("61", "03", "98778987"));
entityManager.persist(staff1);
```

STAFF_ID	NAME	COUNTRY_CODE	AREA_CODE	PHONE_NUMBER
1	Eddie Leung	61	03	98778987

Access Type of an Embeddable Class

- By default, the access type of an embeddable class is determined by the access type of the owing entity class.
- If objects of an embeddable class is owned by multiple entity classes, problems may arise.

- As a result, explicitly specifying access type using `@Access` is strongly recommended.

  `@Access(` value=[FIELD,PROPERTY])

Collection

- A group of objects of:
  – basic types (i.e. non-entities) e.g. `List<Integer>`
  – Embeddable e.g. `Set<PhoneNumber>`
- Support data structures:
  - `java.util.Collection`
  - `java.util.Set`
  - `java.util.List`
- Mapped using `@ElementCollection`
- Customize settings using `@CollectionTable`
- Unless specified, the default table name is:
  Name of containing entity + "_" + attribute name (e.g. MOVIE_TAGS)

Map

- A group of objects that are stored as key-value.
- Since JPA 2.0, key and value can be of any types (e.g. basic types, embeddable objects, entities)
- Mapped using `@ElementCollection`
- Customize settings using `@CollectionTable`, `@MapKeyColumn` and `@Column`
- By default, the name of the key & value of a map is mapped to :
  - Key: The name of the referencing table + "S_KEY" (E.g. CHAPTERS_KEY)
  - Value: The name of the referencing table + "S" (E.g. CHAPTERS)

Relationship Mapping

- Similar to records in relational database, objects often have relationships with each other.
- In ORM, we need to map the relationships in one to another

Relationship Directions

- Unlike relational database design, these relationships have directions.
- The direction of a relationship indicates whether object(s) on one side are "aware" of that on another.
- A relationship can be either ***unidirectional*** or ***bidirectional***.

Unidirectional Relationship

- Object(s) on one side are NOT aware of that on another.
- In UML, an arrow is used to indicate the orientation

- In Java, the direction is represented by the source class having an attribute of the target class e.g. Class1 having an attribute of type Class 2.

Bidirectional Relationship

- Object(s) on BOTH sides are "aware" of that on another.
- In UML, a line (with no arrow) is used to indicate the relationship.

- In Java, the direction is represented by both classes having an attribute of each other e.g. Class1 has an attribute of type Class2 and Class2 has an attribute of type Class1

Cardinality (1)

- Similar to relational database design, object oriented data model has cardinality.
- Specify the minimum and maximum number of referring objects are involved in the relationship.
- In Java, the data structure used to store the attribute of each other indicates the cardinality

Cardinality (2)

UML Notation	Min. No. of Objects	Max. No. of Objects	Java Attribute
1	1	1	Single object
0..1	0	1	Single object (null accepted)
0..*	0	As many as needed	Dynamic data structure (e.g. List, Set, Map and etc.)
2..5	2	5	An array of size 5

Cardinality (3)

Cardinality	Direction	Representation in Java
One-to-one	Unidirectional	Class1 has an Class2 object as attribute.
One-to-one	Bidirectional	Class1 has an Class2 object as attribute, and Class2 has an Class1 object as attribute.
One-to-many	Unidirectional	Class1 has a collection of Class2 objects as attribute.
One-to-many	Bidirectional	Class1 has a collection of Class2 objects as attribute, and Class2 has a single Class1 objects as attribute.
Many-to-one	Unidirectional	Class1 has a single Class2 objects as attribute
Many-to-one	Bidirectional	Class1 has a single Class2 objects as attribute, and Class2 has a collection of Class1 objects as attribute.
Many-to-many	Unidirectional	Class1 has a collection of Class2 objects as attribute.
Many-to-many	Bidirectional	Class1 has a collection of Class 2 objects as attribute, and Class2 has a collection of Class1 objects as attribute.

Summary

- More advanced ORM

Readings

- ***Recommended:*** Chapter 6: Managing Persistence Objects in *Beginning Java EE 7, Antonio Goncalves*