Java Persistence API

Part-III

Annotation المسئولة عن تعريف Entity

- @Entity
- تستخدم لعمل mapping مابين class وجدول في قاعدة البيانات.
- @Table
- تستخدم لتحديد أسم الجدول الذي سيتم عمل g mapping له مع class .

```
@Entity
@Table(name = "t_book")
public class Book {

@Id
  private Long id;
  private String title;
  private Float price;
  private String description;
  private String isbn;
  private Integer nbOfPage;
  private Boolean illustrations;

  // Constructors, getters, setters
}
```

Composite Primary Key

- عندما تكون هناك حاجة للتعامل مع composite primary key يتم تكوين class تمثل composite key مع استخدام احد annotations التالية:
- @IdClass
- @EmbeddedId

Composite Primary Key

- في هذه الطريقة يتم تكوين class تحتوي على primary keys .
- هذه class يجب ان تكون public وتحتوي على no-arg construction أضافة إلى كل من الدوال ()equal و ()hashCode.
- تضمين هذه class في entity التي تحتاج composite key مع استخدام @IdClass annotation .
- Entity التي ستستخدم هذه class يجب ان تحتوي على نفس attributes الموجودة فيها مع عمل annotation باستخدام @Id .

```
public class NewsId {

private String title;
  private String language;

  // Constructors, getters, setters, equals, and hashcode
}
```

```
@Entity
@IdClass(NewsId.class)
public class News {

@Id private String title;
@Id private String language;
private String content;

  // Constructors, getters, setters
}
```

@EmbeddedId

- في هذه الطريقة يتم تكوين class تحتوي على primary keys مع عمل annotation لها باستخدام @Embeddable .
- هذه class يجب ان تكون public وتحتوي على no-arg construction أضافة إلى كل من الدوال ()equal و ()hashCode.
- تضمين هذه class في entity التي تحتاج composite key مع استخدام @EmbeddedId annotation .

```
@Embeddable
public class NewsId {

  private String title;
  private String language;

  // Constructors, getters, setters, equals, and hashcode
}
```

```
@Entity
public class News {

  @EmbeddedId
  private NewsId id;
  private String content;

  // Constructors, getters, setters
}
```

```
NewsId pk = new NewsId("Richard Wright has died on September 2008", "EN")
News news = em.find(News.class, pk);
```

@Column

- هي annotation أختيارية تستخدم لتحديد أسم العمود في قاعدة البيانات الذي ستقوم entity attribute بعمل mapping معه.
- كما يمكن عن طريقها تحديد هل سيسمح بعمل عمليات ادخال او تحديث لقيمة هذه attribute .

```
@Entity
public class Book {

    @Column(name = "title", updatable = false, insertable = true)
    private String title;

    ...
}
```

@Id

- كل entity تحتاج ان يكون لديها primary key attribute يتم ذلك من خلال استخدام هذه annotation على احد attribute .

```
@Entity
public class Author {

    @Id
    private Long id;

    ...
}
```

@GeneratedValue

- تستخدم للحصول على قيمة Id المستخدم في Entity وهي توفر أربع طرق للحصول على هذه القيمة.
- GenerationType.IDENTITY
- GenerationType.SEQUENCE
- GenerationType.TABLE
- GenerationType.AUTO

```
@Entity
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    ...
}
```

@Enumerated

- تسمح لنا بتحديد طريق تخزين enum في قاعدة البيانات.

```
@Entity
public class Author {

    @Enumerated(EnumType.STRING)
    private AuthorStatus status;

    ...
}
```

@Temporal

- عند استخدام java.util.Date او java.util.Calendar في Entity يمكن عن طريقها تحديد نوع mapping المستخدم مثل Date، TIME او TIMESTAMP.

```
@Entity
public class Author {

    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;

    ...
}
```

@Lob

- تستخدم لعمل map بين النوع BLOB في قاعدة البيانات و مصفوفة من البايت byte[].

```
@Entity
public class Book {

    @Lob
    private byte[] cover;

    ...
}
```

@Transient

- تستخدم عندما لا نحتاج لعمل mapping لاحد attribute في Entity.

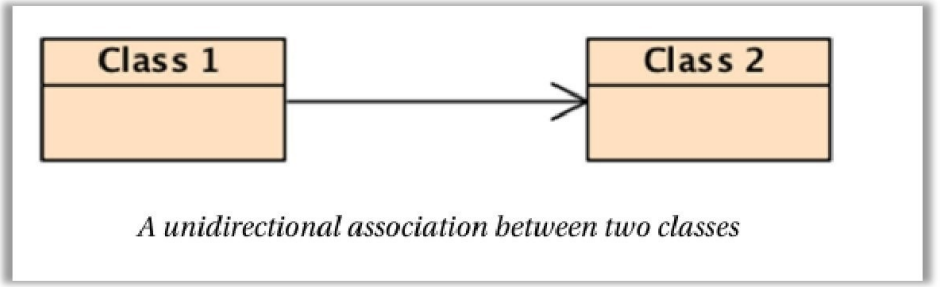
```
@Entity
public class Customer {

  @Id @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private String email;
  private String phoneNumber;
  @Temporal(TemporalType.DATE)
  private Date dateOfBirth;
  @Transient
  private Integer age;
  @Temporal(TemporalType.TIMESTAMP)
  private Date creationDate;

  // Constructors, getters, setters
}
```

@Basic

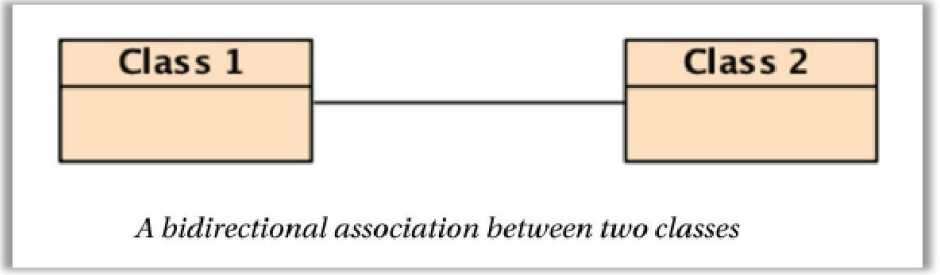
- تستخدم لتحديد ما اذا كانت قيمة هذه attribute يمكن ان تكون null ، وكذلك تستخدم لتحديد طريقة جلب البيانات هل هي EAGER أو LAZY .

```
@Entity
public class Track {

  @Id @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String title;
  private Float duration;
  @Basic(fetch = FetchType.LAZY)
  @Lob
  private byte[] wav;
  private String description;

  // Constructors, getters, setters
}
```

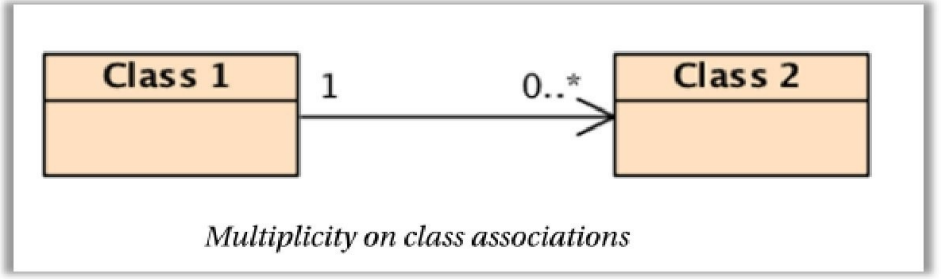
Relationship Mapping



A unidirectional association between two classes

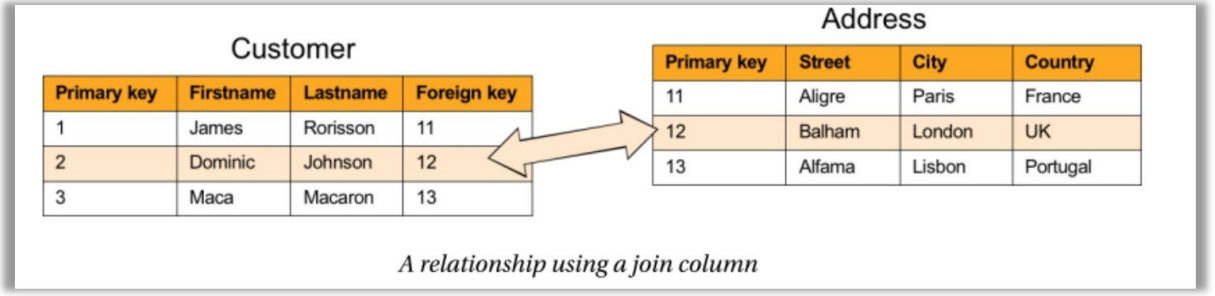


A bidirectional association between two classes



Multiplicity on class associations

Relationship Mapping in DataBase



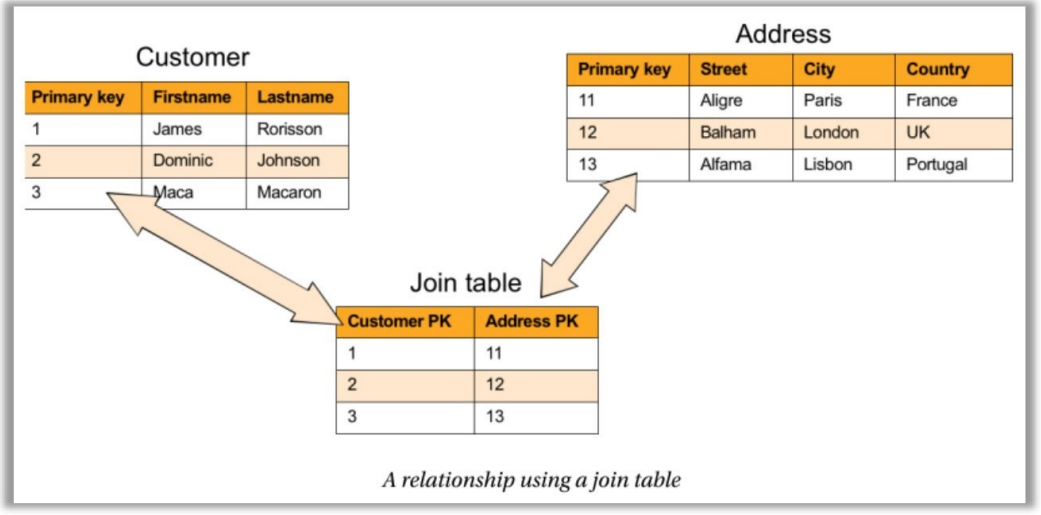
Customer

Primary key	Firstname	Lastname	Foreign key
1	James	Rorisson	11
2	Dominic	Johnson	12
3	Maca	Macaron	13

Address

Primary key	Street	City	Country
11	Aligre	Paris	France
12	Balham	London	UK
13	Alfama	Lisbon	Portugal

A relationship using a join column



Customer

Primary key	Firstname	Lastname
1	James	Rorisson
2	Dominic	Johnson
3	Maca	Macaron

Address

Primary key	Street	City	Country
11	Aligre	Paris	France
12	Balham	London	UK
13	Alfama	Lisbon	Portugal

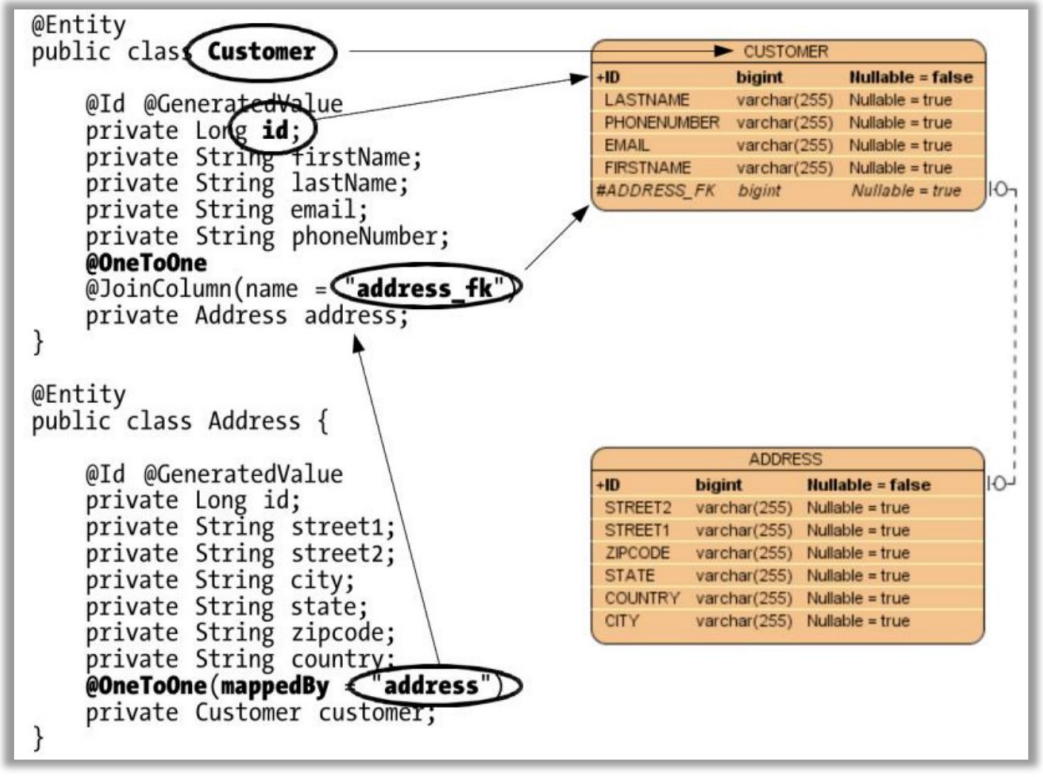
Join table

Customer PK	Address PK
1	11
2	12
3	13

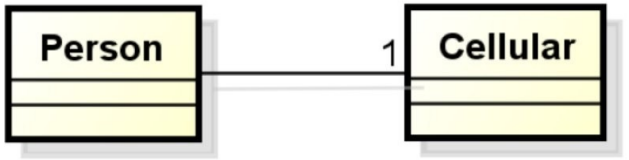
A relationship using a join table

Example

- المثال التالي يبين كيفية تمثيل العلاقة بين كل من Customer class و Address class.



One-To-One Unidirectional




```java
import javax.persistence.*;

@Entity
public class Person {

    @Id
    @GeneratedValue
    private int id;

    private String name;

    @OneToOne
    @JoinColumn(name="cellular_id")
    private Cellular cellular;

    // get and set
}
```

```java
import javax.persistence.*;

@Entity
public class Cellular {

    @Id
    @GeneratedValue
    private int id;

    private int number;

    // get and set
}
```

One-To-One Bidirectional

```java
import javax.persistence.*;

@Entity
public class Person {

    @Id
    @GeneratedValue
    private int id;

    private String name;

    @OneToOne
    @JoinColumn(name="cellular_id")
    private Cellular cellular;

    // get and set
}
```

```java
import javax.persistence.*;

@Entity
public class Cellular {

    @Id
    @GeneratedValue
    private int id;

    private int number;

    @OneToOne(mappedBy="cellular")
    private Person person;

    // get and set
}
```

One-To-Many/Many-To-One

```java
import javax.persistence.*;

@Entity
public class Call {

    @Id
    @GeneratedValue
    private int id;

    @ManyToOne
    @JoinColumn(name = "cellular_id")
    private Cellular cellular;

    private long duration;

    // get and set
}
```

```java
@Entity
public class Cellular {

    @Id
    @GeneratedValue
    private int id;

    @OneToOne(mappedBy = "cellular")
    private Person person;

    @OneToMany(mappedBy = "cellular")
    private List<Call> calls;

    private int number;

    // get and set
}
```

Many-To-Many

```java
import java.util.List;

import javax.persistence.*;

@Entity
public class Person {

    @Id
    @GeneratedValue
    private int id;

    private String name;

    @ManyToMany
    @JoinTable(name = "person_dog", joinColumns = @JoinColumn(name = "person_id"), inverseJoinColumns =
@JoinColumn(name = "dog_id"))
    private List<Dog> dogs;

}
```

```java
@Entity
public class Dog {

    @Id
    @GeneratedValue
    private int id;

    private String name;

    @ManyToMany(mappedBy="dogs")
    private List<Person> persons;

    // get and set
}
```

JPA Cascade Types

- تدعم JPA أنواع cascade التالية.

1. **CascadeType.PERSIST** : means that persist() operations cascade to related entities.
2. **CascadeType.MERGE** : means that related entities are merged when the owning entity is merged.
3. **CascadeType.REFRESH** : does the same thing for the refresh() operation.
4. **CascadeType.REMOVE** : removes all related entities association with this setting when the owning entity is deleted.
5. **CascadeType.DETACH** : detaches all related entities if a “manual detach” occurs.
6. **CascadeType.ALL** : is shorthand for all of the above cascade operations.