Persistence & Transactions

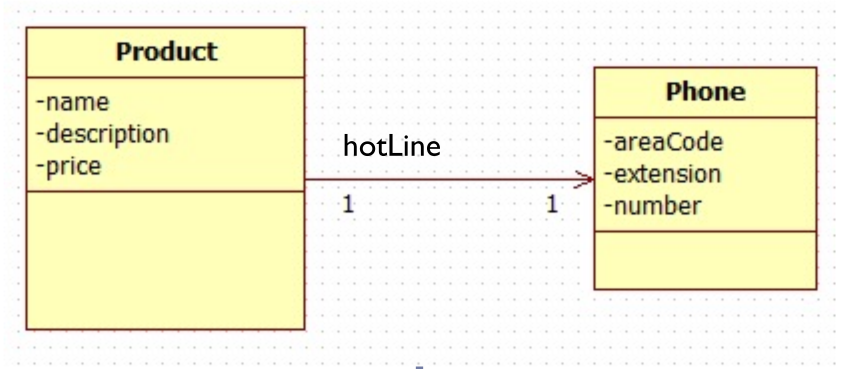
Topping the Source of Pure Knowledge

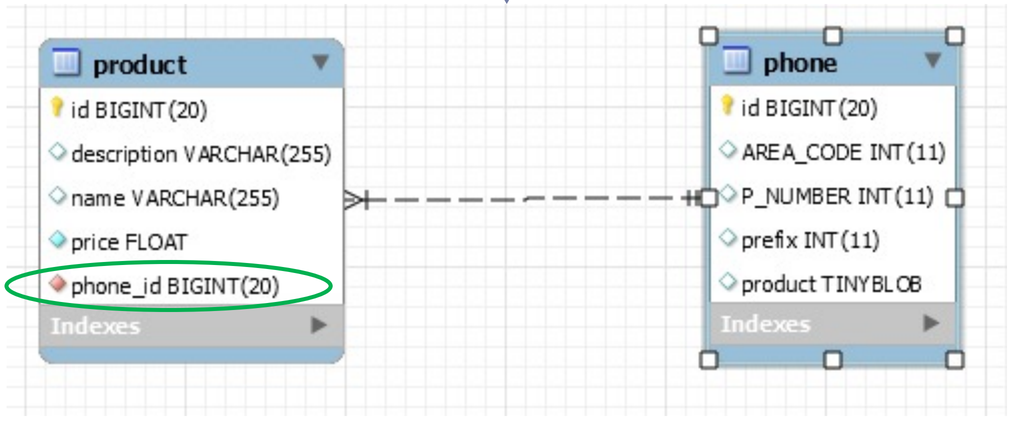
ORM Relationships

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

- Unidirectional - Bidirectional

OneToOne Unidirectional





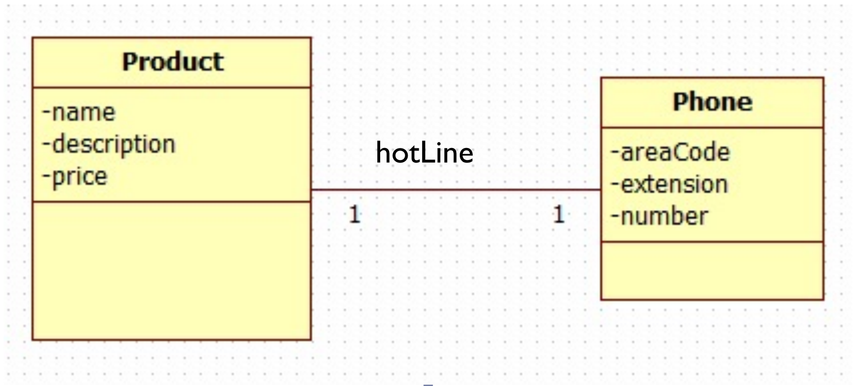
OneToOne Unidirectional

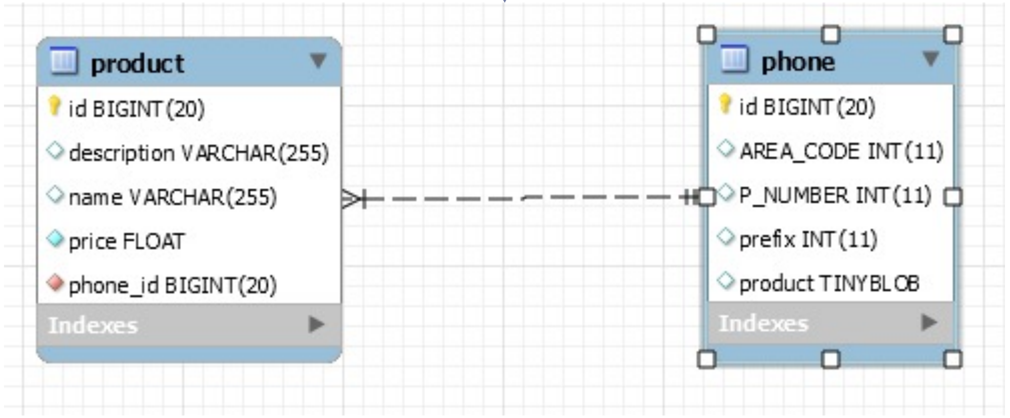
```
@Entity
public class Product implements
   Serializable {

   @Id
   @GeneratedValue(strategy=Generat
      ionType.AUTO)
   private long id;

   private String name;
   private String description;
   private float price;

   @OneToOne(cascade =
      CascadeType.ALL)
   @JoinColumn(name = "phone_id",
      nullable = false)
   private Phone hotLine;
}
```

```
@Entity
public class Phone implements
   Serializable {

   @Id
   @GeneratedValue(strategy=Generat
      ionType.AUTO)
   private long id;

   private Integer areacode;
   private Integer number;
   private Integer prefix;
}
```

Default is JoinColumn

OneToOne Bi-directional





OneToOne Bi-directional

- Annotation the OTHER side of the relationship ALSO…

```
@Entity
public class Product implements
   Serializable {

   @Id
   @GeneratedValue(strategy=GenerationT
      ype.AUTO)
   private long id;

   private String name;
   private String description;
   private float price;

   @OneToOne(cascade = CascadeType.ALL)
   @JoinColumn(name = "phone_id",
      nullable = false)
   private Phone hotLine;
}
```

```
@Entity
public class Phone implements
   Serializable {

   @Id
   @GeneratedValue(strategy =
      GenerationType.AUTO)
   private long id;

   private Integer areacode;
   private Integer number;
   private Integer prefix;

   @OneToOne(mappedBy = "hotLine",
      cascade = CascadeType.ALL)
   private Product product;
}
```

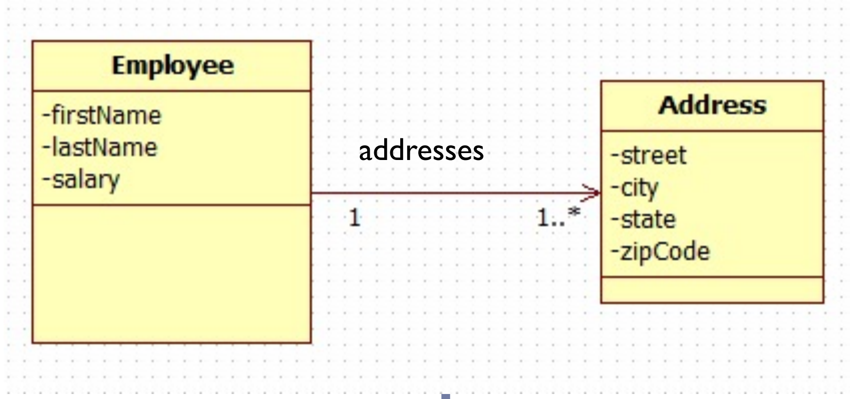
mappedBy – use the foreign key and mapping in the *source* to define the *target* mapping

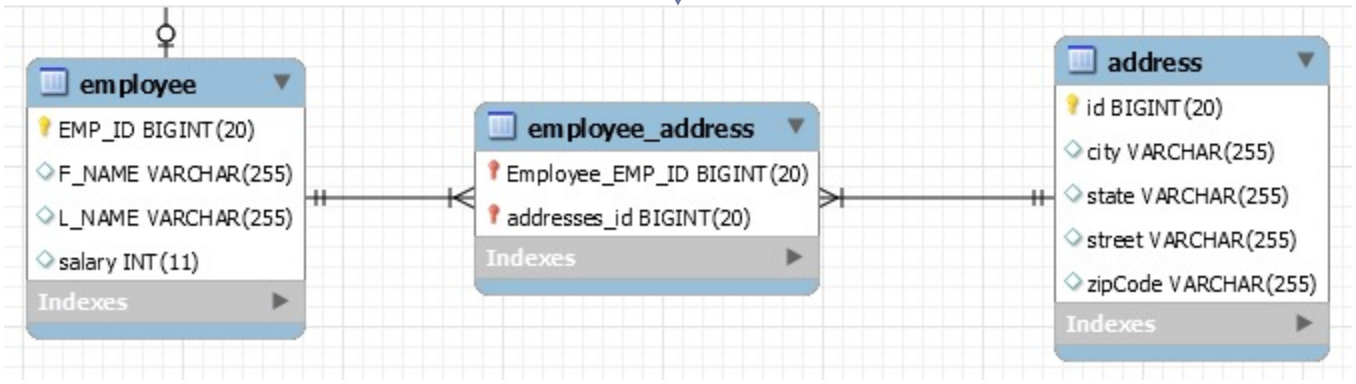
Bi-directional Relationships

WARNING NOTICE

- If you add or remove to one side of the collection, you **must** also add or remove from the other side
- Database will be updated correctly **ONLY** if you add/remove from the owning side of the relationship
- Your object model can get out of sync if you do not pay attention…

One-to-Many Join Table





OneToMany Unidirectional JoinTable

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "EMP_ID")
    private long id;

    @Column(name = "F_NAME")
    private String firstName;

    @Column(name = "L_NAME")
    private String lastName;

    private Integer salary;

    @OneToMany(cascade = CascadeType.ALL)
    // FetchMode.JOIN will do eager load also
    @Fetch(FetchMode.JOIN)
    private List<Address> addresses;
}
```

```
@Entity
public class Address implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    private String street;
    private String city;

    private String state;

    private String zipCode;
}
```

This is the Default

OneToMany Bidirectional JoinTable

OneToMany side same as unidirectional example

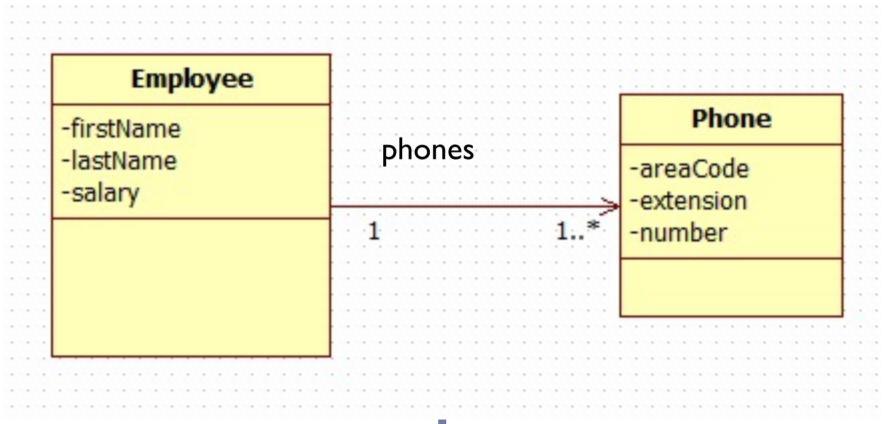
```
@Entity
@Table(name = "emp")
public class Employee implements Serializable {

    @Id
    @GeneratedValue(strategy =
        GenerationType.AUTO)
    @Column(name = "EMP_ID")
    private long id;

    @Column(name = "F_NAME")
    private String firstName;

    @OneToMany(cascade = CascadeType.ALL)
    // FetchMode.JOIN will do eager load also
    @Fetch(FetchMode.JOIN)
    @JoinTable
    private List<Address> addresses;
}
```

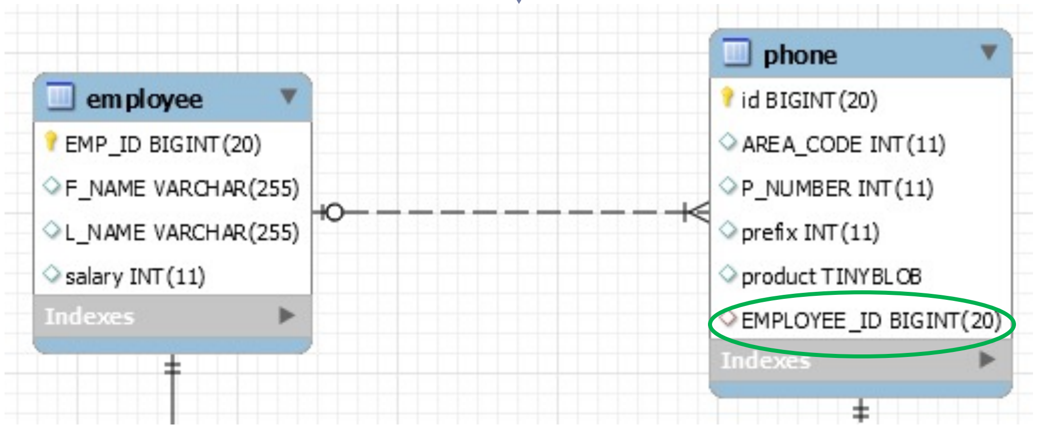
Simply Add ManyToOne on child object

```
@Entity
public class Address implements Serializable {
    @Id
    @GeneratedValue(strategy =
        GenerationType.AUTO)
    private long id;

    private String street;
    private String city;

    private String state;

    private String zipCode;

    @ManyToOne
    private Employee employee;
}
```

https://www.solidsyntax.be/2013/10/17/fetching-collections-hibernate/

OneToMany Unidirectional JoinColumn





OneToMany Unidirectional JoinColumn

```
@Entity
@Table(name = "emp")
public class Employee implements
   Serializable {

   @Id
   @GeneratedValue(strategy =
      GenerationType.AUTO)
   @Column(name = "EMP_ID")
   private long id;

   @Column(name = "F_NAME")
   private String firstName;

   @OneToMany(cascade =
      CascadeType.ALL)
   @Fetch(FetchMode.JOIN)
   @JoinColumn(name = "EMPLOYEE_ID")
   private List<Phone> phones;

}
```

```
@Entity
public class Phone implements
   Serializable {

   @Id
   @GeneratedValue(strategy=GenerationT
     ype.AUTO)
   private long id;

   private Integer areacode;
   private Integer number;
   private Integer prefix;
}
```

HIBERNATE REFERENCE DOC:
A *unidirectional one-to-many association on a foreign key* is an unusual case, and is not recommended. You should instead use a join table for this kind of association.
* *Add a column to Phone*

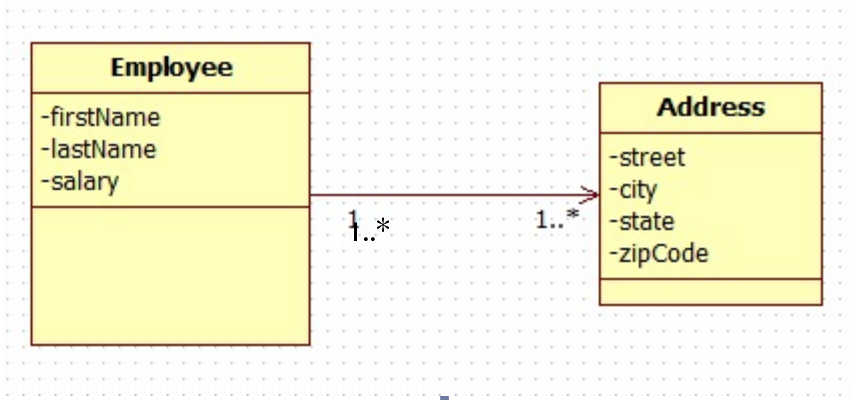
OneToMany Bi-directional JoinColumn

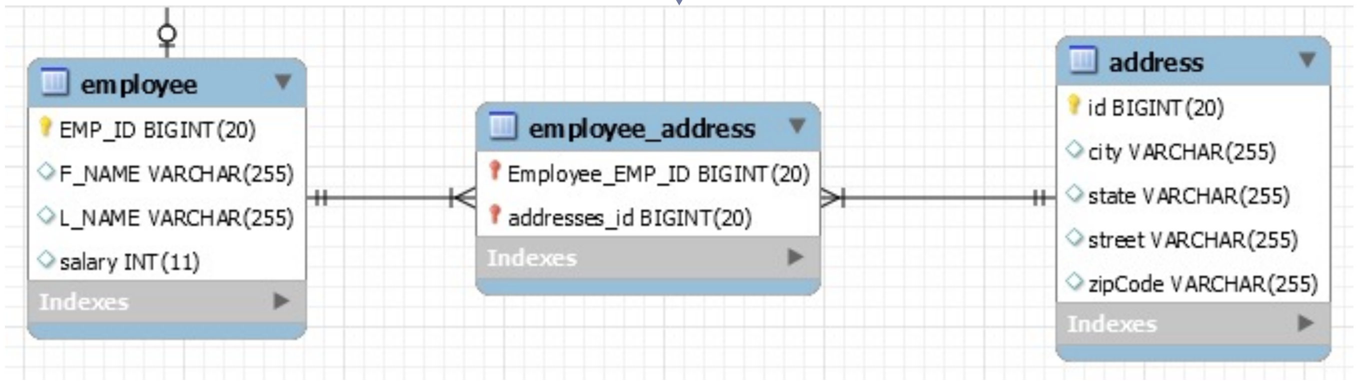
```
@Entity
@Table(name = "emp")
public class Employee implements
   Serializable {

   @Id
   @GeneratedValue(strategy =
      GenerationType.AUTO)
   @Column(name = "EMP_ID")
   private long id;

   @Column(name = "F_NAME")
   private String firstName;

   @OneToMany(cascade = CascadeType.ALL,
      mappedBy = "employee")
   private List<Phone> phones;

}
```

```
@Entity
public class Phone implements
   Serializable {

   @Id
   @GeneratedValue(strategy =
      GenerationType.AUTO)
   private long id;

   private Integer areacode;
   private Integer number;
   private Integer prefix;

   @ManyToOne
   @JoinColumn(name = "EMP_ID")
   private Employee employee;

}
```

Owns relationship

Many-to-Many


Employee
-firstName
-lastName
-salary
Address
-street
-city
-state
-zipCode
1..*
1..*


employee
EMP_ID BIGINT(20)
F_NAME VARCHAR(255)
L_NAME VARCHAR(255)
salary INT(11)
Indexes
employee_address
Employee_EMP_ID BIGINT(20)
addresses_id BIGINT(20)
Indexes
address
id BIGINT(20)
city VARCHAR(255)
state VARCHAR(255)
street VARCHAR(255)
zipCode VARCHAR(255)
Indexes

Many-To-Many

```
@Entity
@Table(name = "emp")
public class Employee implements
   Serializable {

   @Id
   @GeneratedValue(strategy =
      GenerationType.AUTO)
   @Column(name = "EMP_ID")
   private long id;

   @ManyToMany(cascade = {
      CascadeType.ALL })
   @JoinTable()
   Set<Project> projects = new
      HashSet<>();
}
```

```
@Entity
public class Project implements
   Serializable{

   @Id
   @GeneratedValue(strategy =
      GenerationType.AUTO)
   private long id;

   private String name;

   @ManyToMany(mappedBy = "projects")
   private Set<Employee> employees =
      new HashSet<>();
}
```

If Converting from OneToMany [Join table] – The ManyToMany is achieved by simply dropping the unique constraint on the JoinTable created by OneToMany

Main Point

- JPA is a specification not an implementation. It provides a consistent, reliable mechanism for data storage and retrieval that alleviates the application developer from the details involved in the persistence layer.
- *The mechanism of transcending allows the individual to tap into Transcendental Consciousness and enlivens its qualities in activity.*

CrudRepository

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {

    <S extends T> S save(S entity);

    <S extends T> Iterable<S> saveAll(Iterable<S> entities);

    Optional<T> findById(ID id);

    boolean existsById(ID id);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> ids);

    long count();

    void deleteById(ID id);

    void delete(T entity);

    void deleteAll(Iterable<? extends T> entities);

    void deleteAll();
}
```

LOOKS just Like [what is Known as] a "genericDAO interface"
HOWEVER, Spring provides [default] implementations – effectively Java 8-like default methods in an interface

Spring Data Repository Query Resolution Query examples - CREATE example

- CREATE example
  - attempts to construct a store-specific query from the query method name.
  - The name of our query method must start with one of the following prefixes: `find…By`, `read…By`, `query…By`, `count…By`, and `get…By`.
  - limit the number of returned query results: `findTopBy`, `findTop8By`, `findFirstBy`, and `findFirst8By`
  - select unique results: `findTitleDistinctBy` or `findDistinctTitleBy`

```java
@Repository
public interface PhoneRepository extends CrudRepository<Phone, Long> {
    public List<Phone> findByAreacodeOrPrefix(String areacode, String prefix);
    public long countByAreacode(String areacode);
}
```

Query Key Words   Query lookup strategies

JPQL - Data Object Queries

- JPQL is similar to SQL, but operates on objects, attributes and relationships instead of tables and columns.

```
@Entity
public class Product implements Serializable {
    private String name;
    @OneToOne(cascade = CascadeType.ALL)
    private Phone hotLine;
}
```

- `JPQL:`
  - **SELECT p FROM Product p**
- Will Yield:
  - **Product with Phone;**
- Where:
  - **product.getHotLine().getNumber();** is populated

- **NOTE: JPA `OneToOne relationship defaults to eager`**

Spring Data Repository Query Resolution Query examples - USE_DECLARED_QUERY

- USE_DECLARED_QUERY
  - tries to find a declared query and will throw an exception in case it can't find one.

- **JPQL Queries**

```
@Query(value = "SELECT e FROM Employee e WHERE e.lastName = :lastname")
public List<Employee> findByLastName(String lastname);
```

- **SQL Queries**

```
@Query(value = "SELECT * FROM emp e WHERE e.F_NAME = ?1", nativeQuery = true)
public List<Employee> findByFirstName(String firstName);
```

- **CLASS LEVEL DECLARED**

```
public final static  String FIND_BY_SALARY_QUERY = "SELECT e FROM
   Employee e WHERE e.salary = :salary";

@Query(FIND_BY_SALARY_QUERY)
public List<Employee> findByAddress(@Param("salary") Integer salary);
```

Spring Data Repository Query Resolution Query examples - JPA Named Query

- Declaration:

```
@Entity
@Table(name = "emp")
@NamedQuery(name = "Employee.findEmployeesByLastName", query = "SELECT e FROM Employee e
   WHERE LOWER(e.lastName) = LOWER(:lastName)")
public class Employee implements Serializable {


}
```

Query name convention: @{EntityName}.{queryName}

- Usage:

```
@Repository
public interface EmployeeRepository extends CrudRepository<Employee, Long> {
   public List<Employee> findEmployeesByLastName(@Param("lastName") String lastName);
}
```

Main Point

- Spring provides a Transactional capability for ORM applications.
- The *mechanism of transcending allows the individual to tap into Transcendental Consciousness and enlivens its qualities in activity.*