# Part 2: Hibernate - JPA Annotations

This tutorial is part 2 of 5-part tutorial on JEE annotations. We recommend that you read Prerequisite section first, review the abstract and Example Application to understand the context. You can also jump to other parts by clicking on the links below.





## Hibernate JPA Annotations - Contents:

| Annotation | Package Detail/Import statement |
|---|---|
| @Entity | import javax.persistence.Entity; |
| @Table | import javax.persistence.Table; |
| @Column | import javax.persistence.Column; |
| @Id | import javax.persistence.Id; |
| @GeneratedValue | import javax.persistence.GeneratedValue; |
| @Version | import javax.persistence.Version; |
| @OrderBy | import javax.persistence.OrderBy; |
| @Transient | import javax.persistence.Transient; |
| | import javax.persistence.Lob; |
| Hibernate Association Mapping Annotations | |
| oOne | import javax.persistence.OneToOne; |
| ToOne | import javax.persistence.ManyToOne; |
| oMany | import javax.persistence.OneToMany; |
| ToMany | import javax.persistence.ManyToMany; |
| aryKeyJoinColumn | import javax.persistence.PrimaryKeyJoinColumn; |
| olumn | import javax.persistence.JoinColumn; |
| able | import javax.persistence.JoinTable; |
| @MapsId | import javax.persistence.MapsId; |
| Hibernate Inheritance Mapping Annotations | |
| @Inheritance | import javax.persistence.Inheritance; |
| @DiscriminatorColumn | import javax.persistence.DiscriminatorColumn; |
| @DiscriminatorValue | import javax.persistence.DiscriminatorValue; |



### @Entity

Annotate all your entity beans with @Entity.

```
@Entity
public class Company implements Serializable {
...
}
```

### @Table

Specify the database table this Entity maps to using the name attribute of @Table annotation. In the example below, the data will be stored in 'company' table in the database.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {
...
}
```

### @Column

Specify the column mapping using @Column annotation.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

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

...
}
```

### @Id

Annotate the id column using @Id.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

  @Id
  @Column(name = "id")
  private int id;

...
}
```

### @GeneratedValue

base generate (auto-increment) the id column.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

  @Id
  @Column(name = "id")
  @GeneratedValue
  private int id;

...
}
```

### @Version

Control versioning or concurrency using @Version annotation.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

  @Version
  @Column(name = "version")
  private Date version;

...
}
```

### @OrderBy

ır data using @OrderBy annotation. In example below, it will sort all contacts in a company by their firstname in ascending order.

```
@OrderBy("firstName asc")
private Set contacts;
```

### @Transient

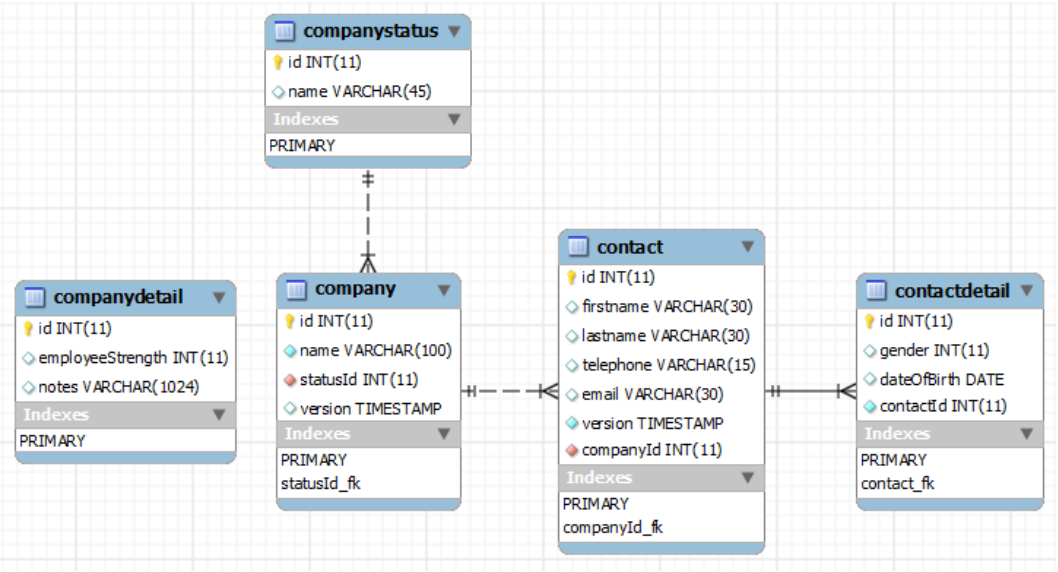
Annotate your transient properties with @Transient.

### @Lob

Annotate large objects with @Lob.

## Hibernate Association Mapping Annotations

### Example App DB Schema




The database for this tutorial is designed to illustrate various association mapping concepts.
In RDBMS implementations, entities are joined using the following ways:

- Shared Primary Key
- reign Key
- sociation Table

ample app,

bles company and companyDetail have shared values for primary key. It is a one-to-one assoication.
bles contact and contactDetail are linked through a foreign key. It is also a one to one association.
bles contact and company are linked through a foriegn key in many-to-one association with contact being the owner.
bles company and companyStatus are linked through a foreign key in many-to-one association with company being the owner.

### ToOne


- Use @PrimaryKeyJoinColumn for associated entities sharing the same primary key.
- Use @JoinColumn & @OneToOne mappedBy attribute when foreign key is held by one of the entities.
- Use @JoinTable and mappedBy entities linked through an association table.
- Persist two entities with shared key using @MapsId

For entities Company and CompanyDetail sharing the same primary key, we can associate them using @OneToOne and @PrimaryKeyJoinColumn as shown in the example below.

Notice that the id property of CompanyDetail is NOT annotated with @GeneratedValue. It will be populated by id value of Company.

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4
5    @Id
6    @Column(name = "id")
7    @GeneratedValue
8    private int id;
```

```
  @OneToOne(cascade = CascadeType.MERGE)
  @PrimaryKeyJoinColumn
  private CompanyDetail companyDetail;

  ...
}

@Entity
@Table(name = "companyDetail")
public class CompanyDetail implements Serializable {

  @Id
  @Column(name = "id")
  private int id;

  ...
}
```

For entities Contact and ContactDetail linked through a foriegn key, we can use @OneToOne and @JoinColumn annotations. In example below, the id genereated for Contact will be mapped to 'contact_id' column of ContactDetail table. Please note the usage of @MapsId for the same.

```
@Entity
@Table(name = "contactDetail")
public class ContactDetail implements Serializable {

  @Id
  @Column(name = "id")
  @GeneratedValue
  private int id;

  @OneToOne
  @MapsId
  @JoinColumn(name = "contactId")
  private Contact contact;

  ...
}

@Entity
@Table(name = "contact")
public class Contact implements Serializable {

  @Id
  @Column(name = "ID")
  @GeneratedValue
  private Integer id;

  @OneToOne(mappedBy = "contact", cascade = CascadeType.ALL)
  private ContactDetail contactDetail;

  ....
}
```



e that the relationship between Company and CompanyDetail is uni-directional. On the other hand, the relationship between Contact tact Detail is bi-directional and that can be achieved using 'mappedBy' attribute.

nale to have one relationship as uni-directional and other as bi-directional in this tutorial is to illustrate both concepts and their usage. You can opt for uni-directional or bi-directional relationships to suit your needs.

### @ManyToOne


- Use @JoinColumn when foreign key is held by one of the entities.
- Use @JoinTable for entities linked through an association table.

The two examples below illustrate many-to-one relationships. Contact to Company and Company to CompanyStatus. Many contacts can belong to a company. Similary many companies can share the same status (Lead, Prospect, Customer) - there will be many companies that are currently leads.

```
@Entity
@Table(name = "contact")
public class Contact implements Serializable {

  @ManyToOne
```

```
  @JoinColumn(name = "companyId")
  private Company company;

  ...

 }

@Entity
@Table(name = "company")
public class Company implements Serializable {

  @ManyToOne
  @JoinColumn(name = "statusId")
  private CompanyStatus status;

  ...

 }
```

### @OneToMany

- Use mappedBy attribute for bi-directional associations with ManyToOne being the owner.
- OneToMany being the owner or unidirectional with foreign key - try to avoid such associations but can be achieved with @JoinColumn
- @JoinTable for Unidirectional with association table

Please see the many-to-one relationship between Contact and Company above. Company to Contact will be a one-to-many relationship. The owner of this relationship is Contact and hence we will use 'mappedBy' attribute in Company to make it bi-directional relationship.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

  @OneToMany(mappedBy = "company", fetch = FetchType.EAGER)
  @OrderBy("firstName asc")
  private Set contacts;

  ...

 }
```

r this tutorial, we have kept Company to CompanyStatus relationship as uni-directional.

### yToMany

- Use @JoinTable for entities linked through an association table.
- Use mappedBy attribute for bi-directional association.

### aryKeyJoinColumn

yKeyJoinColumn annotation is used for associated entities sharing the same primary key. See OneToOne section for details.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

  @Id
  @Column(name = "id")
  @GeneratedValue
  private int id;

  @OneToOne(cascade = CascadeType.MERGE)
  @PrimaryKeyJoinColumn
  private CompanyDetail companyDetail;

  ...
}
```

### @JoinColumn

Use @JoinColumn annotation for one-to-one or many-to-one associations when foreign key is held by one of the entities. We can use @OneToOne or @ManyToOne mappedBy attribute for bi-directional relations. Also see OneToOne and ManyToOne sections for more details.

```
@ManyToOne
@JoinColumn(name = "statusId")
private CompanyStatus status;
```

### @JoinTable

Use @JoinTable and mappedBy for entities linked through an association table.

### @MapsId

Persist two entities with shared key (when one entity holds a foreign key to the other) using @MapsId annotation. See OneToOne section for details.

```
@OneToOne
@MapsId
@JoinColumn(name = "contactId")
private Contact contact;
```

## Hibernate Inheritance Mapping Annotations

To understand Inheritance Mapping annotations, you must first understand Inheritance Mapping in Hiberate in detail. Once you understand Inheritance mapping concepts, please review below for annotations to be used.

- table per class hierarchy - single table per Class Hierarchy Strategy: the <subclass> element in Hibernate

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="planetype", discriminatorType=DiscriminatorType.STRING )

@DiscriminatorValue("Plane")
public class Plane { ... }

@Entity
@DiscriminatorValue("A320")
public class A320 extends Plane { ... }
```

- table per class/subclass - joined subclass Strategy: the <joined-subclass> element in Hibernate

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Boat implements Serializable { ... }

@Entity
@PrimaryKeyJoinColumn
public class Ferry extends Boat { ... }
```

- le per concrete class - table per Class Strategy: the <union-class> element in Hibernate

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Flight implements Serializable { ... }
```


Note: This strategy does not support the IDENTITY generator strategy: the id has to be shared across several tables. Consequently, when using this strategy, you should not use AUTO nor IDENTITY.

### @Inheritance

See Hibernate Inheritance Mapping Annotations section for details.

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
```

### @DiscriminatorColumn

See Hibernate Inheritance Mapping Annotations section for details.

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="planetype", discriminatorType=DiscriminatorType.STRING )
```

### @DiscriminatorValue

See Hibernate Inheritance Mapping Annotations section for details.

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="planetype", discriminatorType=DiscriminatorType.STRING )

@DiscriminatorValue("Plane")
public class Plane { ... }

@Entity
@DiscriminatorValue("A320")
public class A320 extends Plane { ... }
```

### References:

1. Hibernate Annotations: http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html_single/
2. Inheritance Mapping Reference: http://docs.jboss.org/hibernate/core/3.5/reference/en/html/inheritance.html