Hibernate - JPA Annotations

Baseado em: http://www.techferry.com/articles/hibernate-jpa-annotations.html

Annotation	Pacote/Import
@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;
@OrderBy	import javax.persistence.OrderBy;
@Transient	import javax.persistence.Transient;
@Lob	import javax.persistence.Lob;
Annotations para Mapeamento de Associações	
@OneToOne	import javax.persistence.OneToOne;
@ManyToOne	import javax.persistence.ManyToOne;
@OneToMany	import javax.persistence.OneToMany;
@ManyToMany	import javax.persistence.ManyToMany;
@PrimaryKeyJoinColumn	import javax.persistence.PrimaryKeyJoinColumn;
@JoinColumn	import javax.persistence.JoinColumn;
@JoinTable	import javax.persistence.JoinTable;
@MapsId	import javax.persistence.MapsId;
Annotations para Mapeamento de Herança	
@Inheritance	import javax.persistence.Inheritance;
@DiscriminatorColumn	import javax.persistence.DiscriminatorColumn;
@DiscriminatorValue	import javax.persistence.DiscriminatorValue;

@Entity

Indica a entidade a ser persistida.

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

@Table

Indica a tabela na qual a entidade será persistida. No exemplo abaixo, os dados serão armazenados na tabela 'company'. Por *default*, o Hibernate usa o nome da própria classe como nome da tabela.

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

@Column

Especifica a coluna (o nome da coluna) na qual o atributo será persistido. Por *default*, o Hibernate usa o nome do próprio atributo como nome da coluna.

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

     @Column(name = "name")
     private String name;
...
}
```

@Id

Indica qual é o atributo de identificação única do objeto a ser persistido (“id”).

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

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

@GeneratedValue

Ao usar esta anotação, o SGBD fica responsável por gerar o *id* do objeto de maneira incremental.

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

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

```
}
```

@OrderBy

Usado para ordenar os objetos recuperados. No exemplo abaixo, os contatos serão ordenados pelo atributo “firstname” e por ordem crescente, ou seja, alfabética.

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

@Transient

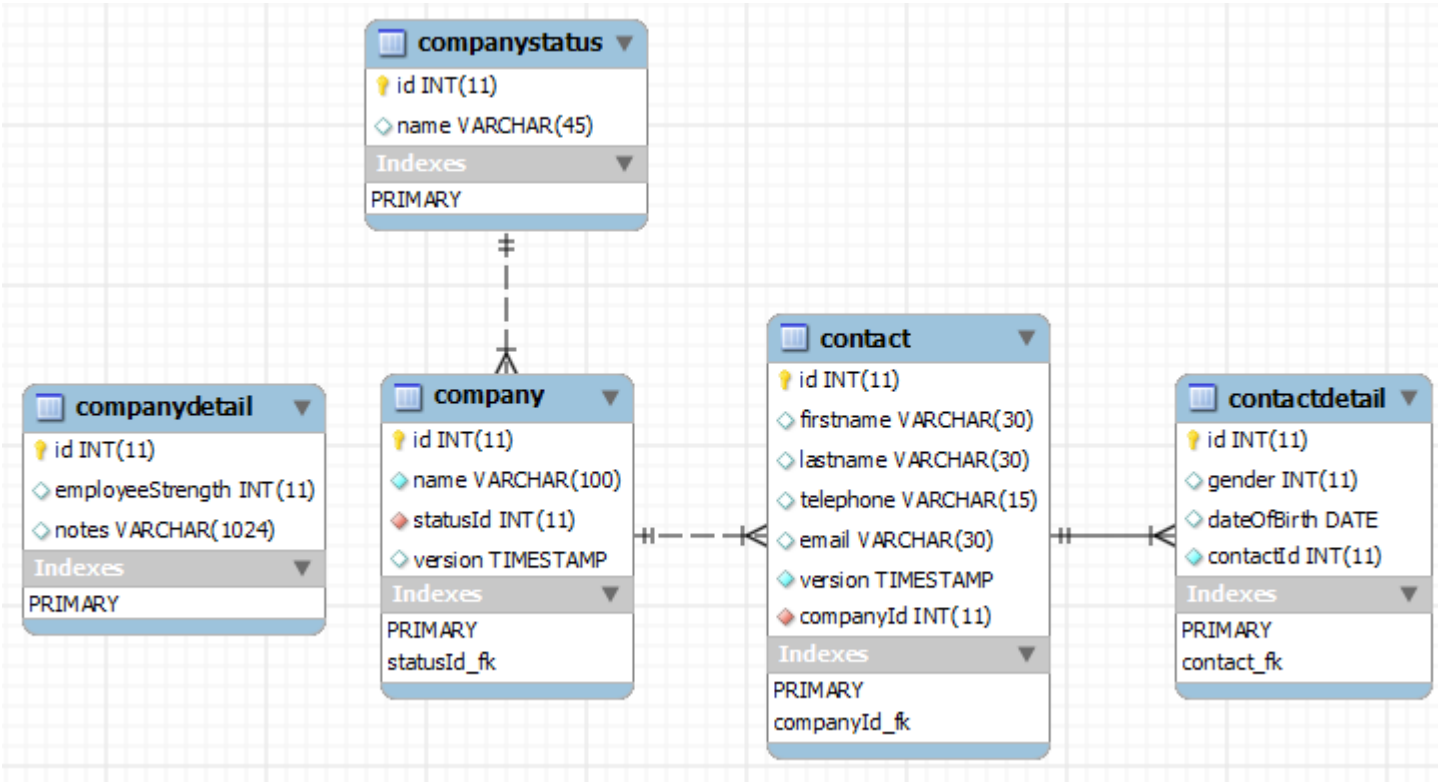
Usando para anotar atributos que não são persistentes (aqueles que não se deseja armazenar).

@Lob

Usado para indicar os atributos que serão persistidos como BLOB no banco de dados (por exemplo, figuras, textos etc).

Annotations para Mapeamento de Associações

Exemplo:



Em bancos de dados relacionais, as entidades são, basicamente, relacionadas por meio das seguintes maneiras:

- Chave primária compartilhada

- Chave estrangeira (*foreign key*)
- Tabela Associativa

De acordo com o exemplo:

- As tabelas *company* and *companyDetail* tem chave primária compartilhada. É uma associação “one-to-one”.
- As tabelas *contact* and *contactDetail* são ligadas por meio de chave estrangeira. Também é uma associação “one-to-one”.
- As tabelas *contact* and *company* são ligadas por meio de chave estrangeira em uma associação “many-to-one”, com *contact* sendo o “dono” da associação (isto é, a entidade que receberá a chave estrangeira).
- As tabelas *company* and *companyStatus* são ligadas por meio de chave estrangeira em uma associação “many-to-one”, com *company* sendo o “dono” da associação.

@OneToOne

- Use @PrimaryKeyJoinColumn para entidades que compartilham a mesma chave primária.
- Use @JoinColumn E @OneToOne com o atributo “mappedBy” quando a chave estrangeira está em uma das entidades.
- Persista duas entidades com chave compartilhada usando @MapsId.

Para as entidades *company* and *companyDetail*, que compartilham a mesma chave primária, nós podemos associá-las usando @OneToOne e @PrimaryKeyJoinColumn, conforme exemplo abaixo. Note que o atributo *id* de *companyDetail* não é anotado com @GeneratedValue. Tal atributo receberá como valor o *id* da entidade *company*.

```
@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;
...
}

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

	@Id
```

```
    @Column(name = "id")
    private int id;
...
}
```

Para as entidades *contact* e *contactDetail* ligadas por uma chave estrangeira, nós podemos usar as anotações @OneToOne e @JoinColumn. No exemplo abaixo, o atributo *id* é gerado para a entidade *contact*, será mapeado na coluna *contact_id* da tabela *contactDetail*.

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

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

    @OneToOne
    @MapsId
    @JoinColumn(name = "id")
    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;
...
}
```

A entidade *contactDetail* foi anotada com @MapsId. Essa anotação indica que será utilizado o *id* da entidade *contact* como *id* da entidade *contactDetail* e que esse *id* desse ser refletido no atributo definido com @Id.

Note que o relacionamento entre *company* e *companyDetail* é unidirecional. Já o relacionamento entre *contact* e *contactDetail* é bidirecional e pode ser obtido por meio do atributo *mappedBy*. Você pode optar por relacionamentos unidirecionais ou bidirecionais de acordo com suas necessidades.

@ManyToOne

- Use @JoinColumn quando a chave estranheira está em uma das entidades.

Os dois exemplos abaixo ilustram relacionamentos “many-to-one”: *contact* / *company* e company / *companyStatus*.

```
@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” para associações bidirecionais com “ManyToOne” sendo o “dono” do relacionamento.

Note o relacionamento entre *contact* and *company* acima apresentado. De *company* para *contact* será um relacionamento “one-to-many”. O “dono” do relacionamento é *contact* e, portanto, nós usaremos 'mappedBy' em *company* para torná-lo bi-direcional.

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

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

Nesse exemplo, mantivemos o relacionamento entre *company* e *companyStatus* como unidirectional.

@ManyToMany

- Use @JoinTable para entidades ligadas por meio de uma tabela associativa. Se não for usada, o Hibernate cria a tabela associativa e, por default, usa os nomes das duas classes associadas (<nomeclasse1>_<nomeclasse2>) para nomear tal tabela.
- Use o atributo “mappedBy” para a definição de associações bidirecionais.

```
@Entity
public class Titulo {
     @Id
     @GeneratedValue
     private long id;
     @Column(nullable = false)
     private String nome;
     private String ano;
     private String sinopse;
     private String categoria;
     @ManyToMany(cascade = CascadeType.ALL)
     @JoinTable(name="Titulo_Ator",joinColumns=
{@JoinColumn(name="titulo_id",referencedColumnName="id")},
inverseJoinColumns={@JoinColumn(name="ator_id",referencedColumnNam
e="id")})
     private Set<Ator> atores = new HashSet<Ator>();
     @ManyToOne
     private Diretor diretor;

     public Titulo(){

     }
…

@Entity
public class Ator {
     @Id
    @GeneratedValue
     private long id;
     @Column(nullable = false)
     private String nome;
     @ManyToMany(cascade = CascadeType.ALL, mappedBy="atores")
     private Set<Titulo> titulos = new HashSet<Titulo>();

     public Ator(){

     }
...
```

@PrimaryKeyJoinColumn

Esta anotação é usada para entidades associadas que compartilham a mesma chave primária. Veja a seção que trata de relacionamentos “One-to-One”.

```
@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 esta anotação para associações “one-to-one” or “many-to-one”, quando a chave estrangeira está em uma das entidades associadas. Nós podemos usar “@OneToOne” ou “@ManyToOne” com “mappedBy” para relacionamentos bidirecionais. Veja as seções anteriores que tratam em detalhes das anotações “OneToOne” e “ManyToOne” para mais detalhes.

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

@JoinTable

Use esta anotação com o atributo “mappedBy” para entidades ligadas por uma tabela associativa.

@MapsId

Esta anotação persiste duas entidades que possuem chave compartilhada e uma dessas entidades mantém a chave estrangeira para a outra entidade. Veja a seção que trata de relacionamento “OneToOne”.

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

Annotations para Mapeamento de Herança

Uma tabela para toda a hierarquia.

```
@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 { ... }
```

Uma tabela por classe.

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

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

Uma tabela por classe concreta.

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

Nota: Essa estratégia não suporta gerador de identidade, isto é, o *id* tem que ser compartilhado entre as várias tabelas.

@Inheritance

Indica a estratégia de mapeamento de herança a ser adotada.

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

@DiscriminatorColumn

Indica qual será a coluna que identificará o tipo do objeto persistido. O atributo “name” indica o nome da coluna e o atributo “discriminatorType” indica o tipo do atributo no banco de dados.

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

@DiscriminatorValue

Especifica o valor do atributo para identificar o tipo do objeto. No caso do exemplo, para a classe Plane, o valor do atributo será “Plane”.

```
@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 { ... }
```

Referências:

1. http://www.techferry.com/articles/hibernate-jpa-annotations.html
2. Hibernate Annotations:
   http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html_single/
3. Inheritance Mapping Reference:
   http://docs.jboss.org/hibernate/core/3.5/reference/en/html/inheritance.html