# Table of Contents

## JPA Class-Level Annotations

- @Entity ................................................................. 3
- @MappedSuperclass .............................................. 3
- @Embeddable ........................................................ 3
- @Id ................................................................. 4
- @Cacheable .......................................................... 4
- @EntityListeners .................................................. 4
- @NamedQuery .......................................................... 5
- @NamedNativeQuery ............................................... 5
- @NamedQuery ........................................................ 6
- @TableGenerator .................................................... 7
- @Table ............................................................... 9
- @SecondaryTable ................................................... 9
- @DiscriminatorColumn ............................................ 10
- @DiscriminatorValue ............................................... 10
- @PrimaryKeyJoinColumn ............................................ 11
- @AttributeOverride ............................................... 12
- @AttributeOverride ............................................... 12
- @AssociationOverride ............................................ 13
- @SequenceGenerator ............................................... 14
- @TableGenerator .................................................... 14

## JPA Field-Level Annotations

- @PrePersist .......................................................... 17
- @PostPersist ........................................................ 17
- @PreRemove .......................................................... 17
- @PostRemove .......................................................... 18
- @PreUpdate .......................................................... 18
- @PostUpdate .......................................................... 19
- @PostLoad ............................................................ 19
- @Id .................................................................. 20
- @Embedded ............................................................ 20
- @EmbeddedId .......................................................... 20
- @Version .............................................................. 21
- @Basic ................................................................. 21
- @Transient ............................................................ 22
JPA provides the ability to use annotations to define the persistence of entities, and DataNucleus JPA supports both JPA and JDO annotations. In this section we provide a reference to the primary JPA annotations. When selecting to use annotations please bear in mind the following:-

- You must have the `datanucleus-api-jpa` jar available in your CLASSPATH.
- You must have the `javax.persistence` jar in your CLASSPATH since this provides the annotations.
- Annotations should really only be used for attributes of persistence that you won't be changing at deployment. Things such as table and column names shouldn't really be specified using annotations although it is permitted. Instead it would be better to put such information in an ORM file.
- Annotations can be added in two places - for the class as a whole, or for a field in particular.
- You can annotate fields or getters with field-level information. It doesn't matter which.
- Annotations are prefixed by the `@` symbol and can take attributes (in brackets after the name, comma-separated).
- JPA doesn't provide for some key JDO concepts and DataNucleus provides its own annotations for these cases.
- You have to import `javax.persistence.XXX` where XXX is the annotation name of a JPA annotation.
- You have to import `org.datanucleus.api.jpa.annotations.XXX` where XXX is the annotation name of a DataNucleus value-added annotation.

Annotations supported by DataNucleus are shown below. Not all have their documentation written yet.
## JPA Class-Level Annotations

The following annotations are specified at class-level and are JPA standard. Using these provide portability for your application.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Class/Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Entity</td>
<td>Class</td>
<td>Specifies that the class is persistent</td>
</tr>
<tr>
<td>@MappedSuperclass</td>
<td>Class</td>
<td>Specifies that this class contains persistent information to be mapped</td>
</tr>
<tr>
<td>@Embeddable</td>
<td>Class</td>
<td>Specifies that the class is persistent embedded in another persistent class</td>
</tr>
<tr>
<td>@IdClass</td>
<td>Class</td>
<td>Defines the primary key class for this class</td>
</tr>
<tr>
<td>@Cacheable</td>
<td>Class</td>
<td>Specifies that instances of this class can be cached in the L2 cache</td>
</tr>
<tr>
<td>@EntityListeners</td>
<td>Class</td>
<td>Specifies class(es) that are listeners for events from instances of this class</td>
</tr>
<tr>
<td>@NamedQuery</td>
<td>Class</td>
<td>Defines a named JPQL query for use in the current persistence unit</td>
</tr>
<tr>
<td>@NamedNativeQuery</td>
<td>Class</td>
<td>Defines a named SQL query for use in the current persistence unit</td>
</tr>
<tr>
<td>@Named Stored Procedure Query</td>
<td>Class</td>
<td>Defines a named stored procedure query for use in the current persistence unit</td>
</tr>
<tr>
<td>@SqlResultSetMapping</td>
<td>Class</td>
<td>Defines a result mapping for an SQL query for use in the current persistence unit</td>
</tr>
<tr>
<td>@NamedEntityGraph</td>
<td>Class</td>
<td>Defines a named entity graph with root of the class it is specified on</td>
</tr>
<tr>
<td>@Converter</td>
<td>Class</td>
<td>Defines a java type converter for a field type</td>
</tr>
<tr>
<td>@Inheritance</td>
<td>Class</td>
<td>Specifies the inheritance model for persisting this class</td>
</tr>
<tr>
<td>@Table</td>
<td>Class</td>
<td>Defines the table where this class will be stored</td>
</tr>
<tr>
<td>@SecondaryTable</td>
<td>Class</td>
<td>Defines a secondary table where some fields of this class will be stored</td>
</tr>
<tr>
<td>@DiscriminatorColumn</td>
<td>Class</td>
<td>Defines the column where any discriminator will be stored</td>
</tr>
<tr>
<td>@DiscriminatorValue</td>
<td>Class</td>
<td>Defines the value to be used in the discriminator for objects of this class</td>
</tr>
<tr>
<td>@PrimaryKeyJoinColumn</td>
<td>Class</td>
<td>Defines the name of the PK column when this class has a superclass</td>
</tr>
<tr>
<td>@AttributeOverride</td>
<td>Class</td>
<td>Defines a field in a superclass that will have its column overridden</td>
</tr>
<tr>
<td>Annotation</td>
<td>Class/Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@AssociationOverride</td>
<td>Class</td>
<td>Defines a N-1/1-1 field in a superclass that will have its column overridden</td>
</tr>
<tr>
<td>@SequenceGenerator</td>
<td>Class/Field/Method</td>
<td>Defines a generator of values using sequences in the datastore for use with persistent entities</td>
</tr>
<tr>
<td>@TableGenerator</td>
<td>Class/Field/Method</td>
<td>Defines a generator of sequences using a table in the datastore for use with persistent entities</td>
</tr>
</tbody>
</table>

@**Entity**

This annotation is used when you want to mark a class as persistent. Specified on the `class`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the entity (used in JPQL to refer to the class)</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class MyClass {
    ...
}
```

See the documentation for [Class Mapping](#)

@**MappedSuperclass**

This annotation is used when you want to mark a class as persistent but without a table of its own and being the superclass of the class that has a table, meaning that all of its fields are persisted into the table of its subclass. Specified on the `class`.

```java
@MappedSuperclass
public class MyClass {
    ...
}
```

See the documentation for [Inheritance](#)

@**Embeddable**

This annotation is used when you want to mark a class as persistent and only storable embedded in another object. Specified on the `class`.

---

3
@Embeddable
public class MyClass
{
    ...
}

@IdClass
This annotation is used to define a primary-key class for the identity of this class. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Class</td>
<td>Identity class</td>
<td></td>
</tr>
</tbody>
</table>

@Entity
@IdClass(mydomain.samples.MyIdentity.class)
public class MyClass
{
    ...
}

See the documentation for Primary Keys

@Cacheable
This annotation is used when you want to mark a class so that instance of that class can be cached. Specified on the class.

@Cacheable
public class MyClass
{
    ...
}

See the documentation for L2 Cache

@EntityListeners
This annotation is used to define a class or classes that are listeners for events from instances of this class. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Class[]</td>
<td>Entity listener class(es)</td>
<td></td>
</tr>
</tbody>
</table>
@Entity
@EntityListeners(mydomain.MyListener.class)

public class MyClass
{
  ...
}

See the documentation for Lifecycle Callbacks

@NamedQuery

This annotation is used to define a named (JPQL) query that can be used in this persistence unit. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Symbolic name for the query. The query will be referred to under this name</td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>String</td>
<td>The JPQL query</td>
<td></td>
</tr>
</tbody>
</table>

@NamedQuery(name="AllPeople", query="SELECT p FROM Person p")

public class Person
{
  ...
}

Note that with DataNucleus you can also specify @NamedQuery on non-persistable classes See the documentation for Named Queries

> There is a @NamedQueries annotation but you can achieve the same cleaner using multiple @NamedQuery annotations.

@NamedNativeQuery

This annotation is used to define a named (SQL) query that can be used in this persistence unit. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Symbolic name for the query. The query will be referred to under this name</td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>String</td>
<td>The SQL query</td>
<td></td>
</tr>
<tr>
<td>resultClass</td>
<td>Class</td>
<td>Class into which the result rows will be placed</td>
<td>void.class</td>
</tr>
</tbody>
</table>
@Entity
@NamedNativeQuery(name="PeopleCalledSmith", query="SELECT * FROM PERSON WHERE SURNAME = 'Smith'")

public class Person
{
    ...
}

Note that with DataNucleus you can also specify @NamedNativeQuery on non-persistable classes
See the documentation for Named Native Queries

There is a @NamedNativeQueries annotation but you can achieve the same cleaner using multiple @NamedNativeQuery annotations.

@NamedStoredProcedureQuery
This annotation is used to define a named stored procedure query that can be used in this persistence unit. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Symbolic name for the query. The query will be referred to under this name</td>
<td></td>
</tr>
<tr>
<td>procedureName</td>
<td>String</td>
<td>Name of the stored procedure in the datastore</td>
<td></td>
</tr>
<tr>
<td>parameters</td>
<td>StoredProcedureParameter[]</td>
<td>Any parameter definitions for this stored procedure</td>
<td></td>
</tr>
<tr>
<td>resultClasses</td>
<td>Class[]</td>
<td>Any result class(es) for this stored procedure (one per result set)</td>
<td></td>
</tr>
<tr>
<td>resultSetMappings</td>
<td>Class[]</td>
<td>Any result set mapping(s) for this stored procedure (one per result set)</td>
<td></td>
</tr>
<tr>
<td>hints</td>
<td>QueryHint[]</td>
<td>Any query hints for this stored procedure</td>
<td></td>
</tr>
</tbody>
</table>

@Entity
@NamedStoredProcedureQuery(name="MyProc", procedureName="MY_PROC_SP1",
parameters={@StoredProcedureParameter(name="PARAM1", mode=ParameterMode.IN,
type=String.class)})

public class Person
{
    ...
}

Note that with DataNucleus you can also specify @NamedStoredProcedureQuery on non-persistable classes
See the documentation for Named StoredProcedures
There is a `@NamedStoredProcedureQueries` annotation but you can achieve the same cleaner using multiple `@NamedStoredProcedureQuery` annotations.

### @SqlResultSetMapping

This annotation is used to define a mapping for the results of an SQL query and can be used in this persistence unit. Specified on the `class`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Symbolic name for the mapping. The mapping will be referenced under this name</td>
<td></td>
</tr>
<tr>
<td>entities</td>
<td>EntityResult[]</td>
<td>Set of entities extracted from the SQL query</td>
<td></td>
</tr>
<tr>
<td>columns</td>
<td>ColumnResult[]</td>
<td>Set of columns extracted directly from the SQL query</td>
<td></td>
</tr>
</tbody>
</table>

    @Entity
    @SqlResultSetMapping(name="PEOPLE_PLUS_AGE",
                           entities={@EntityResult(entityClass=Person.class)},
                           columns={@ColumnResult(name = "AGE")})

    public class Person
    {
        ... 
    }

There is a `@SqlResultSetMappings` annotation but you can achieve the same cleaner using multiple `@SqlResultSetMapping` annotations.

### @NamedEntityGraph

This annotation is used to define a named EntityGraph and can be used in this persistence unit. Specified on the `class`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>name for the Entity Graph.</td>
<td></td>
</tr>
<tr>
<td>attributeNodes</td>
<td>AttributeNode[]</td>
<td>Set of nodes in this EntityGraph</td>
<td></td>
</tr>
</tbody>
</table>
@Entity
@NamedEntityGraph(name="PERSON_FULL",
    attributeNodes={@NamedAttributeNode(name="friends"),
    @NamedAttributeNode(name="parents"))})
public class Person
{
    ...
}

There is a @NamedEntityGraphs annotation but you can achieve the same cleaner using multiple @NamedEntityGraph annotations.

@Converter

This annotation is used to mark a class as being an attribute converter. Note that DataNucleus doesn't require this specifying against a converter class except if you want to set the "autoApply". Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoApply</td>
<td>boolean</td>
<td>Whether this converter should always be used when storing this java type</td>
<td>false</td>
</tr>
</tbody>
</table>

@Converter

public class MyConverter
{
    ...
}

@Inheritance

This annotation is used to define the inheritance persistence for this class. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy</td>
<td>InheritanceType</td>
<td>Inheritance strategy</td>
<td>SINGLE_TABLE, JOINED, TABLE_PER_CLASS</td>
</tr>
</tbody>
</table>
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class MyClass
{
  ...
}

See the documentation for Inheritance

@Table
This annotation is used to define the table where objects of a class will be stored. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the table</td>
<td></td>
</tr>
<tr>
<td>catalog</td>
<td>String</td>
<td>Name of the catalog</td>
<td></td>
</tr>
<tr>
<td>schema</td>
<td>String</td>
<td>Name of the schema</td>
<td></td>
</tr>
<tr>
<td>uniqueConstraints</td>
<td>UniqueConstraints[]</td>
<td>Any unique constraints to apply to the table</td>
<td></td>
</tr>
<tr>
<td>indexes</td>
<td>Index[]</td>
<td>Details of indexes if wanting to override provider default</td>
<td></td>
</tr>
</tbody>
</table>

@Entity
@Table(name="MYTABLE", schema="PUBLIC")
public class MyClass
{
  ...
}

@SecondaryTable
This annotation is used to define a secondary table where some fields of this class are stored in another table. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the table</td>
<td></td>
</tr>
<tr>
<td>catalog</td>
<td>String</td>
<td>Name of the catalog</td>
<td></td>
</tr>
<tr>
<td>schema</td>
<td>String</td>
<td>Name of the schema</td>
<td></td>
</tr>
<tr>
<td>pkJoinColumnNames</td>
<td>PrimaryKeyJoinColumnNames[]</td>
<td>Join columns for the PK of the secondary table back to the primary table</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>uniqueConstraints</td>
<td>UniqueConstraints[]</td>
<td>Any unique constraints to apply to the table</td>
<td></td>
</tr>
<tr>
<td>indexes</td>
<td>Index[]</td>
<td>Details of indexes if wanting to override provider default</td>
<td></td>
</tr>
<tr>
<td>foreignKey</td>
<td>ForeignKey</td>
<td>Foreign key details if wanting to override provider default</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@Table(name="MYTABLE", schema="PUBLIC")
@SecondaryTable(name="MYOTHERTABLE", schema="PUBLIC", columns={@PrimaryKeyJoinColumn (name="MYCLASS_ID")})

public class MyClass
{
  ...
}
```

See the documentation for Secondary Tables

**@DiscriminatorColumn**

This annotation is used to define the discriminator column for a class. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the discriminator column</td>
<td>DTYPE</td>
</tr>
<tr>
<td>discriminatorType</td>
<td>DiscriminatorType</td>
<td>Type of the discriminator column</td>
<td>STRING, CHAR, INTEGER</td>
</tr>
<tr>
<td>length</td>
<td>String</td>
<td>Length of the discriminator column</td>
<td>31</td>
</tr>
</tbody>
</table>

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

public class MyClass
{
  ...
}
```

See the documentation for Inheritance

**@DiscriminatorValue**

This annotation is used to define the value to be stored in the discriminator column for a class (when used). Specified on the class.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>Value for the discriminator column</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="OBJECT_TYPE", discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("MyClass")
public class MyClass
{
    ...
}
```

See the documentation for `Inheritance`

### @PrimaryKeyJoinColumn

This annotation is used to define the name of the primary key column when this class has a superclass. Specified on the `class`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the column</td>
<td></td>
</tr>
<tr>
<td>referencedColumn</td>
<td>String</td>
<td>Name of the associated PK column in the superclass. This is for use when you have a composite PK so acts as a way of aligning the respective columns. <strong>It is not to allow joining to some non-PK column</strong></td>
<td></td>
</tr>
<tr>
<td>columnDefinition</td>
<td>String</td>
<td>DDL to use for the column (everything except the column name). This must include the SQL type of the column</td>
<td></td>
</tr>
<tr>
<td>foreignKey</td>
<td>ForeignKey</td>
<td>Foreign key details if wanting to override provider default</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
@PrimaryKeyJoinColumn(name="PK_FIELD_1")
public class MyClass
{
    ...
}
```

There is a `@PrimaryKeyJoinColumns` annotation but you can achieve the same more cleanly with multiple `@PrimaryKeyJoinColumn` annotations.
@AttributeOverride

This annotation is used to define a field of a superclass that has its column overridden. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the field</td>
<td></td>
</tr>
<tr>
<td>column</td>
<td>Column</td>
<td>Column information</td>
<td></td>
</tr>
</tbody>
</table>

@AttributeOverride(name="attr", column=@Column(name="NEW_NAME"))

public class MyClass extends MySuperClass
{
    ...
}

There is also an @AttributeOverrides annotation but you can achieve the same cleaner using multiple @AttributeOverride annotations.

@AttributeOverride

This annotation is used to define a field of an embedded class that has its column overridden. Specified on the field/property.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the field</td>
<td></td>
</tr>
<tr>
<td>column</td>
<td>Column</td>
<td>Column information</td>
<td></td>
</tr>
</tbody>
</table>

@Entity
public class MyClass extends MySuperClass
{
    @Embedded
    @AttributeOverride(name="attr", column=@Column(name="NEW_NAME"))
    MyEmbeddedType embedded;
    ...
}

There is also an @AttributeOverrides annotation but you can achieve the same cleaner using multiple @AttributeOverride annotations.

@AssociationOverride

This annotation is used to define a 1-1/N-1 field of a superclass that has its column overridden.

@Entity
public class MyClass extends MySuperClass
{
    ...
}

There is also an @AssociationOverrides annotation but you can achieve the same cleaner using multiple @AssociationOverride annotations.
Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the field</td>
<td></td>
</tr>
<tr>
<td>joinColumn</td>
<td>JoinColumn</td>
<td>Column information for the FK column</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@AssociationOverride(name="friend", joinColumn=@JoinColumn(name="FRIEND_ID"))
public class Employee extends Person
{
    ...
}
```

There is also an `@AssociationOverrides` annotation but you can achieve the same cleaner using multiple `@AssociationOverride` annotations.

### @SequenceGenerator

This annotation is used to define a generator using sequences in the datastore. It is scoped to the persistence unit. Specified on the class/field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name for the generator (required)</td>
<td></td>
</tr>
<tr>
<td>sequenceName</td>
<td>String</td>
<td>Name of the underlying sequence that will be used</td>
<td></td>
</tr>
<tr>
<td>initialValue</td>
<td>int</td>
<td>Initial value for the sequence (optional)</td>
<td>1</td>
</tr>
<tr>
<td>allocationSize</td>
<td>int</td>
<td>Number of values to be allocated each time (optional)</td>
<td>50</td>
</tr>
<tr>
<td>schema</td>
<td>String</td>
<td>Name of the schema where the sequence will be stored (optional)</td>
<td></td>
</tr>
<tr>
<td>catalog</td>
<td>String</td>
<td>Name of the catalog where the sequence will be stored (optional)</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@SequenceGenerator(name="MySeq", sequenceName="SEQ_2")
public class MyClass
{
    ...
}
```
@TableGenerator

This annotation is used to define a generator using a table in the datastore for storing the values. It is scoped to the persistence unit. Specified on the class/field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name for the generator (required)</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>String</td>
<td>Name of the table to use</td>
<td>SEQUENCE_TABLE</td>
</tr>
<tr>
<td>catalog</td>
<td>String</td>
<td>Catalog of the table to use (optional)</td>
<td></td>
</tr>
<tr>
<td>schema</td>
<td>String</td>
<td>Schema of the table to use (optional)</td>
<td></td>
</tr>
<tr>
<td>pkColumnName</td>
<td>String</td>
<td>Name of the primary key column for the table</td>
<td>SEQUENCE_NAME</td>
</tr>
<tr>
<td>valueColumnName</td>
<td>String</td>
<td>Name of the value column for the table</td>
<td>NEXT_VAL</td>
</tr>
<tr>
<td>pkColumnValue</td>
<td>String</td>
<td>Value to store in the PK column for the row used by this generator</td>
<td>{name of the class}</td>
</tr>
<tr>
<td>initialValue</td>
<td>int</td>
<td>Initial value for the table row (optional)</td>
<td>0</td>
</tr>
<tr>
<td>allocationSize</td>
<td>int</td>
<td>Number of values to be allocated each time (optional)</td>
<td>50</td>
</tr>
<tr>
<td>indexes</td>
<td>Index[]</td>
<td>Index(es) if wanting to override the provider default</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@TableGenerator(name="MySeq", table="MYAPP_IDENTITIES", pkColumnValue="MyClass")
public class MyClass
{
    ...
}
```
# JPA Field-Level Annotations

The following annotations are specified at field/method-level and are JPA standard. Using these provide portability for your application.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Class/Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@SequenceGenerator</code></td>
<td>Class/Field/Method</td>
<td>Defines a generator of values using sequences in the datastore for use with persistent entities</td>
</tr>
<tr>
<td><code>@TableGenerator</code></td>
<td>Class/Field/Method</td>
<td>Defines a generator of sequences using a table in the datastore for use with persistent entities</td>
</tr>
<tr>
<td><code>@Embedded</code></td>
<td>Field/Method</td>
<td>Defines this field as being embedded</td>
</tr>
<tr>
<td><code>@AttributeOverride</code></td>
<td>Class</td>
<td>Defines a field in an embedded class that will have its column overridden</td>
</tr>
<tr>
<td><code>@Id</code></td>
<td>Field/Method</td>
<td>Defines this field as being (part of) the identity for the class</td>
</tr>
<tr>
<td><code>@EmbeddedId</code></td>
<td>Field/Method</td>
<td>Defines this field as being (part of) the identity for the class, and being embedded into this class.</td>
</tr>
<tr>
<td><code>@Version</code></td>
<td>Field/Method</td>
<td>Defines this field as storing the version for the class</td>
</tr>
<tr>
<td><code>@Basic</code></td>
<td>Field/Method</td>
<td>Defines this field as being persistent</td>
</tr>
<tr>
<td><code>@Transient</code></td>
<td>Field/Method</td>
<td>Defines this field as being transient (not persisted)</td>
</tr>
<tr>
<td><code>@OneToMany</code></td>
<td>Field/Method</td>
<td>Defines this field as being a 1-N relation with other persistent entities</td>
</tr>
<tr>
<td><code>@ManyToOne</code></td>
<td>Field/Method</td>
<td>Defines this field as being a 1-1 relation with another persistent entity</td>
</tr>
<tr>
<td><code>@ElementCollection</code></td>
<td>Field/Method</td>
<td>Defines this field as being a 1-N relation of Objects that are not Entities.</td>
</tr>
<tr>
<td><code>@GeneratedValue</code></td>
<td>Field/Method</td>
<td>Defines that this field has its value generated using a generator</td>
</tr>
<tr>
<td><code>@MapKey</code></td>
<td>Field/Method</td>
<td>Defines that this field is the key to a map</td>
</tr>
<tr>
<td><code>@MapKeyClass</code></td>
<td>Field/Method</td>
<td>Defines that the key type for the map in this field</td>
</tr>
<tr>
<td><code>@MapKeyEnumerated</code></td>
<td>Field/Method</td>
<td>Defines the datastore type for the map key when it is an enum</td>
</tr>
<tr>
<td><code>@MapKeyTemporal</code></td>
<td>Field/Method</td>
<td>Defines the datastore type for the map key when it is a temporal type</td>
</tr>
<tr>
<td>Annotation</td>
<td>Class/Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@MapKeyColumn</td>
<td>Field/Method</td>
<td>Defines the column details for the map key when stored in a join table</td>
</tr>
<tr>
<td>@OrderBy</td>
<td>Field/Method</td>
<td>Defines the field(s) used for ordering the elements in this collection</td>
</tr>
<tr>
<td>@OrderColumn</td>
<td>Field/Method</td>
<td>Defines that ordering should be attributed by the implementation using a surrogate column.</td>
</tr>
<tr>
<td>@PrePersist</td>
<td>Field/Method</td>
<td>Defines this method as being a callback for pre-persist events</td>
</tr>
<tr>
<td>@PostPersist</td>
<td>Field/Method</td>
<td>Defines this method as being a callback for post-persist events</td>
</tr>
<tr>
<td>@PreRemove</td>
<td>Field/Method</td>
<td>Defines this method as being a callback for pre-remove events</td>
</tr>
<tr>
<td>@PostRemove</td>
<td>Field/Method</td>
<td>Defines this method as being a callback for post-remove events</td>
</tr>
<tr>
<td>@PreUpdate</td>
<td>Field/Method</td>
<td>Defines this method as being a callback for pre-update events</td>
</tr>
<tr>
<td>@PostUpdate</td>
<td>Field/Method</td>
<td>Defines this method as being a callback for post-update events</td>
</tr>
<tr>
<td>@PostLoad</td>
<td>Field/Method</td>
<td>Defines this method as being a callback for post-load events</td>
</tr>
<tr>
<td>@JoinTable</td>
<td>Field/Method</td>
<td>Defines this field as being stored using a join table</td>
</tr>
<tr>
<td>@CollectionTable</td>
<td>Field/Method</td>
<td>Defines this field as being stored using a join table when containing non-entity objects.</td>
</tr>
<tr>
<td>@Lob</td>
<td>Field/Method</td>
<td>Defines this field as being stored as a large object</td>
</tr>
<tr>
<td>@Temporal</td>
<td>Field/Method</td>
<td>Defines this field as storing temporal data</td>
</tr>
<tr>
<td>@Enumerated</td>
<td>Field/Method</td>
<td>Defines this field as storing enumerated data</td>
</tr>
<tr>
<td>@Convert</td>
<td>Field/Method</td>
<td>Defines a converter for this field/property</td>
</tr>
<tr>
<td>@Column</td>
<td>Field/Method</td>
<td>Defines the column where this field is stored</td>
</tr>
<tr>
<td>@JoinColumn</td>
<td>Field/Method</td>
<td>Defines a column for joining to either a join table or foreign key relation</td>
</tr>
<tr>
<td>@Index</td>
<td>-</td>
<td>Defines the details of an index when overriding the provider default.</td>
</tr>
<tr>
<td>@ForeignKey</td>
<td>-</td>
<td>Defines the details of a foreign key when overriding the provider default.</td>
</tr>
<tr>
<td>@MapsId</td>
<td>Field/Method</td>
<td>Defines that this field maps one part of the id of the overall class. <strong>NOT SUPPORTED</strong>.</td>
</tr>
</tbody>
</table>
@PrePersist

This annotation is used to define a method that is a callback for pre-persist events. Specified on the method. It has no attributes.

```java
@Entity
public class MyClass {
    ...
    @PrePersist
    void registerObject() {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@PostPersist

This annotation is used to define a method that is a callback for post-persist events. Specified on the method. It has no attributes.

```java
@Entity
public class MyClass {
    ...
    @PostPersist
    void doSomething() {
        ...
    }
}
```

See the documentation for Lifecycle Callbacks

@PreRemove

This annotation is used to define a method that is a callback for pre-remove events. Specified on the method. It has no attributes.
@Entity
class MyClass
{
  ...

  @PreRemove
  void registerObject()
  {
    ...
  }
}

See the documentation for Lifecycle Callbacks

@PostRemove

This annotation is used to define a method that is a callback for post-remove events. Specified on the method. It has no attributes.

@Entity
class MyClass
{
  ...

  @PostRemove
  void doSomething()
  {
    ...
  }
}

See the documentation for Lifecycle Callbacks

@PreUpdate

This annotation is used to define a method that is a callback for pre-update events. Specified on the method. It has no attributes.
```java
@Entity
public class MyClass {

    @PreUpdate
    void registerObject() {
        ...
    }
}

See the documentation for Lifecycle Callbacks

@PostUpdate

This annotation is used to define a method that is a callback for post-update events. Specified on the method. It has no attributes.

```java
@Entity
public class MyClass {

    @PostUpdate
    void doSomething() {
        ...
    }
}

See the documentation for Lifecycle Callbacks.

@PostLoad

This annotation is used to define a method that is a callback for post-load events. Specified on the method. It has no attributes.
@Entity
class MyClass {
    ...

    @PostLoad
    void registerObject() {
        ...
    }
}

See the documentation for Lifecycle Callbacks

@Id
This annotation is used to define a field to use for the identity of the class. Specified on the field/method.

@Entity
class MyClass {
    @Id
    long id;
    ...
}

@Embedded
This annotation is used to define a field as being embedded. Specified on the field/method.

@Entity
class MyClass {
    @Embedded
    Object myField;
    ...
}

@EmbeddedId
This annotation is used to define a field to use for the identity of the class when embedded. Specified on the field/method.
```java
@Entity
public class MyClass {
    @EmbeddedId
    MyPrimaryKey pk;
    ...
}

@Version
This annotation is used to define a field as holding the version for the class. Specified on the field/method.
```
See the documentation for **Fields/Properties**

### @Transient

This annotation is used to define a field of the class as not persistent. Specified on the **field/method**.

```java
@Entity
public class Person {
    @Id
    long id;

    @Transient
    String personalInformation;
    ...
}
```

See the documentation for **Fields/Properties**

### @JoinTable

This annotation is used to define that a collection/map is stored using a join table. Specified on the **field/method**.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the table</td>
<td></td>
</tr>
<tr>
<td>catalog</td>
<td>String</td>
<td>Name of the catalog</td>
<td></td>
</tr>
<tr>
<td>schema</td>
<td>String</td>
<td>Name of the schema</td>
<td></td>
</tr>
<tr>
<td>joinColumns</td>
<td>JoinColumn[]</td>
<td>Columns back to the owning object (with the collection/map)</td>
<td></td>
</tr>
<tr>
<td>inverseJoinColumns</td>
<td>JoinColumn[]</td>
<td>Columns to the element object (stored in the collection/map)</td>
<td></td>
</tr>
<tr>
<td>uniqueConstraints</td>
<td>UniqueConstraint[]</td>
<td>Any unique constraints to apply to the table</td>
<td></td>
</tr>
<tr>
<td>indexes</td>
<td>Index[]</td>
<td>Details of indexes if wanting to override provider default</td>
<td></td>
</tr>
<tr>
<td>foreignKey</td>
<td>ForeignKey</td>
<td>Foreign key details if wanting to override provider default for the join columns</td>
<td></td>
</tr>
<tr>
<td>inverseForeignKey</td>
<td>ForeignKey</td>
<td>Foreign key details if wanting to override provider default for the inverse join columns</td>
<td></td>
</tr>
</tbody>
</table>
@Entity
public class Person
{
    @OneToMany
    @JoinTable(name = "PEOPLES_FRIENDS")
    Collection friends;
    ...
}

@CollectionTable
This annotation is used to define that a collection/map of non-entities is stored using a join table. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the table</td>
<td></td>
</tr>
<tr>
<td>catalog</td>
<td>String</td>
<td>Name of the catalog</td>
<td></td>
</tr>
<tr>
<td>schema</td>
<td>String</td>
<td>Name of the schema</td>
<td></td>
</tr>
<tr>
<td>joinColumns</td>
<td>JoinColumn[]</td>
<td>Columns back to the owning object (with the collection/map)</td>
<td></td>
</tr>
<tr>
<td>uniqueConstraints</td>
<td>UniqueConstraint[]</td>
<td>Any unique constraints to apply to the table</td>
<td></td>
</tr>
<tr>
<td>indexes</td>
<td>Index[]</td>
<td>Details of indexes if wanting to override provider default</td>
<td></td>
</tr>
<tr>
<td>foreignKey</td>
<td>ForeignKey</td>
<td>Details of foreign key if wanting to override provider default</td>
<td></td>
</tr>
</tbody>
</table>

@Entity
public class Person
{
    @ElementCollection
    @CollectionTable(name = "PEOPLES_FRIENDS")
    Collection<String> values;
    ...
}

@Lob
This annotation is used to define that a field will be stored using a large object in the datastore. Specified on the field/method.
```java
@Entity
class Person {
    @Temporal(TemporalType.TIMESTAMP)
    java.util.Date dateOfBirth;
    ...
}
```

@Temporal
This annotation is used to define that a field is stored as a temporal type. It specifies the JDBC type to use for storage of this type, so whether it stores the date, the time, or both. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>TemporalType</td>
<td>Type for storage</td>
<td>DATE, TIME, TIMESTAMP</td>
</tr>
</tbody>
</table>

```java
@Entity
class Person {
    @Enumerated
    Gender gender;
    ...
}
```

@Enumerated
This annotation is used to define that a field is stored enumerated (not that it wasn't obvious from the type!). Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>EnumType</td>
<td>Type for storage</td>
<td>ORDINAL, STRING</td>
</tr>
</tbody>
</table>
@OneToOne

This annotation is used to define that a field represents a 1-1 relation. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetEntity</td>
<td>Class</td>
<td>Class at the other side of the relation</td>
<td></td>
</tr>
<tr>
<td>fetch</td>
<td>FetchType</td>
<td>Whether the field should be fetched immediately</td>
<td>EAGER, LAZY</td>
</tr>
<tr>
<td>optional</td>
<td>boolean</td>
<td>Whether the field can store nulls.</td>
<td>true, false</td>
</tr>
<tr>
<td>mappedBy</td>
<td>String</td>
<td>Name of the field that owns the relation (specified on the inverse side). If the field that owns the relation is stored in an embedded object on the other side, use DOT notation to identify it.</td>
<td></td>
</tr>
<tr>
<td>cascade</td>
<td>CascadeType[]</td>
<td>Whether persist, update, delete, refresh operations are cascaded</td>
<td></td>
</tr>
<tr>
<td>orphanRemoval</td>
<td>boolean</td>
<td>Whether to remove orphans when either removing this side of the relation or when nulling the relation</td>
<td>true, false</td>
</tr>
</tbody>
</table>

@Entity

```java
public class Person {
    @OneToOne
    Person bestFriend;
    ...
}
```

See the documentation for 1-1 Relations

@OneToMany

This annotation is used to define that a field represents a 1-N relation. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetEntity</td>
<td>Class</td>
<td>Class at the other side of the relation</td>
<td></td>
</tr>
<tr>
<td>fetch</td>
<td>FetchType</td>
<td>Whether the field should be fetched immediately</td>
<td>EAGER, LAZY</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>mappedBy</td>
<td>String</td>
<td>Name of the field that owns the relation (specified on the inverse side). If the field that owns the relation is stored in an embedded object on the other side, use DOT notation to identify it.</td>
<td></td>
</tr>
<tr>
<td>cascade</td>
<td>CascadeType[]</td>
<td>Whether persist, update, delete, refresh operations are cascaded</td>
<td></td>
</tr>
<tr>
<td>orphanRemoval</td>
<td>boolean</td>
<td>Whether to remove orphans when either removing this side of the relation or when nulling the relation and removing an element</td>
<td>true, false</td>
</tr>
</tbody>
</table>

```java
@Entity
public class Person {
    @OneToMany
    Collection<Person> friends;
    ...
}
```

See the documentation for 1-N Relations

@ManyToOne

This annotation is used to define that a field represents a M-N relation. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetEntity</td>
<td>Class</td>
<td>Class at the other side of the relation</td>
<td></td>
</tr>
<tr>
<td>fetch</td>
<td>FetchType</td>
<td>Whether the field should be fetched immediately</td>
<td>EAGER, LAZY</td>
</tr>
<tr>
<td>mappedBy</td>
<td>String</td>
<td>Name of the field on the non-owning side that completes the relation. Specified on the owner side. If the field that owns the relation is stored in an embedded object on the other side, use DOT notation to identify it.</td>
<td></td>
</tr>
<tr>
<td>cascade</td>
<td>CascadeType[]</td>
<td>Whether persist, update, delete, refresh operations are cascaded</td>
<td></td>
</tr>
</tbody>
</table>
@Entity
public class Customer
{
    @ManyToMany(
        mappedBy="customers"
    )
    Collection<Supplier> suppliers;
    ... 
}

@Entity
public class Supplier
{
    @ManyToMany
    Collection<Customer> customers;
    ... 
}

See the documentation for M-N Relations

@ManyToMany

This annotation is used to define that a field represents a N-1 relation. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetEntity</td>
<td>Class</td>
<td>Class at the other side of the relation</td>
<td></td>
</tr>
<tr>
<td>fetch</td>
<td>FetchType</td>
<td>Whether the field should be fetched immediately</td>
<td>EAGER, LAZY</td>
</tr>
<tr>
<td>optional</td>
<td>boolean</td>
<td>Whether the field can store nulls.</td>
<td>true, false</td>
</tr>
<tr>
<td>cascade</td>
<td>CascadeType[]</td>
<td>Whether persist, update, delete, refresh operations are cascaded</td>
<td></td>
</tr>
</tbody>
</table>
@Entity
public class House
{
    @OneToMany(mappedBy="house")
    Collection<Window> windows;
    ...
}

@Entity
public class Window
{
    @ManyToOne
    House house;
    ...
}

See the documentation for [N-1 Relations](#)

**@ElementCollection**

This annotation is used to define that a field represents a 1-N relation to non-entity objects. Specified on the `field/method`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetClass</td>
<td>Class</td>
<td>Class at the other side of the relation</td>
<td></td>
</tr>
<tr>
<td>fetch</td>
<td>FetchType</td>
<td>Whether the field should be fetched immediately</td>
<td>EAGER, LAZY</td>
</tr>
</tbody>
</table>

@Entity
public class Person
{
    @ElementCollection
    Collection<String> values;
    ...
}

**@GeneratedValue**

This annotation is used to define the generation of a value for a (PK) field. Specified on the `field/method`.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy</td>
<td>GenerationType</td>
<td>Strategy to use when generating the values for this field. Has possible values of GenerationType TABLE, SEQUENCE, IDENTITY, AUTO, UUID. Note that UUID is only available if using the DN provided javax.persistence.jar (v2.2+)</td>
<td>GenerationType.AUTO</td>
</tr>
<tr>
<td>generator</td>
<td>String</td>
<td>Name of the generator to use. See @TableGenerator and @SequenceGenerator</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class Person {
    @Id
    @GeneratedValue(strategy=GenerationType.TABLE)
    long id;
    ...
}
```

@MapKey

This annotation is used to define the field in the value class that represents the key in a Map. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the field in the value class to use for the key. If no value is supplied and the field is a Map then it is assumed that the key will be the primary key of the value class. DataNucleus only supports this null value treatment if the primary key of the value has a single field.</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class Person {
    @OneToMany
    @MapKey(name="nickname")
    Map<String, Person> friends;
    ...
}
```

@MapKeyClass

This annotation is used to define the key type for a map field when generics have not been specified. Specified on the field/method.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>String</td>
<td>Class to be used for the key of the map.</td>
<td></td>
</tr>
</tbody>
</table>

@Entity
public class Person
{
    @OneToMany(targetEntity=Person.class)
    @MapKeyClass(String.class)
    Map friends;
    ...
}

Avoid use of this class and use Java generics! This is the 21st century after all

@MapKeyTemporal

This annotation is used to define the datastore type used for the key of a map when it is a temporal type. Specified on the field/method.

@Entity
public class Person
{
    @ElementCollection
    @MapKeyTemporal(TemporalType.DATE)
    Map<Date, String> dateMap;
    ...
}

@MapKeyEnumerated

This annotation is used to define the datastore type used for the key of a map when it is an enum. Specified on the field/method.

@Entity
public class Person
{
    @ElementCollection
    @MapKeyEnumerated(EnumType.STRING)
    Map<MyEnum, String> dateMap;
    ...
}
@MapKeyColumn

This annotation is used to define the column details for a key of a Map when stored in a join table. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the column for the key</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class Person {
  @OneToMany
  @MapKeyColumn(name="FRIEND_NAME")
  Map<String, Person> friends;
  ...
}
```

@OrderBy

This annotation is used to define a field in the element class that is used for ordering the elements of the List when it is retrieved. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>Name of the field(s) in the element class to use for ordering the elements of the List when retrieving them from the datastore. This is used by JPA &quot;ordered lists&quot; as opposed to &quot;indexed lists&quot; (which always return the elements in the same order as they were persisted. The value will be a comma separated list of fields and optionally have ASC/DESC to signify ascending or descending</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class Person {
  @OneToMany
  @OrderBy(value="nickname")
  List<Person> friends;
  ...
}
```
@OrderColumn

This annotation is used to define that the JPA implementation will handle the ordering of the List elements using a surrogate column ("ordered list"). Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the column to use.</td>
<td>{fieldName}_ORDER</td>
</tr>
<tr>
<td>nullable</td>
<td>boolean</td>
<td>Whether the column is nullable</td>
<td>true, false</td>
</tr>
<tr>
<td>insertable</td>
<td>boolean</td>
<td>Whether the column is insertable</td>
<td>true, false</td>
</tr>
<tr>
<td>updatable</td>
<td>boolean</td>
<td>Whether the column is updatable</td>
<td>true, false</td>
</tr>
<tr>
<td>base</td>
<td>int</td>
<td>Base for ordering (not currently supported)</td>
<td>0</td>
</tr>
</tbody>
</table>

```java
@Entity
public class Person {
    @OneToMany
    @OrderColumn
    List<Person> friends;
    ...
}
```

@Convert

This annotation is used to define a converter for the field/property. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>converter</td>
<td>Class</td>
<td>Converter class</td>
<td></td>
</tr>
<tr>
<td>attributeName</td>
<td>String</td>
<td>&quot;key&quot; if specified on a Map field and converting the key. &quot;value&quot; if specified on a Map field and converting the value. Alternatively the name of the embedded field to be converted (currently not supported).</td>
<td></td>
</tr>
<tr>
<td>disableConversion</td>
<td>boolean</td>
<td>Whether we should disable any use of @Converter set to auto-apply</td>
<td></td>
</tr>
</tbody>
</table>
@Entity
public class Person {
    @Basic
    @Convert(converter=MyURLConverter.class)
    URL website;
    ...
}

@Column
This annotation is used to define the column where a field is stored. Specified on the field/method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name for the column</td>
<td></td>
</tr>
<tr>
<td>unique</td>
<td>boolean</td>
<td>Whether the field is unique</td>
<td>true, false</td>
</tr>
<tr>
<td>nullable</td>
<td>boolean</td>
<td>Whether the field is nullable</td>
<td>true, false</td>
</tr>
<tr>
<td>insertable</td>
<td>boolean</td>
<td>Whether the field is insertable</td>
<td>true, false</td>
</tr>
<tr>
<td>updatable</td>
<td>boolean</td>
<td>Whether the field is updatable</td>
<td>true, false</td>
</tr>
<tr>
<td>table</td>
<td>String</td>
<td>Name of the table</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td>int</td>
<td>Length for the column</td>
<td>255</td>
</tr>
<tr>
<td>precision</td>
<td>int</td>
<td>Decimal precision for the column</td>
<td>0</td>
</tr>
<tr>
<td>scale</td>
<td>int</td>
<td>Decimal scale for the column</td>
<td>0</td>
</tr>
<tr>
<td>columnDefinition</td>
<td>String</td>
<td>DDL to use for the column (everything except the column name). This must include the SQL type of the column</td>
<td></td>
</tr>
</tbody>
</table>

@Entity
public class Person {
    @Basic
    @Column(name="SURNAME", length=100, nullable=false)
    String surname;
    ...
}

@JoinColumn
This annotation is used to define the FK column for joining to another table. This is part of a 1-1, 1-N, or N-1 relation. Specified on the field/method.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name for the column</td>
</tr>
<tr>
<td>referencedColumnName</td>
<td>String</td>
<td>Name of the column in the other table that this is the FK for. This is for use when you have a composite PK so acts as a way of aligning the respective columns. <strong>It is not to allow joining to some non-PK column</strong></td>
</tr>
<tr>
<td>unique</td>
<td>boolean</td>
<td>Whether the field is unique</td>
</tr>
<tr>
<td>nullable</td>
<td>boolean</td>
<td>Whether the field is nullable</td>
</tr>
<tr>
<td>insertable</td>
<td>boolean</td>
<td>Whether the field is insertable</td>
</tr>
<tr>
<td>updatable</td>
<td>boolean</td>
<td>Whether the field is updatable</td>
</tr>
<tr>
<td>columnDefinition</td>
<td>String</td>
<td>DDL to use for the column (everything except the column name). This must include the SQL type of the column</td>
</tr>
<tr>
<td>foreignKey</td>
<td>ForeignKey</td>
<td>Foreign key details if wanting to override provider default</td>
</tr>
</tbody>
</table>

```java
@Entity
public class Person {
    @OneToOne
    @JoinColumn(name="PET_ID", nullable=true)
    Animal pet;
    ...
}
```

There is a `@JoinColumns` annotation but you can achieve the same more cleanly with multiple `@JoinColumn` annotations.

**@UniqueConstraint**

This annotation is used to define a unique constraint to apply to a table. It is specified as part of `@Table`, `@JoinTable` or `@SecondaryTable`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>columnNames</td>
<td>String[]</td>
<td>Names of the column(s)</td>
<td></td>
</tr>
</tbody>
</table>
@Entity
@Table(name="PERSON", uniqueConstraints={@UniqueConstraint(columnNames={"firstName","lastName"})})
public class Person
{
   @Basic
   String firstName;

   @Basic
   String lastName;
   ...
}

See the documentation for Unique Constraints

@Index

This annotation is used to define the details for an Index. It is specified as part of @Table, @JoinTable, @CollectionTable or @SecondaryTable.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the index</td>
<td></td>
</tr>
<tr>
<td>columnList</td>
<td>String</td>
<td>Columns to be included in this index of the form colName1, colName2</td>
<td></td>
</tr>
<tr>
<td>unique</td>
<td>boolean</td>
<td>Whether the index is unique</td>
<td>false</td>
</tr>
</tbody>
</table>

See the documentation for Index Constraints

@ForeignKey

This annotation is used to define the details for a ForeignKey. It is specified as part of @JoinColumn, @JoinTable, @CollectionTable or @SecondaryTable.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the foreign key</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>ConstraintMode</td>
<td>Constraint mode</td>
<td>ConstraintMode.CONSTRAINT</td>
</tr>
<tr>
<td>foreignKeyDefinition</td>
<td>String</td>
<td>DDL for the FOREIGN KEY statement of the form FOREIGN KEY ( colExpr1 {, colExpr2}... ) REFERENCES tblIdentifier { ( otherColExpr1 {, otherColExpr2}... ) } { ON UPDATE updateAction } { ON DELETE deleteAction }</td>
<td></td>
</tr>
</tbody>
</table>

See the documentation for ForeignKey Constraints
@MapsId

This annotation would be used to mark the current field as mapping on to one of the "id" fields of the current class (embedded-id). This is not currently supported and there are better, more efficient ways of handling it.
DataNucleus Class-Level Extensions

The following annotations are specified at class-level and are vendor extensions providing more functionality than the JPA spec defines. Using these will reduce the portability of your application.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Class/Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@PersistenceAware</td>
<td>Class</td>
<td>Specifies that the class is not persistent but needs to be able to access fields of persistent classes (DataNucleus extension).</td>
</tr>
<tr>
<td>@DatastoreId</td>
<td>Class</td>
<td>Defines a class as using datastore-identity (DataNucleus extension).</td>
</tr>
<tr>
<td>@NonDurableId</td>
<td>Class</td>
<td>Defines a class as using nondurable identity (DataNucleus extension).</td>
</tr>
<tr>
<td>@ReadOnly</td>
<td>Class</td>
<td>Specifies that this class is &quot;read-only&quot; (DataNucleus extension).</td>
</tr>
<tr>
<td>@MultiTenant</td>
<td>Class</td>
<td>Specifies multi-tenancy details for this class (DataNucleus extension).</td>
</tr>
<tr>
<td>@SoftDelete</td>
<td>Class</td>
<td>Specifies that this class will be &quot;soft-deleted&quot; upon deletion of objects (DataNucleus extension).</td>
</tr>
</tbody>
</table>

@PersistenceAware

This annotation is used when you want to mark a class as knowing about persistence but not persistent itself. That is, it manipulates the fields of a persistent class directly rather than using accessors. **This is a DataNucleus extension.** Specified on the class.

```java
@PersistenceAware
public class MyClass
{
    ...
}
```

See the documentation for Class Mapping

@DatastoreId

This DataNucleus-extension annotation is used to define that the class uses datastore-identity. Specified on the class.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>generationType</td>
<td>GenerationType</td>
<td>Strategy to use when generating the values for this field. Has possible values of GenerationType TABLE, SEQUENCE, IDENTITY, AUTO, UUID. Note that UUID is only available if using the DN provided <code>javax.persistence.jar</code> (v2.2+)</td>
<td>GenerationType.AUTO</td>
</tr>
<tr>
<td>generator</td>
<td>String</td>
<td>Name of the generator to use. See @TableGenerator and @SequenceGenerator</td>
<td></td>
</tr>
<tr>
<td>column</td>
<td>String</td>
<td>Name of the column for persisting the datastore identity value</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@DatastoreId(column="MY_ID")
public class MyClass {
    ...
}
```

### @NonDurableId

This DataNucleus-extension annotation is used to define that the class uses non-durable identity. Specified on the class.

```java
@Entity
@NonDurableId
public class MyClass {
    ...
}
```

### @ReadOnly

This DataNucleus-extension annotation is used to define a class as being read-only (equivalent as read-only="true"). Specified on the class.

```java
@Entity
@ReadOnly
public class MyClass {
    ...
}
```
@MultiTenant

This DataNucleus-extension annotation is used specify multi-tenancy details for a class. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>String</td>
<td>Name of the multi-tenancy column for this class.</td>
<td>TENANT_ID</td>
</tr>
<tr>
<td>columnLength</td>
<td>int</td>
<td>Length of the multi-tenancy column.</td>
<td></td>
</tr>
<tr>
<td>disabled</td>
<td>boolean</td>
<td>Whether the multi-tenancy for this class is disabled.</td>
<td>false</td>
</tr>
</tbody>
</table>

@Entity
@MultiTenant(column="TENANT", columnLength=255)
public class MyClass
{
    ...
}

@SoftDelete

This DataNucleus-extension annotation is used to define a class as being soft-deleted whenever objects of this type are removed. Specified on the class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>String</td>
<td>Name of the soft-delete status column for this class.</td>
<td>DELETED</td>
</tr>
</tbody>
</table>

@Entity
@SoftDelete
public class MyClass
{
    ...
}
DataNucleus Field-Level Extensions

The following annotations are specified at field/method-level and are vendor extensions providing more functionality than the JPA spec defines. Using these will reduce the portability of your application.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Class/Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@SharedRelation</td>
<td>Field/Method</td>
<td>Specifies that the relation for this field/property is &quot;shared&quot; (DataNucleus extension).</td>
</tr>
<tr>
<td>@ReadOnly</td>
<td>Field/Method</td>
<td>Specifies that this field/property is &quot;read-only&quot; (DataNucleus extension).</td>
</tr>
<tr>
<td>@Index</td>
<td>Field/Method</td>
<td>Specifies an index on this field/property (DataNucleus extension).</td>
</tr>
<tr>
<td>@JdbcType</td>
<td>Field/Method</td>
<td>Specifies the JDBC Type to use on this field/property (DataNucleus extension).</td>
</tr>
<tr>
<td>@SqlType</td>
<td>Field/Method</td>
<td>Specifies the SQL Type to use on this field/property (DataNucleus extension).</td>
</tr>
<tr>
<td>@ColumnPosition</td>
<td>Field/Method</td>
<td>Specifies the column position to use on this field/property (DataNucleus extension).</td>
</tr>
<tr>
<td>@ValueGenerator</td>
<td>Field/Method</td>
<td>Specifies a non-JPA-standard value generator to use on this field/property (DataNucleus extension).</td>
</tr>
<tr>
<td>@Extension</td>
<td>Class/Field/Method</td>
<td>Defines a DataNucleus extension (DataNucleus extension).</td>
</tr>
<tr>
<td>@CreateTimestamp</td>
<td>Field/Method</td>
<td>Specifies that this field/property should store a creation timestamp when inserting (DataNucleus extension).</td>
</tr>
<tr>
<td>@CreateUser</td>
<td>Field/Method</td>
<td>Specifies that this field/property should store the current user when inserting (DataNucleus extension).</td>
</tr>
<tr>
<td>@UpdateTimestamp</td>
<td>Field/Method</td>
<td>Specifies that this field/property should store an update timestamp when updating (DataNucleus extension).</td>
</tr>
<tr>
<td>@UpdateUser</td>
<td>Field/Method</td>
<td>Specifies that this field/property should store the current user when updating (DataNucleus extension).</td>
</tr>
</tbody>
</table>

@SharedRelation

This DataNucleus-extension annotation is used to define a field with a (1-N/M-N) relation as being "shared" so that a distinguisher column is added. Specified on the field/property.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>value to be stored in the distinguisher column for this relation field</td>
<td></td>
</tr>
<tr>
<td>column</td>
<td>String</td>
<td>Name of the distinguisher column for this relation field</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>primaryKey</td>
<td>boolean</td>
<td>Whether the distinguisher column should be part of the PK (when in a join table)</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class MyClass {
    @OneToMany
    @JoinTable
    @SharedRelation(column="ADDRESS_TYPE", value="home")
    Collection<Address> homeAddresses;

    @OneToMany
    @JoinTable
    @SharedRelation(column="ADDRESS_TYPE", value="work")
    Collection<Address> workAddresses;
...
}
```

**@ValueGenerator**

This DataNucleus-extension annotation is used to allow use of non-JPA-standard value generators on a field/property. Specified on the `field/property`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy</td>
<td>String</td>
<td>Name of the strategy e.g &quot;uuid&quot;</td>
</tr>
</tbody>
</table>

```java
@Entity
public class MyClass {
    @ValueGenerator(strategy="uuid")
    String id;
...
}
```

**@ReadOnly**

This DataNucleus-extension annotation is used to define a field as being read-only (equivalent to `insertable="false", updateable="false"`). Specified on the `field/property`.
@Entity
public class MyClass
{
    @Basic
    @ReadOnly
    String someValue;

    ...
}

@CreateTimeStamp
This DataNucleus-extension annotation is used to define this field as being persisted with a
timestamp of the creation time of this object. Specified on the field/property.

@Entity
public class MyClass
{
    @CreateTimeStamp
    Timestamp createTime;

    ...
}

@CreateUser
This DataNucleus-extension annotation is used to define this field as being persisted with the
current user at insert of this object. Specified on the field/property.

@Entity
public class MyClass
{
    @CreateUser
    String createUser;

    ...
}

@UpdateTimestamp
This DataNucleus-extension annotation is used to define this field as being persisted with a
timestamp of the update time of this object. Specified on the field/property.
@Entity
class MyClass {
    @UpdateTimestamp
    Timestamp updateTime;
    ...
}

@UpdateUser
This DataNucleus-extension annotation is used to define this field as being persisted with the current user at update of this object. Specified on the field/property.

class MyClass {
    @UpdateUser
    String updateUser;
    ...
}

@index (field/method - extension)
This DataNucleus-extension annotation is used to define an index for this field/property. Specified on the field/property.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the index</td>
<td></td>
</tr>
<tr>
<td>unique</td>
<td>boolean</td>
<td>Whether the index is unique</td>
<td>false</td>
</tr>
</tbody>
</table>

class MyClass {
    @Index(name="ENABLED_IDX")
    boolean enabled;
    ...
}

@JdbcType
This DataNucleus-extension annotation is used to define the jdbc-type to use for this field/property. Specified on the field/property.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>JDBC Type (VARCHAR, INTEGER, BLOB, etc)</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class MyClass {
    @JdbcType("CHAR")
    boolean enabled;
    ...
}
```

### @SqlType

This DataNucleus-extension annotation is used to define the sql-type to use for this field/property. Specified on the **field/property**.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>SQL Type (VARCHAR, INTEGER, BLOB, UUID, etc)</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class MyClass {
    @SqlType("CHAR")
    boolean enabled;
    ...
}
```

### @ColumnPosition

This DataNucleus-extension annotation is used to define the column position to use for this field/property. Specified on the **field/property**.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Integer</td>
<td>position of the column (first is &quot;0&quot;, increasing)</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
public class MyClass {
    @ColumnPosition(0)
    boolean enabled;
    ...
}
```
@Extension

DataNucleus Extension Annotation used to define an extension specific to DataNucleus. Specified on the class or field.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendorName</td>
<td>String</td>
<td>Name of the vendor</td>
<td>datanucleus</td>
</tr>
<tr>
<td>key</td>
<td>String</td>
<td>Key for the extension</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>Value of the extension</td>
<td></td>
</tr>
</tbody>
</table>

```java
@Entity
@Extension(key="RunFast", value="true")
public class Person
{
    ...
}
```

There is an @Extensions annotation but you can achieve the same cleaner using multiple @Extension annotations.
Meta-Annotations

JPA annotations are all usable as part of meta-annotations. A meta-annotation is, in simple terms, a user-defined annotation that provides one or multiple other annotations (including annotation attributes). Let's provide a couple of examples

Firstly, say we have

```java
@Entity
@Cacheable
@MultiTenant(column="TENANT")
```

and need to put this on many classes. We can introduce our own annotation

```java
@Target(TYPE)
@Retention(RUNTIME)
@Entity
@Cacheable
@MultiTenant(column="TENANT")
public @interface MultiTenantCacheableEntity {
}
```

so now we can simply annotate a JPA entity with

```java
@MultiTenantCacheableEntity
public class MyClass
{
   ... 
}
```

A second example is where we are specifying several attributes on an annotation, such as

```java
@DiscriminatorColumn(name="DISCRIM", discriminatorType=DiscriminatorType.INTEGER)
```

so we introduce our own convenience annotation

```java
@Target(TYPE)
@Retention(RUNTIME)
@DiscriminatorColumn(name="DISCRIM", discriminatorType=DiscriminatorType.INTEGER)
public @interface MyDiscriminator {
}
```
so now we can simply annotate a JPA entity that needs this discriminator with

```java
@Entity
@TableDiscriminator
public class MyClass {
    ...
}
```

You can also make use of *meta-annotations* on fields/properties.