**Annotations**

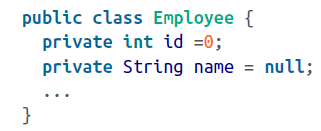
Annotations were introduced in java 5, and since then most of the toolkits have started using them heavily. Annotations are decorators applied at the class and variable level that define the metadata about the class itself. Hibernate loves annotations. As they are necessary tool in developing sophisticated ORM mappings.

In earlier chapters, we worked with the mapping of persistent classes using XML files. Using XML files for our configuration and mapping has both advantages and disadvantages. For one thing, they are quite simple and easily readable, but they are also quite verbose (contains too many words) and unfortunately do not impose type safety.

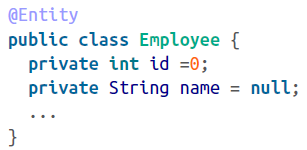
Annotations, on the other hand, are quite concise and enable compile-type checks straightaway. They are metadata decorations applied directly on the class, therefore enabling the entities to be managed effectively.

**Working through an Example**

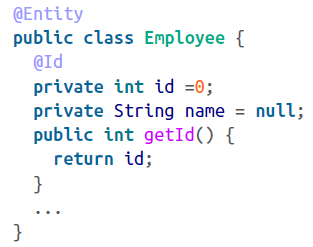
Let’s start with a simple persistent class, *Employee,* defined without annotations (pure POJO) here:



To make this class persistent using annotations, we first define it as *Entity.* We do so by annotating the class with the @*Entity* annotation.



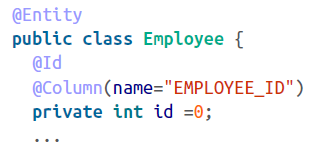
Now that we have a persistent entity, our next step is to define an identifier. Remember, all persistent entities must have their identifiers defined; otherwise, Hibernate will complain. So, we use the @Id annotation to indicate the object’s identifier to the framework. In our Employee class, our primary key is declared as an id variable, so we decorate the id with the relevant annotation:



When we annotate the id variable with the @Id annotation, as in the preceding example, Hibernate maps a field called ID from our *EMPLOYEE* table to the *id* variable on the *Employee* class.

Did you notice that we’ve decorated the variable instead of a getter method? Well, we can choose either the annotation on a variable, in which case Hibernate uses the accessor method to access the field. There is no hard-and-fast rule here; it’s pretty much a personal preference. I usually prefer annotating variable definitions, so that’s what I use for this book’s examples.

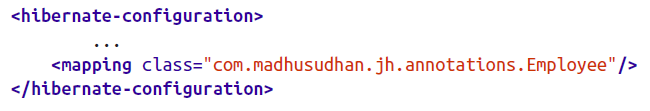
One more thing to note: most database will not allow us to create a field called ID, as it is a reserved keyword. In such circumstances, we usually change the field name- for example, *EMPLOYEE\_ID* in the preceding EMPLOYEE table’s case. As the object field and the column name do not match, we need to set the column name explicitly using an additional annotation, @Column, as shown here:



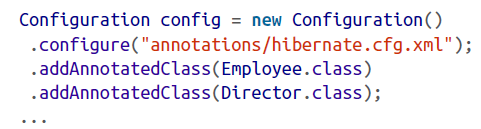
Before we delve into *Entity* and *Id* annotation options, let’s first see how the example is working.

We must let the Hibernate configuration know about annotated classes. You can do this in two ways: declaratively in your mapping file or programmatically in your class.

In your hibernate.cfg.xml file, use the mapping element to declare your class. Here’s how we do it via the mapping file:



Alternatively, use the add AnnotatedClass method if you prefer associating the annotation in your program.



Notice the chaining of the methods on the Configuration class. This is a convenient way of including additional methods in the class.

Run you test class, and it should persist your Employee object without any hiccups!