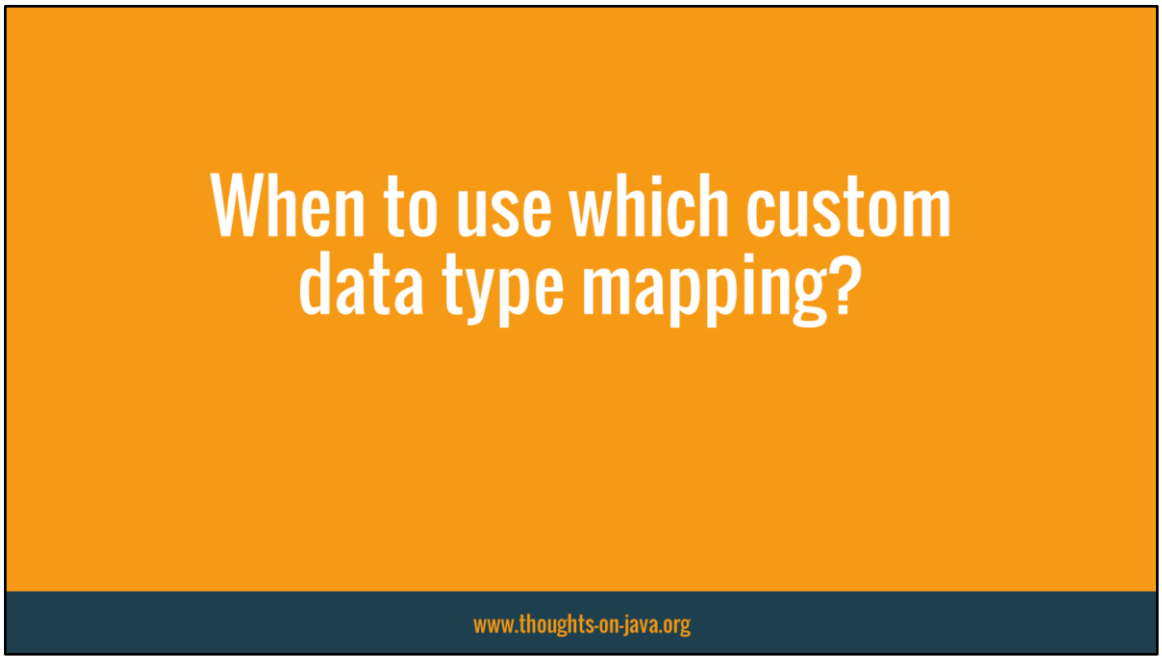
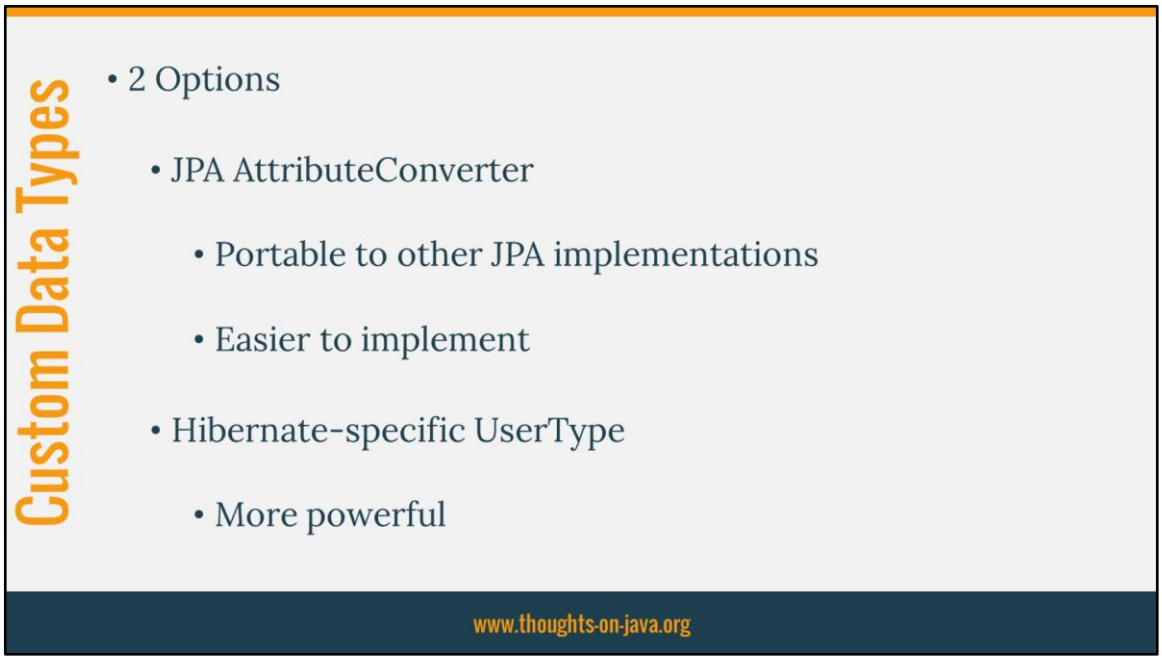
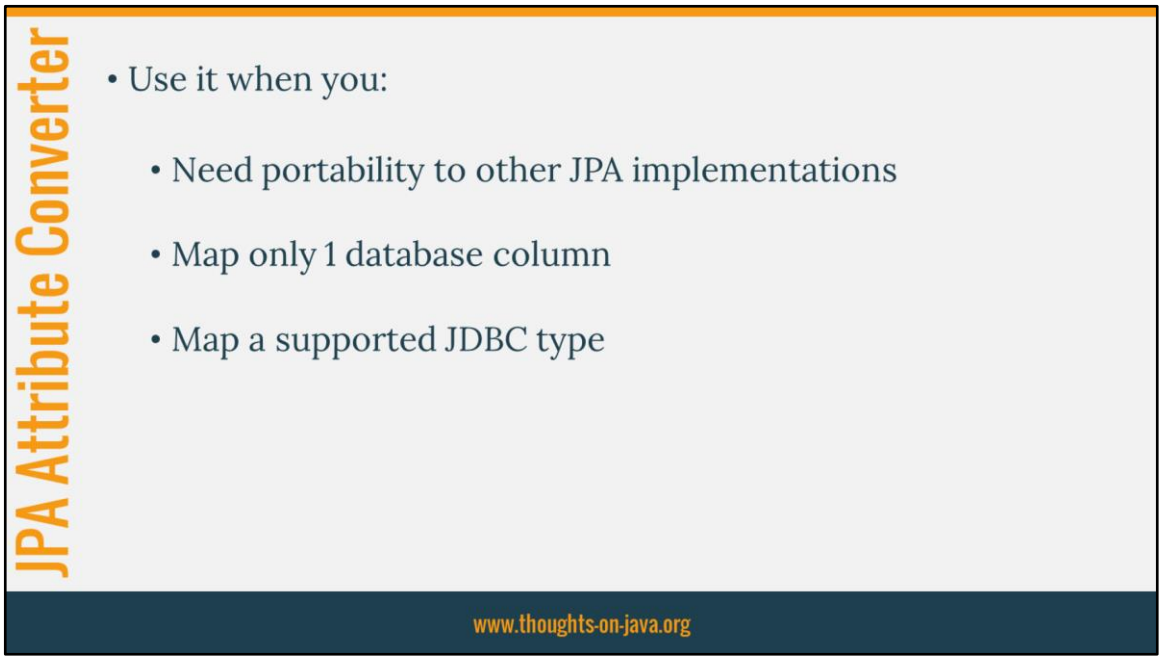


After showing you 2 options to implement custom data types, the interesting question is: Which one should you use?
As so often, that depends on your use case. So, let's discuss when to use which approach.



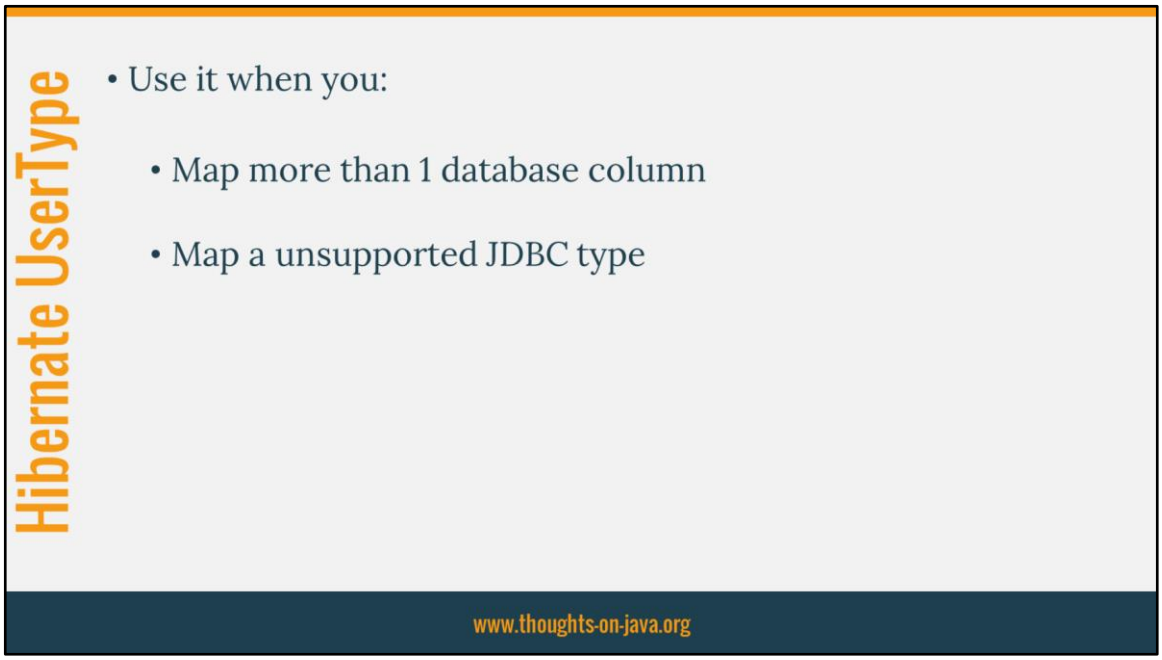
You can choose between JPA's *AttributeConverter*, which is easier to implement and portable to other JPA implementations, and Hibernate's *UserType*. The *UserType* is more powerful but also more complex.



You should use JPA's *AttributeConverter* if you have the following requirements:

1. You want a portable solution that you can also use with other JPA implementations.
2. You map the entity attribute to only one database column.
3. You convert it to a supported JDBC type.

The first requirement is optional. You can, of course, use an *AttributeConverter* even if you have no plans to port your application to other JPA implementations. But you have to fulfill the other two. JPA's *AttributeConverter* doesn't support mappings to multiple database columns, and as long as the bug HHH-9553 hasn't been fixed, you can't use it with unsupported JDBC types.



These are also the two reasons to use Hibernate's *UserType*.
You probably already recognized that I prefer to use standard JPA features, in general. That is, of course, one of the reasons why I prefer the *AttributeConverter*. But it's also much easier to implement if you don't need to map to multiple database columns or convert to unsupported JDBC types. In all other cases, you have to use Hibernate's proprietary *UserTypes*.