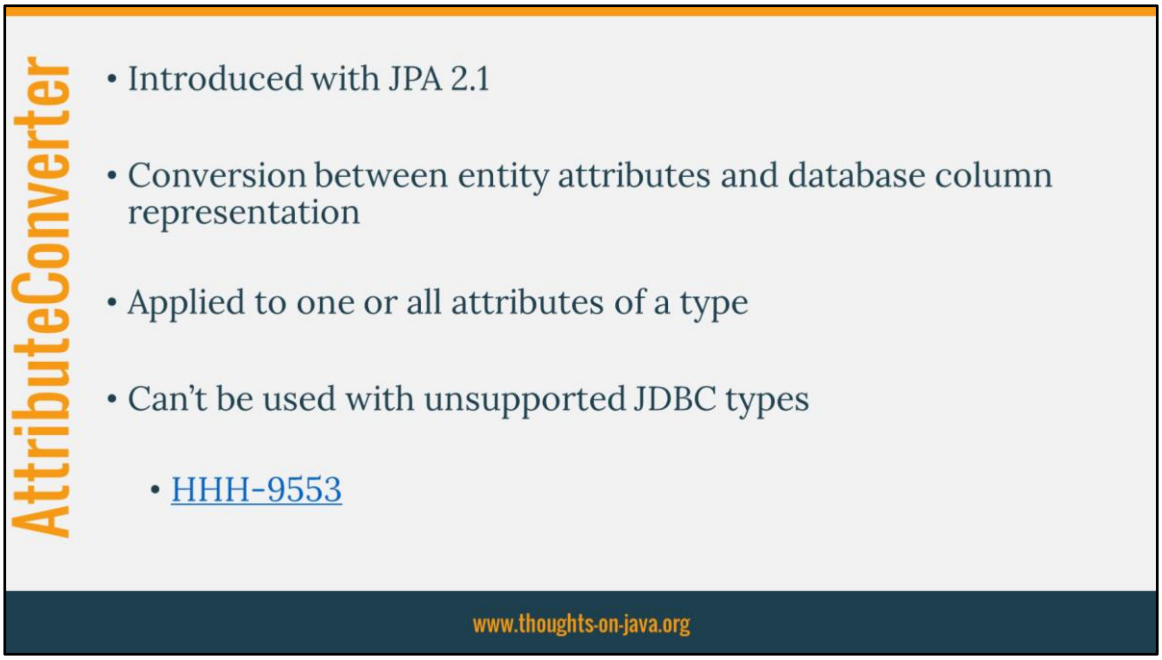
AttributeConverter

www.thoughts-on-java.org

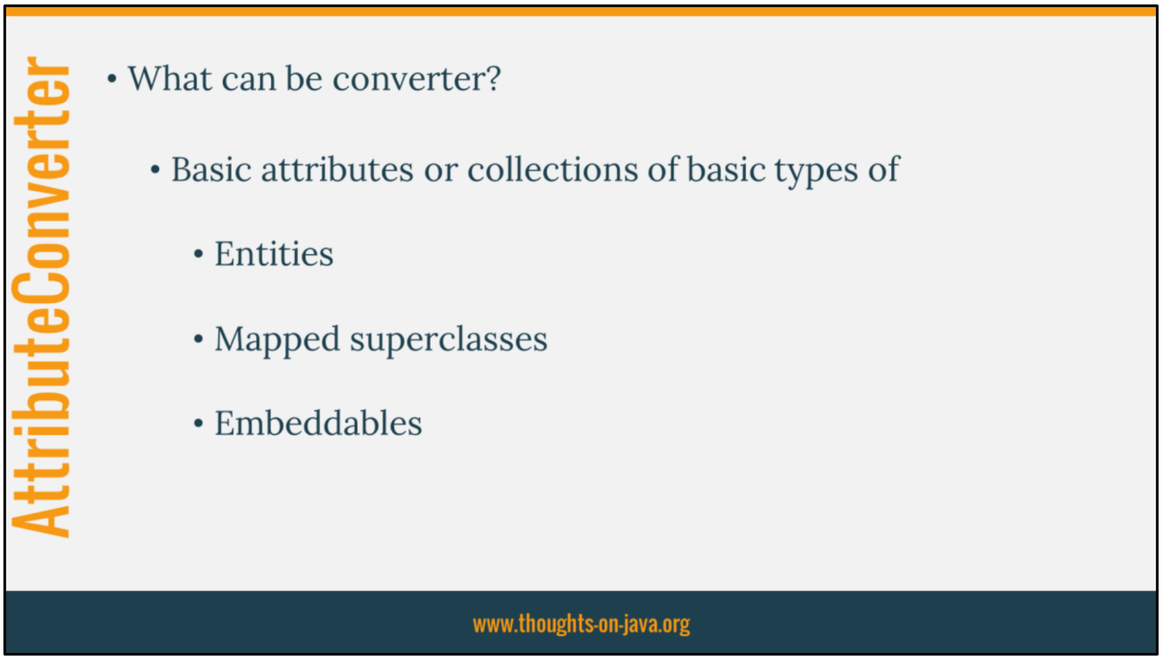
You sometimes want to use data types that are not supported by JPA and Hibernate, or you want to change their standard mapping. *AttributeConverter* are one option to provide your own conversion of an attribute value to its database representation and back.


AttributeConverter

- Introduced with JPA 2.1
- Conversion between entity attributes and database column representation
- Applied to one or all attributes of a type
- Can't be used with unsupported JDBC types
    - HHH-9553

www.thoughts-on-java.org

AttributeConverter were introduced with JPA 2.1. They provide a vendor-independent way to convert entity attributes to their database representation. You can either use them for a specific entity attribute or all attributes of a defined data type. I will show you how to do that on one of the following slides.
When an *AttributeConverter* is registered for an entity attribute, Hibernate applies it transparently for all read and write operations. That allows you to add, remove or change an *AttributeConverter* without changing any Criteria or JPQL queries. But you, of course, might need to migrate your database and change native SQL queries.
Unfortunately, the bug HHH-9553 prevents you from using an *AttributeConverter* with unsupported JDBC types, like the PostgreSQL-specific *JSONB* type. If you want to map a vendor-specific type, you have implement a Hibernate-specific *UserType*. I explain that in more detail in the UserType module.

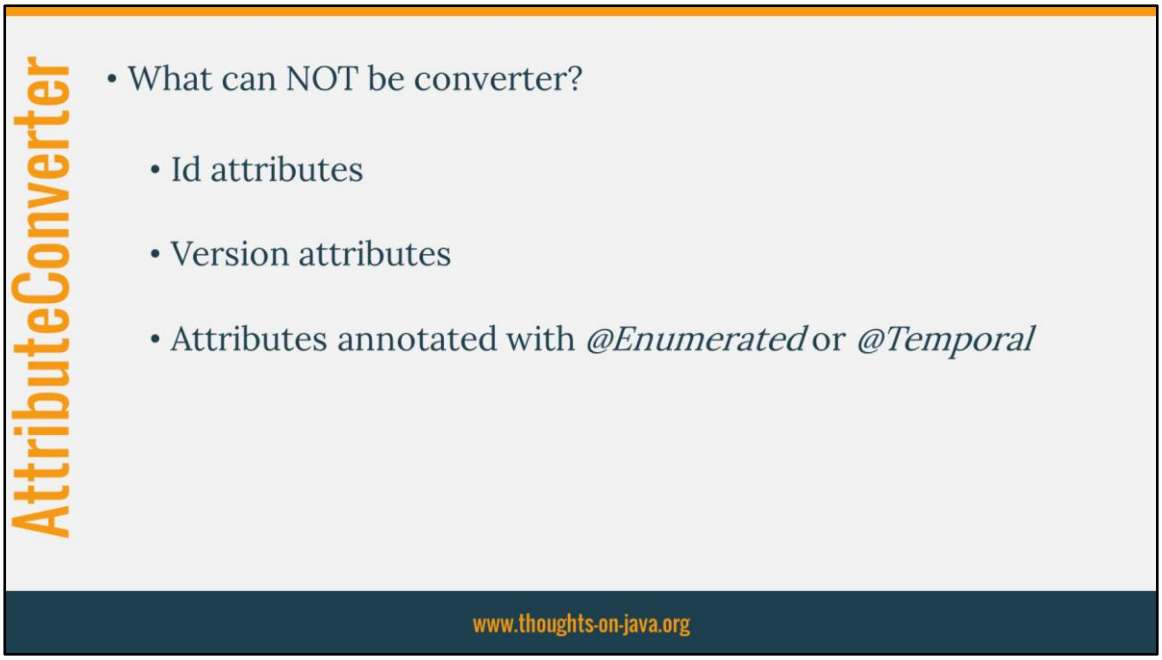


AttributeConverter

- What can be converter?
  - Basic attributes or collections of basic types of
    - Entities
    - Mapped superclasses
    - Embeddables

www.thoughts-on-java.org

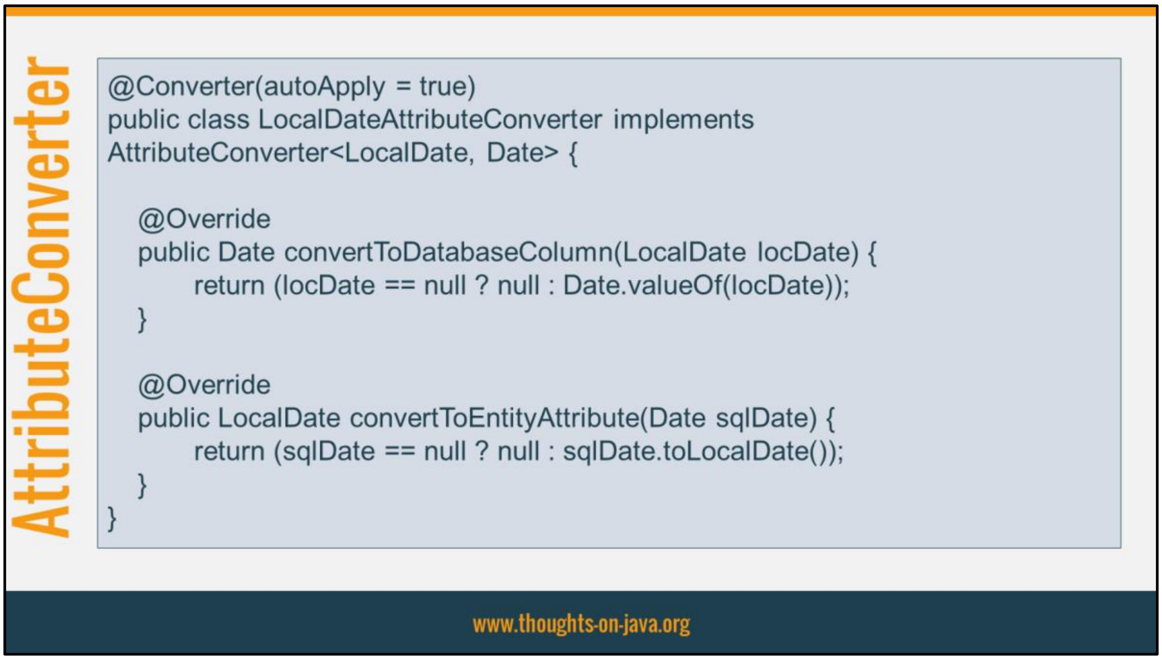
You can use *AttributeConverters* to convert basic attributes and collections of basic types that are defined for entities, mapped superclasses, and embeddable. If you assign an *AttributeConverter* to a collection of basic types, Hibernate will apply it to all its elements.
But there are also some cases in which you can't use an *AttributeConverter*.


AttributeConverter

- What can NOT be converter?
  - Id attributes
  - Version attributes
  - Attributes annotated with *@Enumerated* or *@Temporal*

www.thoughts-on-java.org

If your basic attribute is an id or version attribute or if it's denoted as Enumerated or Temporal, you can't apply an *AttributeConverter*. That is not a big deal for attributes annotated with *@Enumerated* and *@Temporal*. You can simply remove these annotations if you define your own attribute mapping. But it would be great if you could use an *AttributeConverter* on an id attribute. I already had a few situation in which I would've liked to do that. But for now, this is not allowed by the JPA specification, and I haven't heard of any plans to change that with JPA 2.2.



AttributeConverter

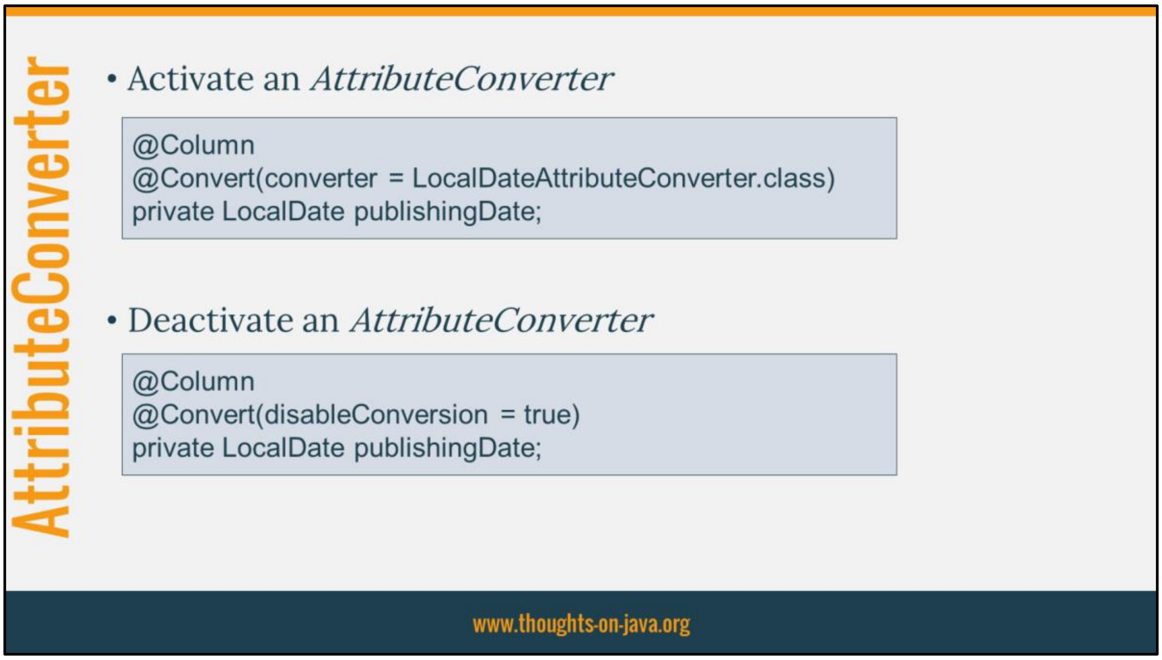
```
@Converter(autoApply = true)
public class LocalDateAttributeConverter implements
AttributeConverter<LocalDate, Date> {

  @Override
  public Date convertToDatabaseColumn(LocalDate locDate) {
       return (locDate == null ? null : Date.valueOf(locDate));
  }

  @Override
  public LocalDate convertToEntityAttribute(Date sqlDate) {
       return (sqlDate == null ? null : sqlDate.toLocalDate());
  }
}
```

www.thoughts-on-java.org

OK, let's have a quick look at an *AttributeConverter* definition and how you can use it before we switch the IDE. This code sample shows an *AttributeConverter* that converts a Java 8 *LocalDate* to a *java.sql.Date*. With Hibernate 5, you don't need this converter anymore. But if you're using older Hibernate versions or standard JPA 2.1, you can use this *AttributeConverter* to add support for Java 8's *LocalDate*.
As you can see, the *AttributeConverter* definition is not too complex. You just have to annotate your class with the *@Converter* annotation and to implement the *AttributeConverter* interface. The first type parameter of the *AttributeConverter* defines the type of the entity attribute and the second one the type of the database representation. There are no restrictions to the first type but the second one has to be supported by Hibernate. The *autoApply* attribute of the *@Converter* annotation defines if Hibernate shall automatically apply it to all attributes of the defined entity attribute type. In this example it's set to true and Hibernate will use it to convert all entity attributes of type *LocalDate* to a *java.sql.Date*. If you don't provide this parameter or set it to false, you have to register the *AttributeConverter* for the entity attributes you want to use it with.


AttributeConverter

- Activate an *AttributeConverter*

```
@Column
@Convert(converter = LocalDateAttributeConverter.class)
private LocalDate publishingDate;
```

- Deactivate an *AttributeConverter*

```
@Column
@Convert(disableConversion = true)
private LocalDate publishingDate;
```

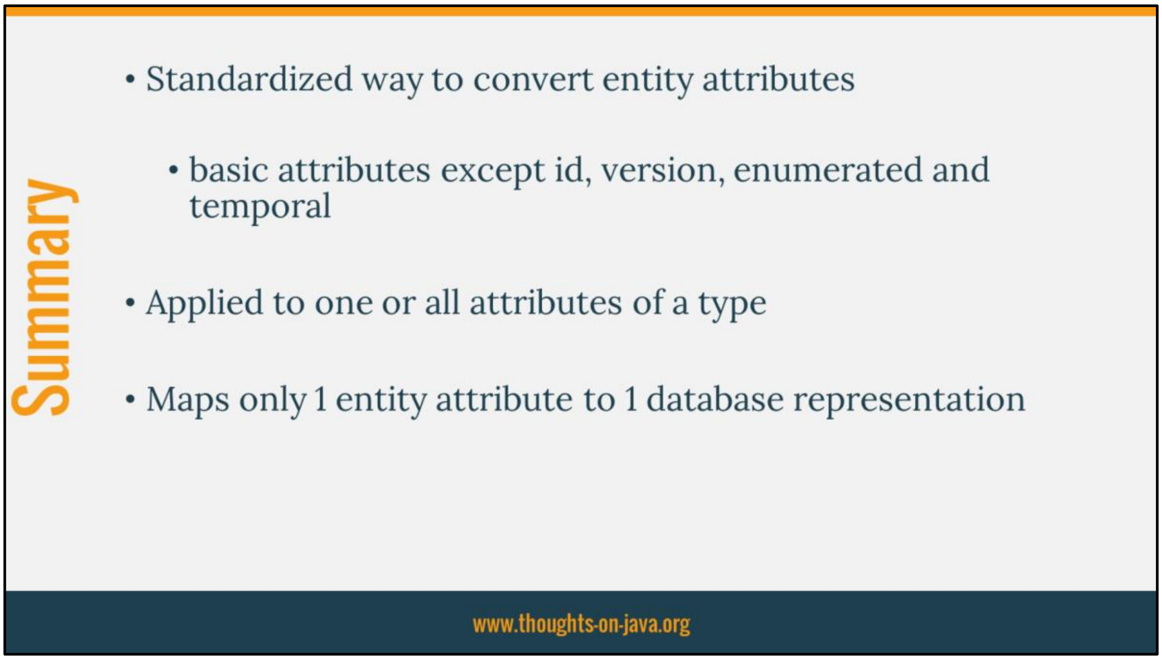
www.thoughts-on-java.org

You can do this the *@Convert* annotation as you can see in the first code snippet. You just need to add this annotation to an entity attribute and provide the class of the *AttributeConverter*.
You can also use the *@Convert* annotation to deactivate an *AttributeConverter* that was defined with *autoApply=true*. That can be a good approach if you want to use an *AttributeConverter* for almost all attributes of a particular type. In these cases, it might be easier to define it with *autoApply=true* and to deactivate it for a few entity attributes.
As always, you have to decide based on your application and use case, which approach fits best for you.

But enough theory, let’s switch to the IDE and try it out.

Demo

www.thoughts-on-java.org


Summary

- Standardized way to convert entity attributes
  - basic attributes except id, version, enumerated and temporal
- Applied to one or all attributes of a type
- Maps only 1 entity attribute to 1 database representation

www.thoughts-on-java.org

As you've seen, the *AttributeConverter* provides an easy and standardized way to convert entity attribute values to their database representation. You can apply it to all basic attributes that are not id or version attributes and that are not annotated with @Enumerated or @Temporal.
When you define the AttributeConverter, you can decide if Hibernate shall automatically apply it to all entity attributes of a certain data type or if you want to explicitly assign it to specific attributes.
One small downside of the *AttributeConverter* is that it only maps 1 entity attribute to 1 database representation. That doesn't seem like a big deal, but it sometimes prevents you from mapping complex data types like the classes of the new Java Money and Currency API.

Exercises

www.thoughts-on-java.org