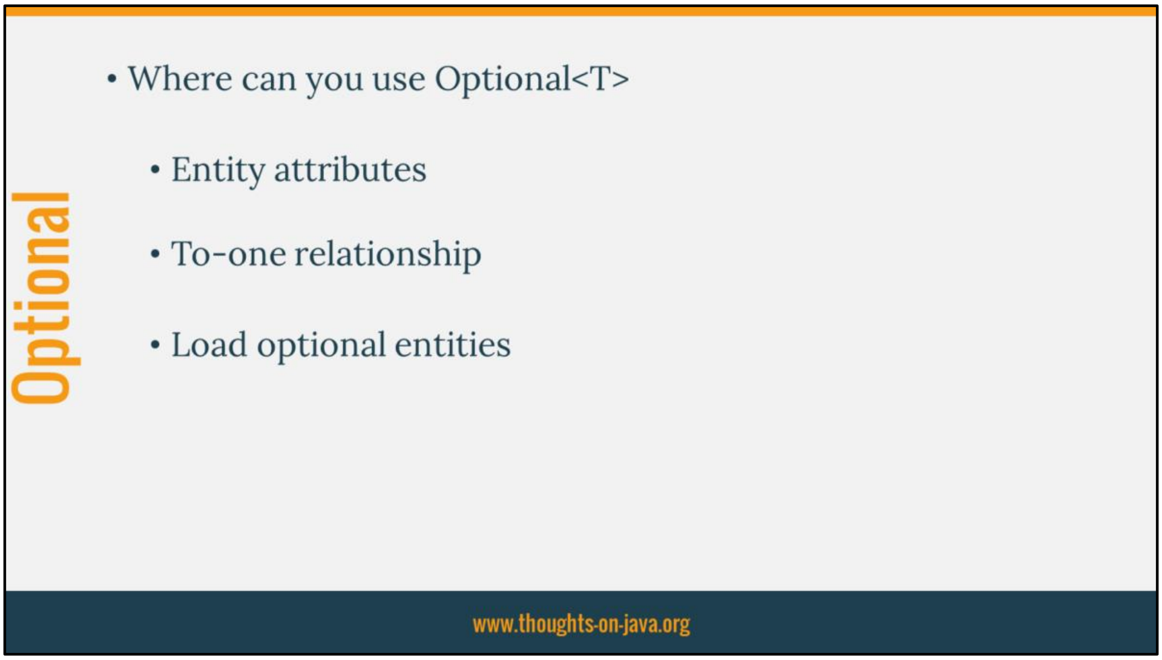
Optional<T>

www.thoughts-on-java.org

The container class Optional<T> is another one of the popular additions in Java 8. It’s often used to indicate that an attribute or the return value of a method might be null and needs to be handled accordingly to avoid NullPointerExceptions.

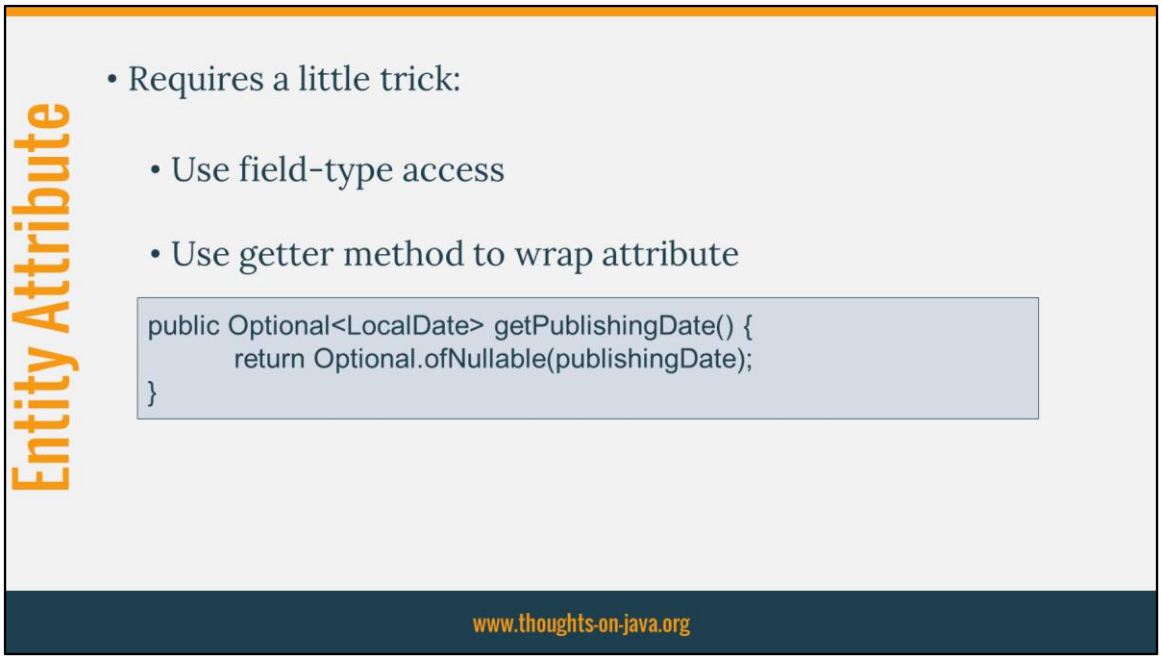


Optional

- Where can you use Optional<T>
  - Entity attributes
  - To-one relationship
  - Load optional entities

www.thoughts-on-java.org

Optional entity attributes and relationships are quite common. They use *null* to represent a missing value. That's fine from a persistence point of view, but it makes the domain model uncomfortable to use. You have to know that an attribute or relationship is optional and you have to learn it from the specification or a JavaDoc comment.
That's error-prone and seems like an ideal scenario for Java 8 *Optional*. Unfortunately, Hibernate doesn't allow you to use *Optional* as an attribute type. And even if it would, you shouldn't use it. *Optional* doesn't implement *Serializable*. That prevents your entity from implementing *Serializable*, and you can't transfer it to a client or store it in an *HttpSession*.
But you probably still want to use *Optional* in your domain model. So let's have a look how you can use it for optional entity attributes.



Entity Attribute

- Requires a little trick:
  - Use field-type access
  - Use getter method to wrap attribute

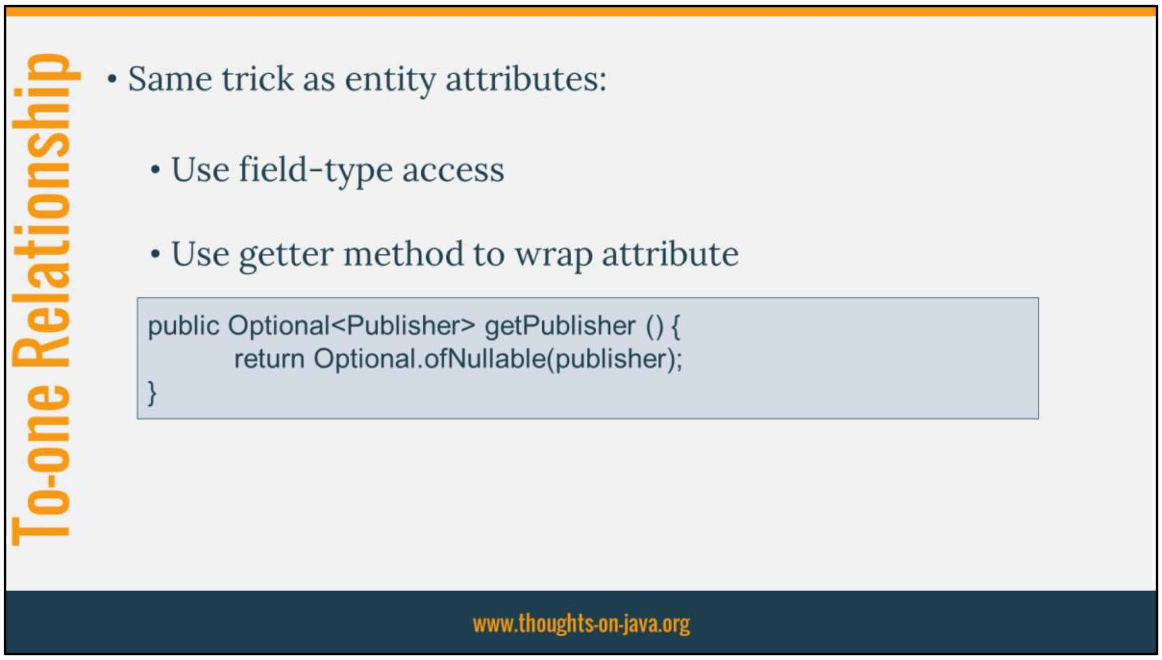
```
public Optional<LocalDate> getPublishingDate() {
        return Optional.ofNullable(publishingDate);
}
```

www.thoughts-on-java.org

As I said, you can't use Optional as an attribute type. You therefore need to apply a little trick. The good thing about it is that you don't need Hibernate 5 to implement it. When you use field-type access for your entity attributes, Hibernate will not use the getter and setter methods. This allows you to implement them in any way you like. You can, for example, implement a getPublishingDate() method that takes the publishingDate attribute and wraps it in an Optional.
Let's have a look at this in more detail.

Demo

www.thoughts-on-java.org


To-one Relationship

- Same trick as entity attributes:
  - Use field-type access
  - Use getter method to wrap attribute

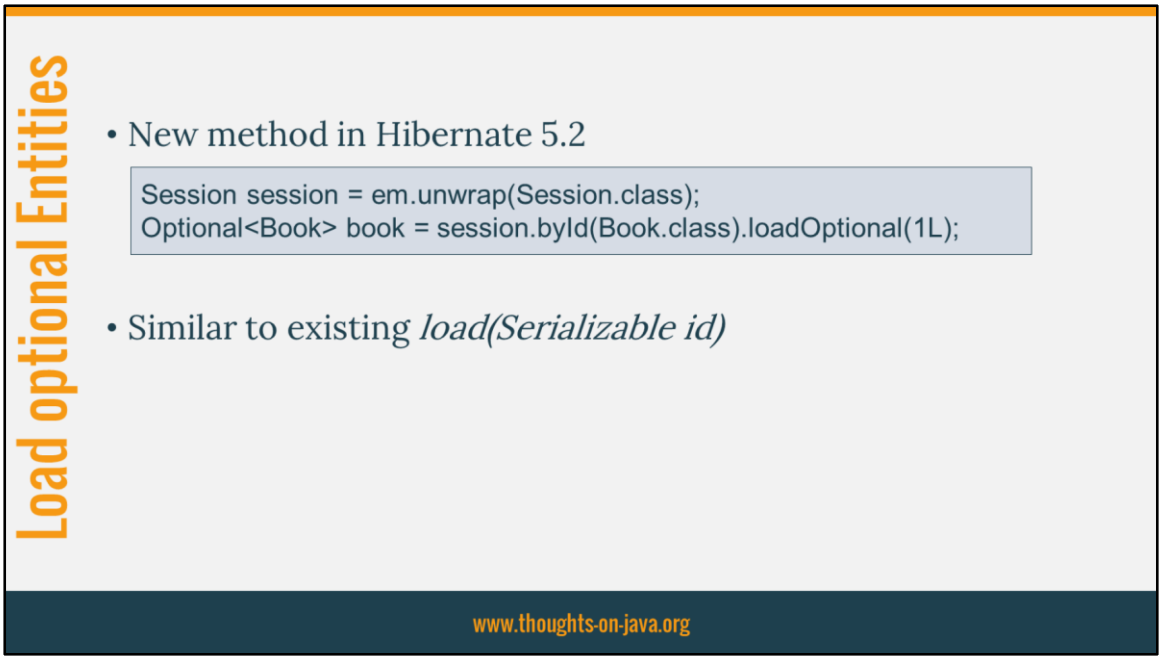
```
public Optional<Publisher> getPublisher () {
        return Optional.ofNullable(publisher);
}
```

www.thoughts-on-java.org

You can handle optional to-one relationships in the same way as optional entity attributes. You just need to use field-type access and implement a getter method that wraps the relationship attribute into an Optional.

Demo

www.thoughts-on-java.org


Load optional Entities

- New method in Hibernate 5.2

```
Session session = em.unwrap(Session.class);
Optional<Book> book = session.byId(Book.class).loadOptional(1L);
```

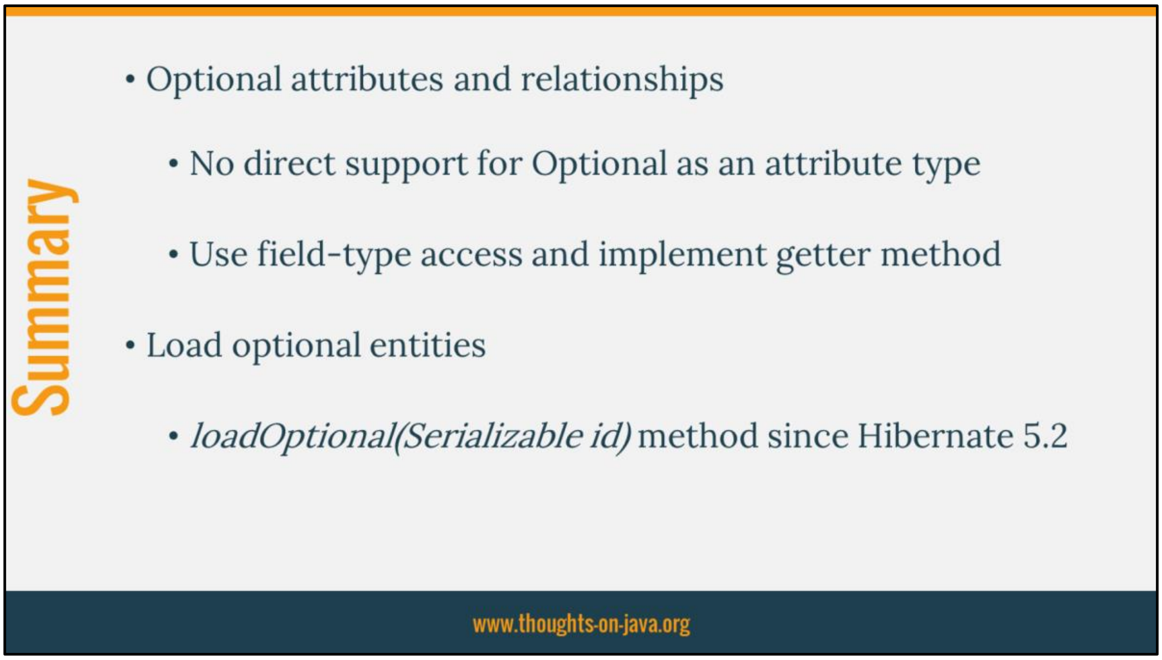
- Similar to existing *load(Serializable id)*

www.thoughts-on-java.org

The previous examples didn't require you to use a specific Hibernate version. But that's not the case for loading optional entities. If you want Hibernate to wrap the result of a database query in an *Optional*, you need the *loadOptional(Serializable id)* method. It was added to Hibernate's *IdentifierLoadAccess* interface in version 5.2. JPA, unfortunately, still uses Java 7 for all its APIs and can't use *Optional*.
As you can see in the code snippet the *loadOptional(Serializable id)* method is similar to the existing *load(Serializable id)* method. If you've used Hibernate's proprietary API in the past, you should be familiar with it. If not, you can compare it with the *EntityManager.find()* method of the JPA API.
OK, let's try it out before we wrap up this video.

Demo

www.thoughts-on-java.org



Summary

- Optional attributes and relationships
  - No direct support for Optional as an attribute type
  - Use field-type access and implement getter method
- Load optional entities
  - *loadOptional(Serializable id)* method since Hibernate 5.2

www.thoughts-on-java.org

As you've seen, you can use Java 8's Optional for attributes and relationships and when Hibernate loads an optional entity from the database.
The first 2 use cases require you to write some additional code. You need to wrap the entity attribute into an Optional because Hibernate doesn't support it as an attribute type. But as you've seen, you just have to use field-type access and implement your own getter method. That's done in a few seconds and doesn't create any further maintenance efforts. It, therefore, shouldn't be an issue to do this, if you think your domain model benefits from it.
Since Hibernate 5.2, you can use the *loadOptional(Serializable id)* method to load optional entities from the database. Hibernate will map the result of the query to an entity and wrap it into an Optional.

Exercises

www.thoughts-on-java.org