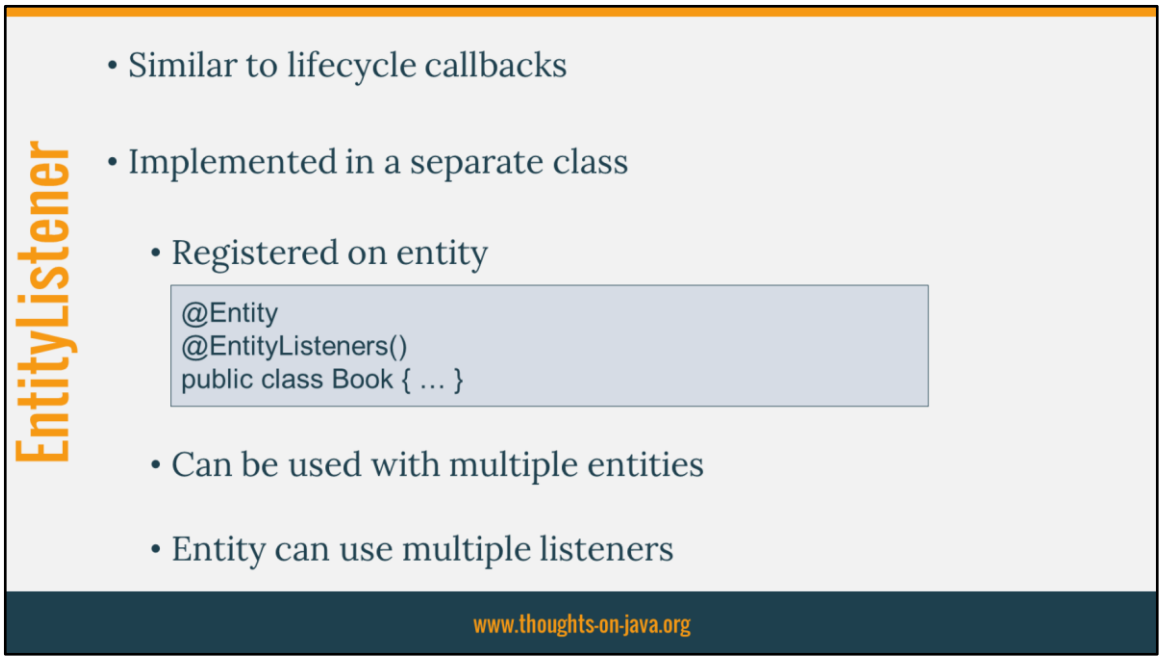
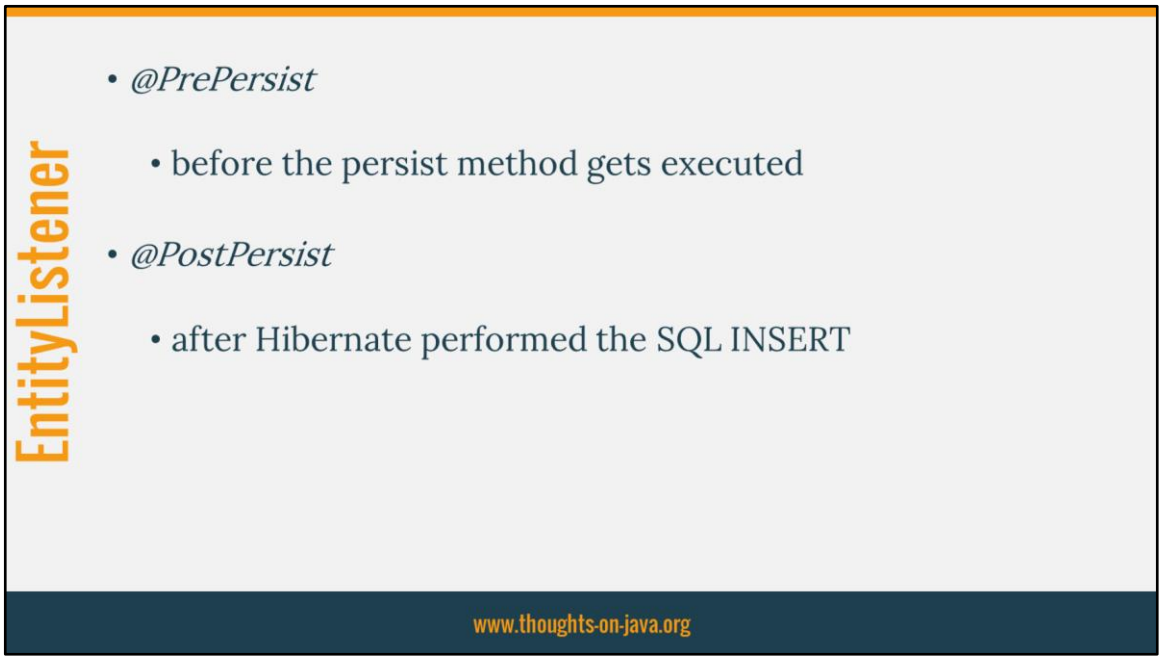
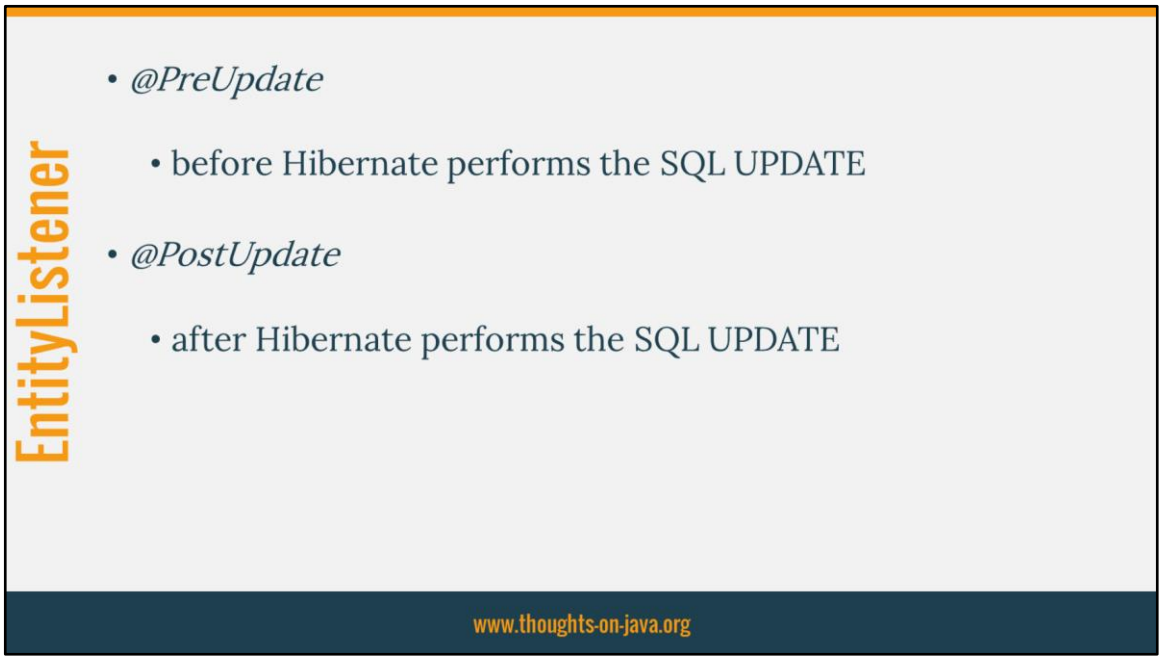
EntityListener are another option to perform custom code on lifecycle events. They use the same lifecycle event callbacks as I showed you in the previous video. But they are not part of the entity itself.



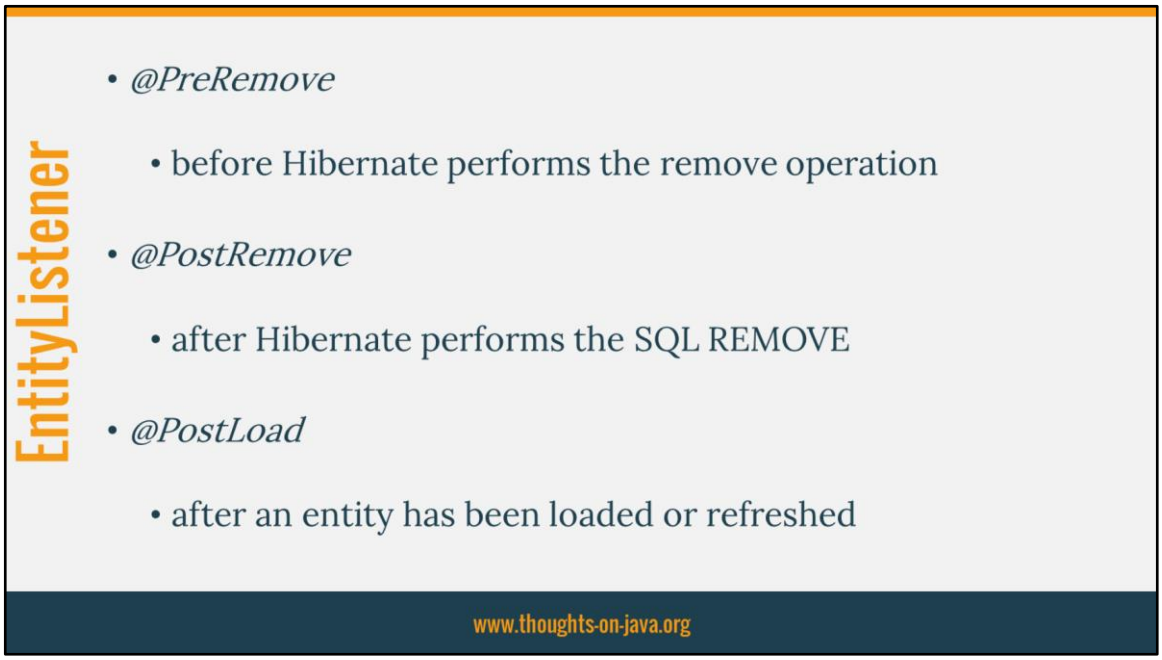
EntityListener are implemented in separate classes and need to be registered for an entity. The independence of the entity makes *EntityListeners* more flexible than the lifecycle callbacks I showed you in the previous video. You can use them with multiple entities, and you can also register multiple listeners for one entity. That allows you, for example, to implement an interface with different entities and to create an *EntityListener* which handles the lifecycle events of all entities which implement the interface.
Let's have a quick look at the callback annotations before we switch to the IDE to implement an *EntityListener*. As I said at the beginning, these are the same callback annotations as I showed you in the previous video. I will, therefore, keep my explanations short.



Methods annotated with *@PrePersist* get called before the persist method gets executed and methods annotated with *@PostPersist* after Hibernate performed the SQL INSERT statement.



You don't need to call a specific Hibernate and JPA method to trigger an update of an entity. It, therefore, makes sense to call methods annotated with *@PreUpdate* and *@PostUpdate* before and after performing the SQL UPDATE statement.



@PreRemove and *@PostRemove* follow the pattern of the persist callbacks. Hibernate calls methods annotated with *@PreRemove* before it performs the remove operations and the ones annotated with *@PostRemove* after executing the SQL REMOVE statement.
Methods annotated with *@PostLoad* are called after an entity has been loaded or refreshed.

Demo

www.thoughts-on-java.org

Let's summarize this video before we have a look at the exercises. *EntityListener* are similar to the lifecycle callbacks I showed you in the previous video. The main difference is that they are implemented in a separate class and that you have to register them for an entity. That provides additional flexibility because you can register multiple entity listeners for one entity and you can reuse existing *EntityListener* with multiple entities.

Exercises

www.thoughts-on-java.org