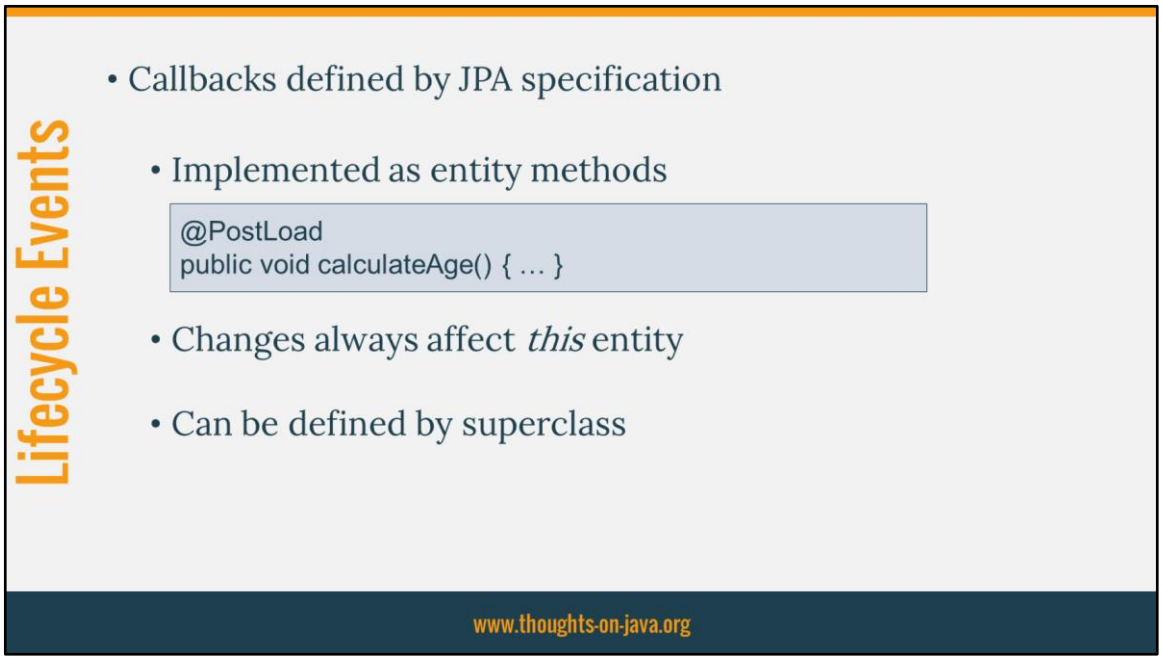
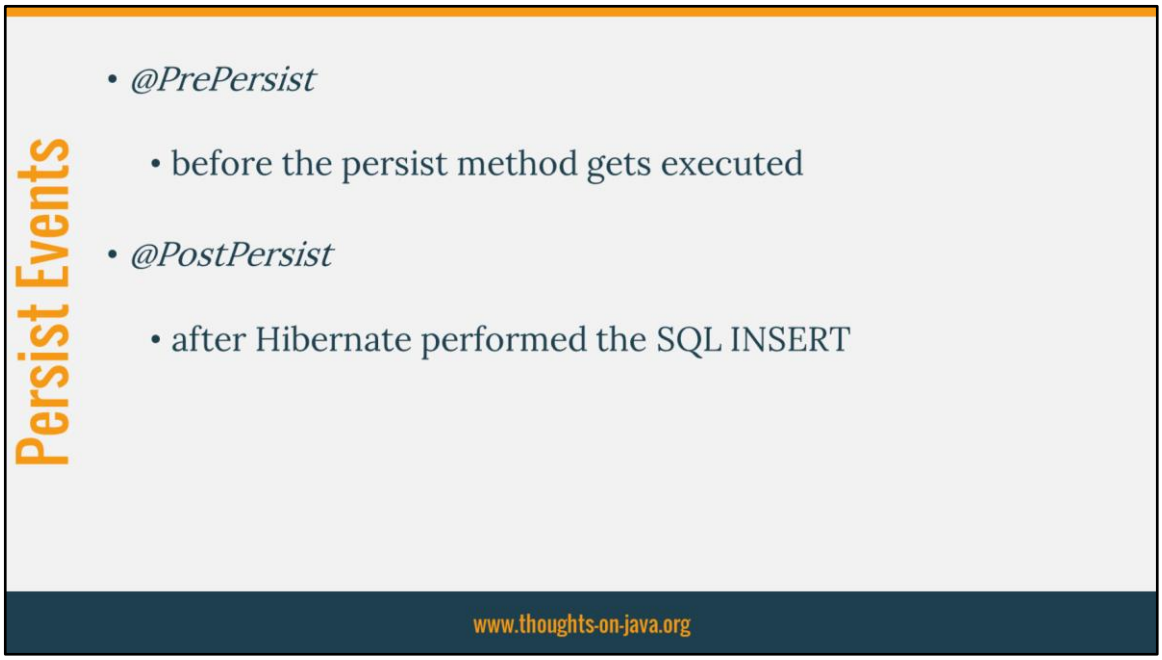
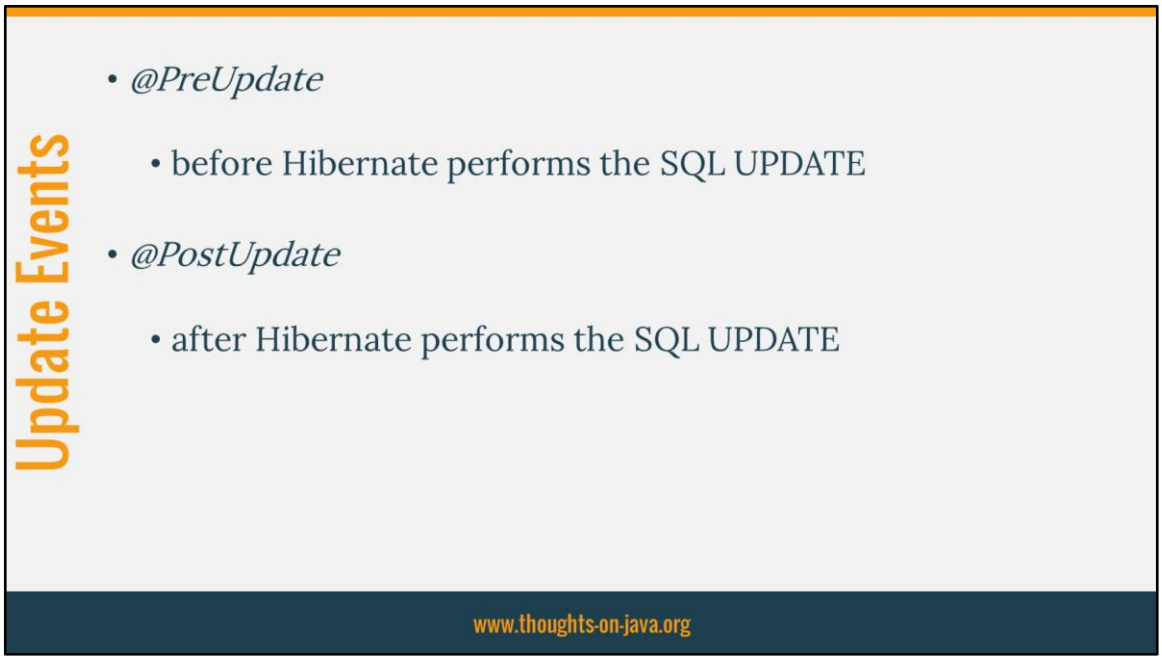
JPA offers 2 options to perform custom code on entity lifecycle events. One of them are the lifecycle event callbacks I want to show you in this video. They are often used to perform operations specific to one entity, like the calculation of transient entity attributes.



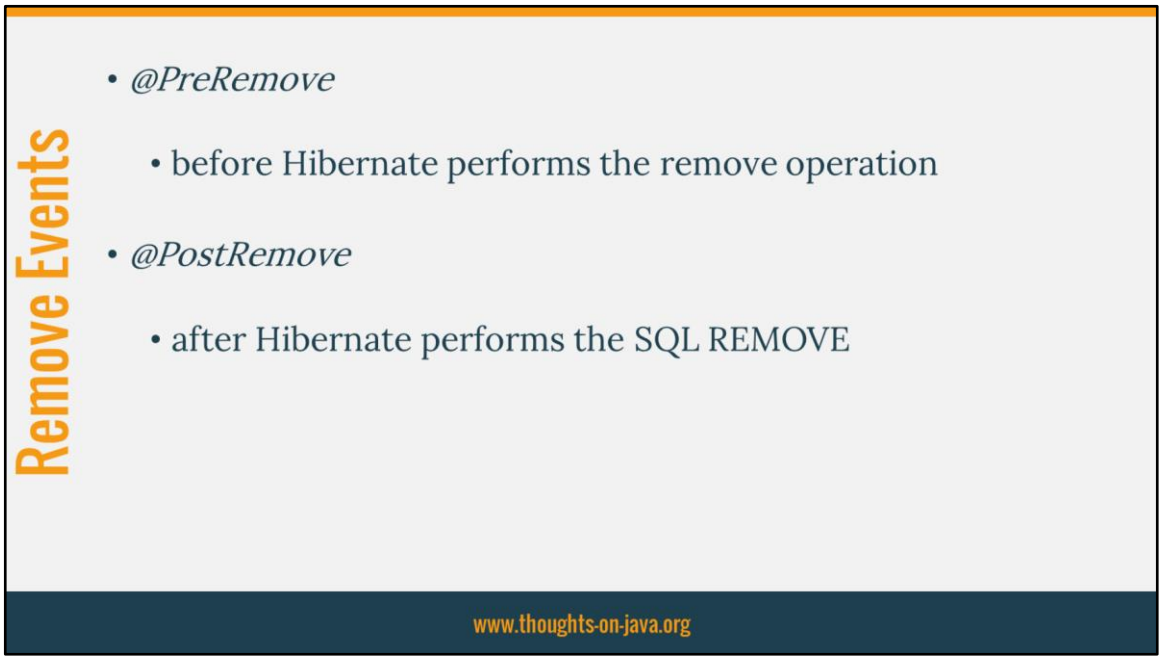
Lifecycle event callbacks are implemented as entity methods. You can either do that on a specific entity or a superclass. Hibernate triggers the callbacks defined on a superclass for all entities of the hierarchy.
The implementation of a lifecycle callback is pretty simple. You just have to add your method to an entity class and annotate it with one of the callback annotations I show you on the following slides. As you can see in the code snippet, the method doesn't support any parameters and its operations should only affect the entity for which the lifecycle event was triggered.



JPA defines 7 annotations which you can use to perform custom code before or after certain lifecycle events. Here you can see the 2 annotations that will trigger a method call when you persist an entity. *@PrePersist* gets called before the *persist* method of the *EntityManager* gets executed and *@PostPersist* after Hibernate performed the SQL INSERT statement.
Keep in mind that Hibernate can delay the execution of the SQL statement. A method annotated with *@PostPersist* might, therefore, be called shortly after you called the *persist* method on the *EntityManager,* when you perform the next database query or any other operation that forces Hibernate to *flush* the current session or at the end of the transaction.



The update callback annotations follow a similar pattern as the persist annotations I showed you on the previous slide. Hibernate calls a method annotated with *@PreUpdate* before it performs the SQL UPDATE statement and one annotated with *@PostUpdate* after executing the SQL statement.



The callback annotations for remove events follow the same pattern. A method annotated with *@PreRemove* gets called before Hibernate performs the remove operation and *@PostRemove* after performing the SQL REMOVE statement.

The load events don't follow the pattern and only support *@PostLoad* callbacks. Hibernate calls methods annotated with this annotation after it loaded or refreshed an entity.
Enough theory, let's switch to the IDE and use a callback annotation.

Demo

www.thoughts-on-java.org

The JPA specification defines 7 annotations to identify methods that shall be triggered by certain lifecycle events. These allow you to perform custom code before inserting, updating and deleting an entity and after Hibernate performed a load, insert, update or delete operation.
The callback methods are implemented on each entity class or on a superclass within an inheritance hierarchy. As you've seen in the code sample, callbacks defined for a superclass will also be called for all subclasses.

Exercises

www.thoughts-on-java.org