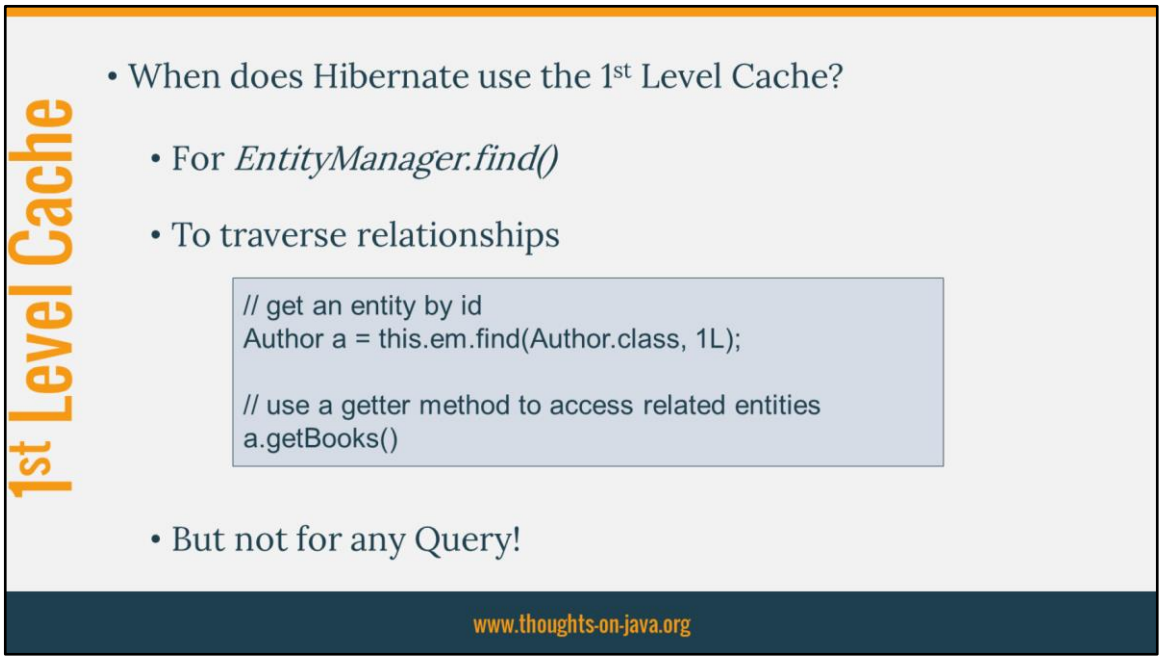
The first level cache is used in all of your applications and a lot of developers don‘t even know about it. The good thing is, you most often also don‘t need to know or care about it. Hibernate does everything on its own and uses the first level cache internally to optimize the communication with the database.

The first level cache is activated by default and Hibernate stores all entities in it, which were used within a session. This provides two main benefits:
It avoids additional database round trips, if you want to get an entity from the EntityManager that was already used within this session.
And Hibernate tries to delay write operations as long as possible and therefore also puts the changed entities into the first level cache. Each session has its own cache and cannot see the entities from other Hibernate sessions.

As you will see in the demo, all these things are hidden by the EntityManager API so that it is easy to forget that the first level cache is involved in several operations.

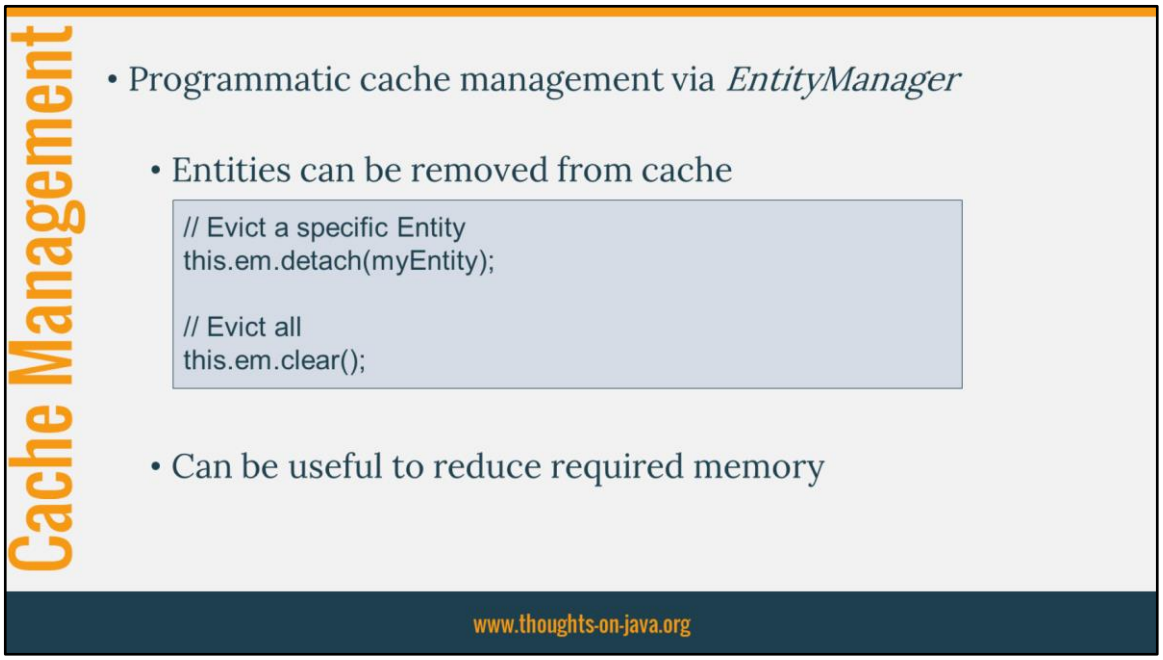


But that has also the disadvantage, that it is not that obvious if an operation uses the first level cache or not. Hibernate uses the cache only, if you use the find method on the EntityManager or if you traverse defined relationships to other entities. But not, if you use a JPQL, native or Criteria Query.

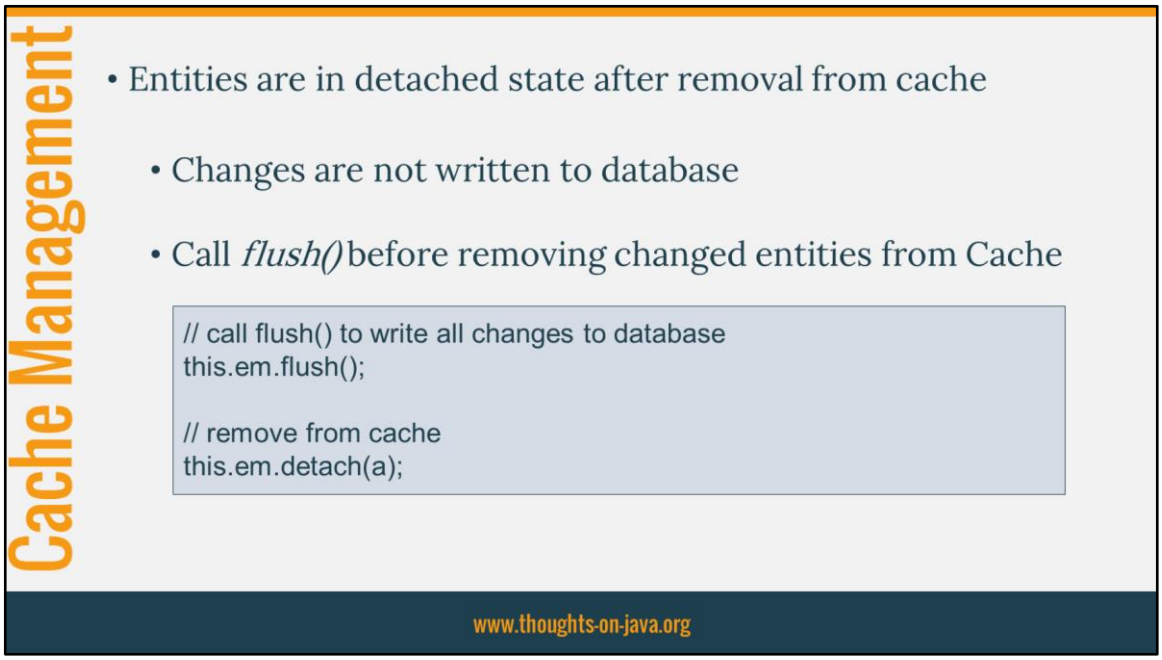
Let's get into the IDE and try it out.

Demo

www.thoughts-on-java.org



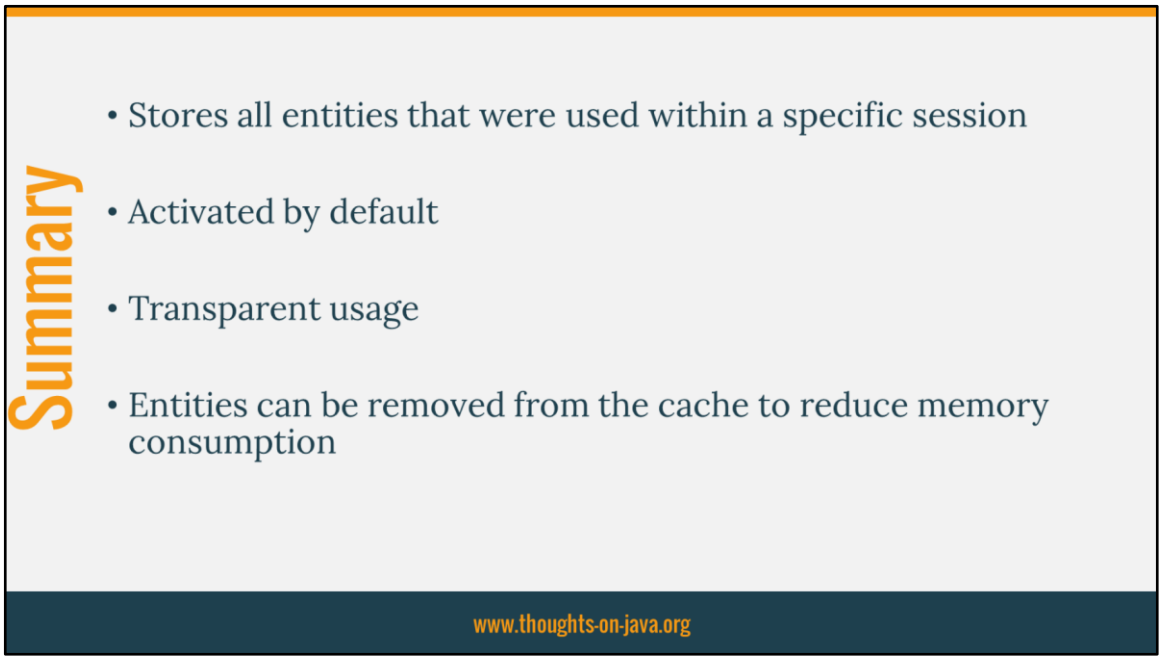
As you have seen, in general it is really easy to use the first level cache because you don't have to do anything. But as good as the first level cache is in most of the regular use cases, it can also become an issue in very huge transactions, like for batch import. Keeping thousands or even hundreds of thousands of entities in the cache requires a lot of memory and management effort. As a result, the first level cache can slow down your application in these use cases. If that is the case, you need to remove entities from your cache programmatically. This can be done via the methods detach and clear of the EntityManager. Detach removes a specific entity from the cache and clear removes all of them.



The entity will be in detached state after it was removed from the first level cache. That saves some memory but it also means that Hibernate will not write any changes on this entity to the database. So you better make sure that all changes are stored in the database before you detach the entity. This can be done by calling the flush method on the EntityManager.

Demo

www.thoughts-on-java.org



That‘s all about the first level cache for now. Let me summarize the most important things before we get into the exercises.
The first level cache is activated by default and stores all entities which were used within a specific session. Hibernate hides the calls to the cache behinds its API so that you don‘t need to take care of anything. Very huge sessions which work with a lot of entities might require to remove entities programmatically from the cache to reduce the memory footprint.

Exercises

www.thoughts-on-java.org