Hibernate

Table Of Contents

[1. Enabling Second Level Cache 3](#_Toc511054806)

[2. Difference between load and save 3](#_Toc511054807)

[3. Difference between save and persist 4](#_Toc511054808)

[4. Caching Mechanism in BIS 4](#_Toc511054809)

# Enabling Second Level Cache

The following are the steps involved in the process of enabling the second level cache.

Step 1: The hibernate.cfg.xml file will be modified to enable the second level cache and also to define the second level cache provider.

*<property name=“cache.use\_second\_level\_cache”>true</property>*

*<property name=”cache.provider\_class”>org.hiernate.cache.EhCacheProvider</property>*

Step 2: The entity that is to be cached will be configured by annotating the respective class with @Cacheable annotation.

Step 3: The last step would be to define the caching strategy which will be only among the following.

* + - **NONSTRICT\_READ\_WRITE**: This strategy will be applied only if the cache has to be updated occasionally and it is more unlikely that two transactions simultaneously update the same data.
    - **READ\_ONLY**: If the application requires reading but not modifying the data the READ\_ONLY strategy can be used which is perfectly compatible with clusters as well.
    - **READ\_WRITE**: If the application needs to update data, a read-write cache might be appropriate. This cache strategy should never be used if serializable transaction isolation level is required. If the cache is used in a JTA environment, you must specify the property hibernate.transaction.manager\_lookup\_class and naming a strategy for obtaining the JTA TransactionManager. In other environments, you should ensure that the transaction is completed when Session.close() or Session.disconnect() is called. If you want to use this strategy in a cluster, you should ensure that the underlying cache implementation supports locking. The built-in cache providers do not support locking.
    - **Transactiona**l: Such a cache can only be used in a JTA environment and you must specifyhibernate.transaction.manager\_lookup\_class.

Example: @Cache(usage=CacheConcurrencyStrategy.READ\_ONLY)

# Difference between load and save

* The first difference is that get always returns the object from the data base where as load returns a proxy object and the actual query to DB is done only when the respective fields are accessed.
* The next difference is that if we invoke get on a primary key which does not have any value in the data base, when the ‘s field is accessed there will be null pointer exception. If we perform the same scenario using Load method it would result in a Hibernate exception.

Message: Exception in thread "main" org.hibernate.ObjectNotFoundException: No row with the given identifier exists: [entityClasses.Service#V3MOD]

# Difference between save and persist

* The first difference is that, persist is supported by JPA while the save is only supported by Hibernate.
* The second difference is both save and persist requires to generate a unique id (which is the primary key) so that the entity can be associated to the currently running persistence context the save method returns the value but persist method returns void.

# Caching Mechanism in BIS

During the start- up of the application a set of Map variables are defined with the signature to accommodate the required values these variables remain unchanged through- out the application.

As these fields are global to the application to ensure that they are not modified, the fields are marked as private and only the getters are defined for the same.

For example: Map<String, List<String>> where the first String represents the service code and the second list represents the list of associated service characters. This saves the time incurred in the traversal of the list.

# Persist, Save, Update and Merge Functions

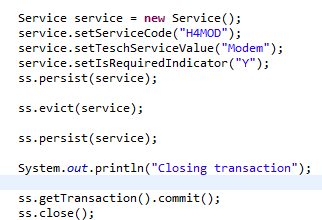
## Persist

The function is used add an object to the data base. When the persist method is called on the object the data does not get saved to the DB immediately rather it goes to the data base upon committing or flushing or closing the transaction.

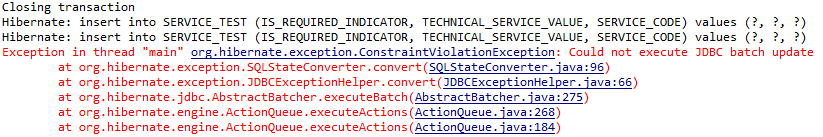
The method “***evict***” method is used to detach an object from the session.

Consider the below mentioned example where a persisted object is detached once again when we try to persist the same object the action results in an exception.

*Example:*



*Exception:*



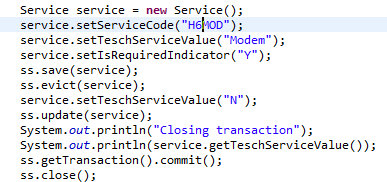
The main reason for the exception is that since the object is detached and once again persisted the “persist” command considers that as a new object and tries multiple insert on the same object which results in an exception.

And also in the event if there is a field for which the id value needs to be generated from the DB there would be a conflict.

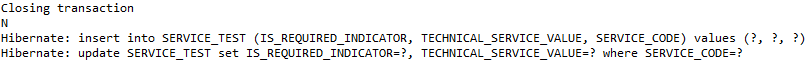
## Update Function

The update function is called within a same session it checks if the object has been modified from the one available on the data base if yes it triggers an update query to persist the modified object and also tracks any further changes to the object.

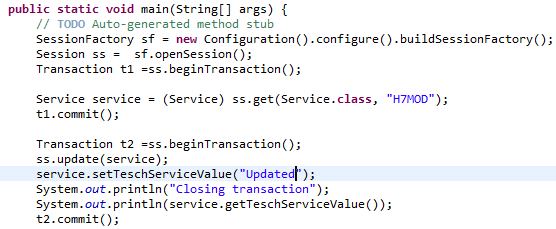
In the event if the object has been detached from the session the update method always performs an update query on the DB irrespective of the fact the object is modified or not.



*Console output:*



***First Execution*:**



*Console Output:*

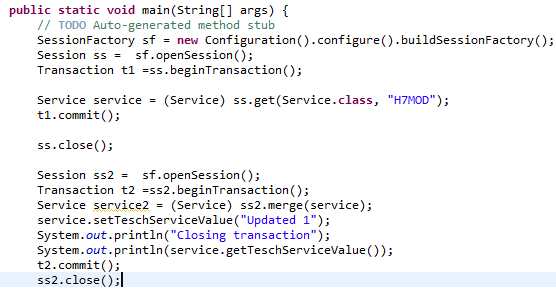


*Second Execution*: Upon running the same code again the following is the output but the difference is that the “update” query is not fired as there is not change to the object.



## Merge Function

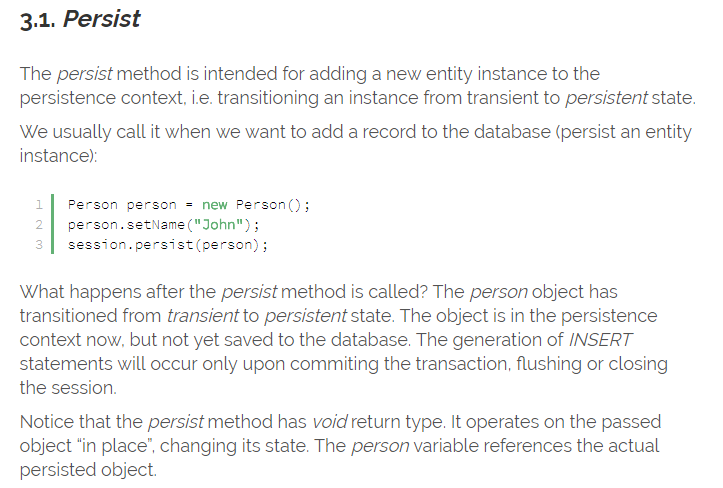
The merge function is also used to update the values but the only difference is that it generates a copy of the object and starts tracking the clone version of the object and not the one passed to it.

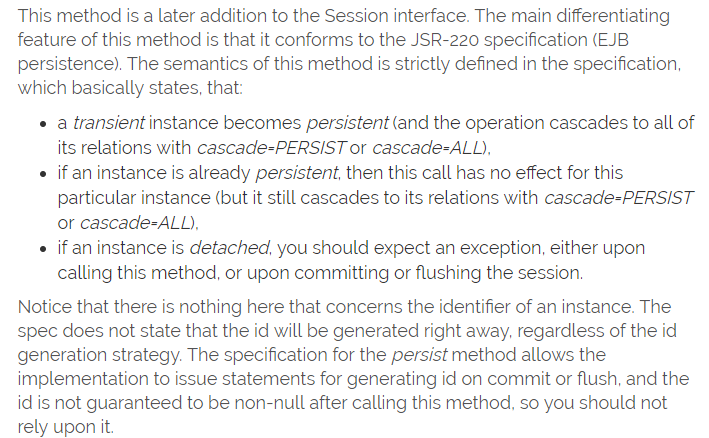


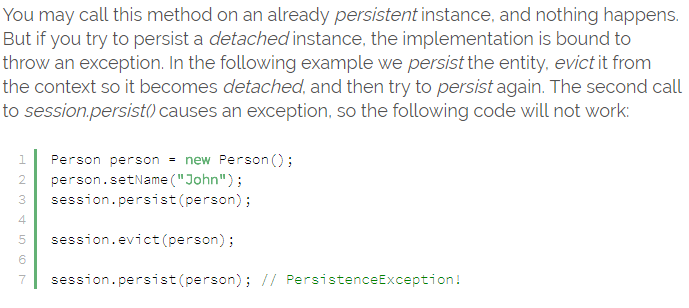
## saveorUpdate Function

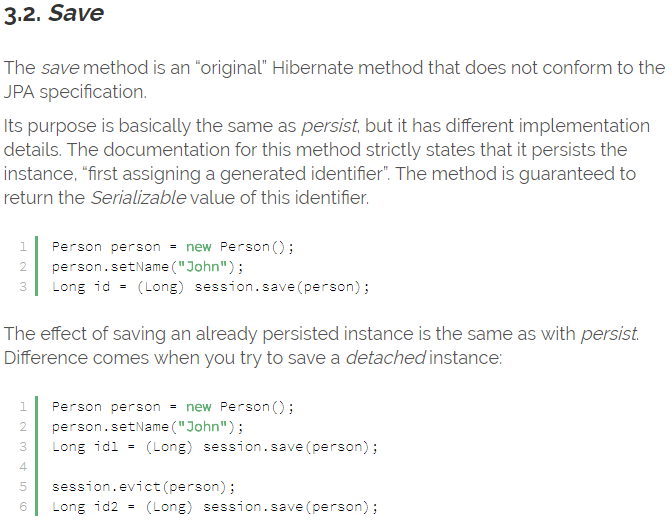
It triggers save or update based on the data availability. Say if the data does not exists it would trigger save if the data exists but some of it fields requires an update then and update query is triggered.

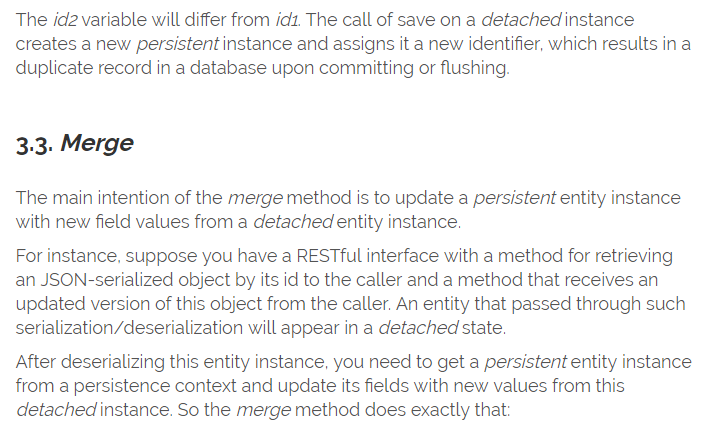
# Persist, Save, Update and Merge Functions

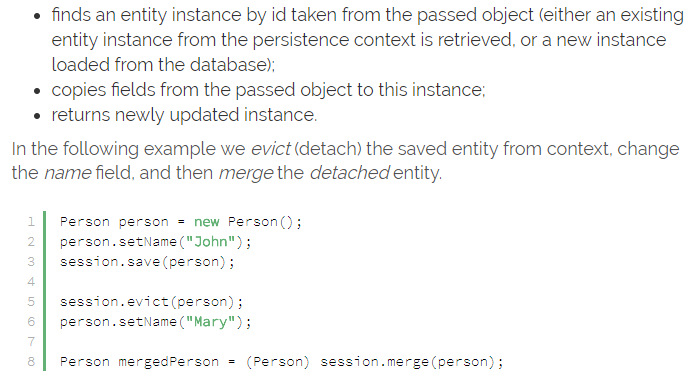


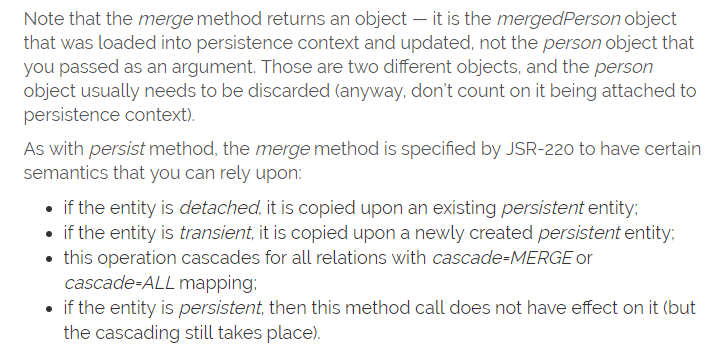


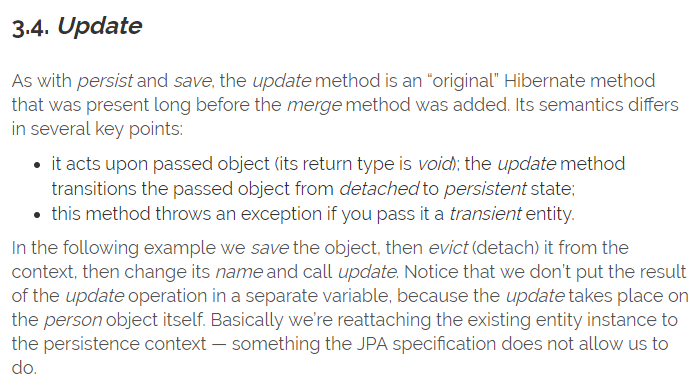


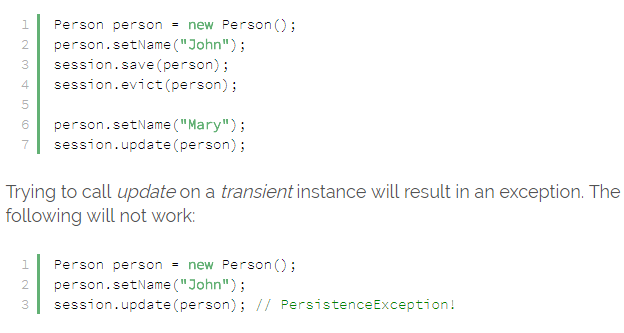


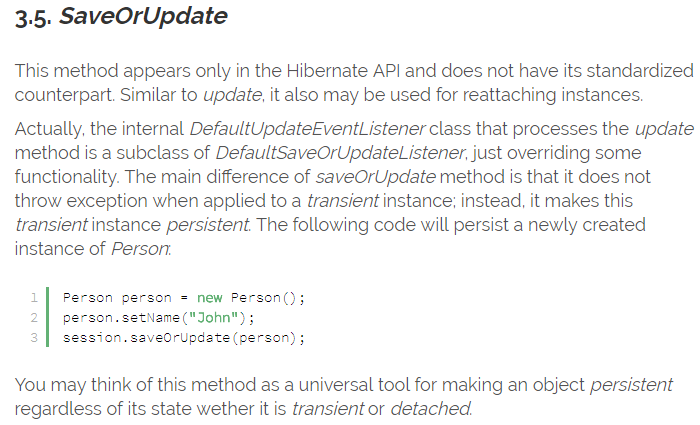


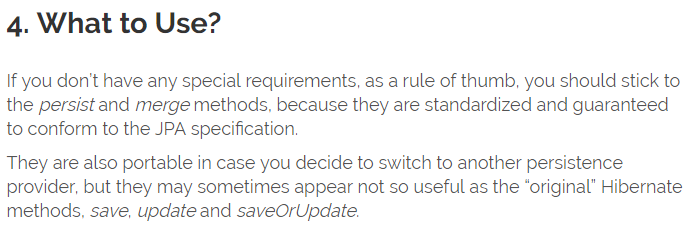






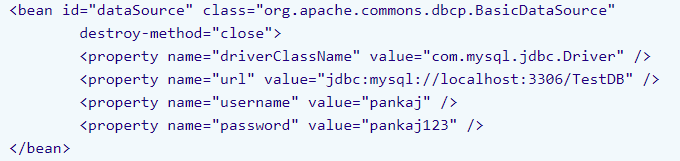


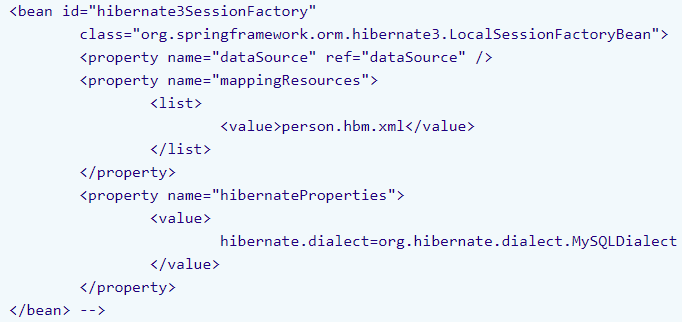




# Configuring hibernate with spring

* Step 1: Add the required dependency in the pom.xml file describing the version of Hibernate used.
* Step 2: The spring configuration file needs to be added with bean providing details on the data base URL, driver class, data base user name and password.





* Step 3: Is to create a DAO class having “SessionFactory” object as a field and can house methods to create, modify and update information on the data base.