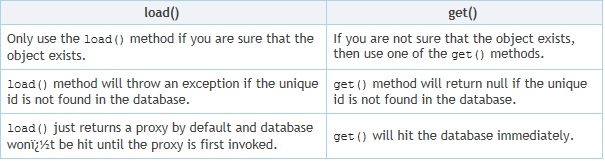
Hibernate

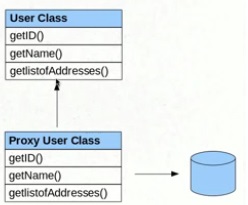
session.load()

* It will always return a “**proxy**” (Hibernate term) without hitting the database. In Hibernate, proxy is an object with the given identifier value, its properties are not initialized yet, it just look like a temporary fake object.
* If no row found , it will throws an **ObjectNotFoundException**.

2. session.get()

* It always **hit the database** and return the real object, an object that represent the database row, not proxy.
* If no row found , it return **null**.



**Proxy** means, hibernate will prepare some fake object with given identifier value in the memory without hitting a database.  
[](https://i.stack.imgur.com/5HteU.jpg)

**For Example:**  
If we call session.load(Student.class,new Integer(107));

hibernate will create one fake Student object [row] in the memory with id 107, but remaining properties of Student class will not even be initialized.

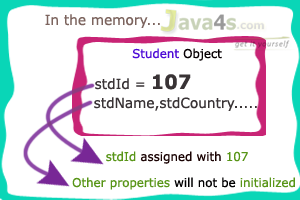
**Few Points About Hibernate get() & load()**

* Both are from Session interface, and we will call them as session.get() & session.load()
* Both will be use for retrieving the object (a row) from the database

Then what’s the difference them ? lets start with load() and then get() method.

Consider a Student class having 3 properties *stdId, stdName, stdCountry*

**1. session.load()**

* When you call session.load() method, it will always return a “proxy” object,  whats the meaning of proxy object ?
* Proxy means, hibernate will prepare some fake object with given *identifier value* in the memory without hitting the database, for example if we call *session.load(Student.class,new Integer(107));*  > hibernate will create one fake Student object [row] in the memory with id 107, but remaining properties of Student class will not even be initialized, observe this graphical representation…  
  
* It will hit the database only when we try to retrieve the other properties of Student object i mean stdName, stdCountry.  If we call s2.getStdName() then hibernate will hit the database and search the row with student id 107 and retrieve the values, if object [row] not found in the database it will throws ObjectNotFoundException.,

Let me explain each point by taking an example

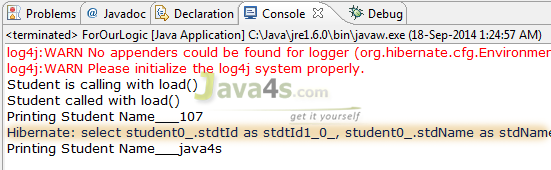
**Example of session.load()**

index.jsp:



|  |  |
| --- | --- |
| 1  2  3  4  5 | System.out.println("Student is calling with load()");  s2 =(Student) session.load(Student.class,new Integer(107));  System.out.println("Student called with load()");  System.out.println("Printing Student Name\_\_\_"+s2.getStdtId());  System.out.println("Printing Student Name\_\_\_"+s2.getStdName()); |

**Output**

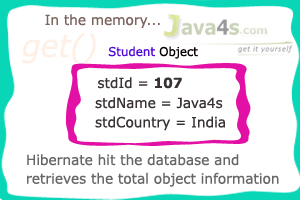


**Explanation:**

In index.jsp > line number 4,  i have called **s2.getStdtId()** and hibernate simply printed 107 [at Output > line number 3] without creating any database query why ? because as i have explained hibernate will prepare some fake object with given *identifier value* in the memory without hitting the database.  So when we call load() method (at index.jsp > line number 2 ) with 107 value, hibernate will create a fake object with 107 as identifier value, i mean temporarily it will consider the student id as 107.  If you observe the output, hibernate was generated the database query (at Output > line number 4 )  when we called **s2.getStdName()**; ( index.jsp > line number 5 ).

That’s it, so finally we came to know that session.load() will hit the database only when we start retrieving the object (row) values.

**2. session.get()**

* When you call session.get() method, it will hit the database immediately and returns the original object  
  
* If the row is not available in the database, it returns null

2) Hibernate cache

Hibernate provides 3 types of caching. 

#### 1. Session Cache

    The session cache caches objects within the current session. It is enabled by default in Hibernate. Read more about  [Session Cache](http://blog.dynatrace.com/2009/03/24/understanding-caching-in-hibernate-part-three-the-second-level-cache/) . Objects in the session cache reside in the same memory location. 

#### 2. Second Level Cache

     The second level cache is responsible for caching objects across sessions. When this is turned on, objects will first be searched in the cache and if they are not found, a database query will be fired. Read here on how to implement  [Second Level Cache](http://blog.dynatrace.com/2009/03/24/understanding-caching-in-hibernate-part-three-the-second-level-cache/). Second level cache will be used when the objects are loaded using their primary key. This includes fetching of associations. Second level cache objects are constructed and reside in different memory locations. 

#### 3. Query Cache

Query Cache is used to cache the results of a query. Read here on how to implement [query cache](http://blog.dynatrace.com/2009/02/16/understanding-caching-in-hibernate-part-two-the-query-cache/). When the query cache is turned on, the results of the query are stored against the combination query and parameters. Every time the query is fired the cache manager  checks for the combination of parameters and query. If the results are found in the cache, they are returned, otherwise a database transaction is initiated.  As you can see, it is not a good idea to cache a query if it has a number of parameters, because then a single parameter can take a number of values. For each of these combinations the results are stored in the memory. This  can lead to extensive memory usage.   
  
Finally, here is a list of good articles written on this topic,    
  
1. [Speed Up Your Hibernate Applications with Second-Level Caching](http://www.devx.com/dbzone/Article/29685/0/page/1)   
2. [Hibernate: Truly Understanding the Second-Level and Query Caches](http://www.javalobby.org/java/forums/t48846.html)   
3.  [EhCache Integration with Spring and Hibernate. Step by Step Tutorial](http://eiconsulting.blogspot.in/2011/10/ehcache-implementation-in-spring.html)   
4. [Configuring Ehcache with hibernate](http://www.ehcache.org/documentation/user-guide/hibernate)