JPA with Entity Listeners and Callback Methods

Hello Everyone!

Today I’ll talk about **Entity Listeners** and **Callback Methods** in JPA Spec!

**Callback Methods and Entity Listener in JPA?**

Imagine that you’ll save a new blog post in you database using JPA (with Hibernate for example). Sometimes we need to change or put new data in our record before or after it is saved, right? In most cases we need to do that after certain events, like after save our data, before save our data, before delete some record or before update a data.

How can we solve our problem? Maybe most of you like to use **Triggers** for that. But we have some problems with Triggers. I worked for a company that has used 4 kinds of databases and it was not fun rewrite triggers for all databases. Another problem is you can’t test your code! I like to use **TDD** in my projects, so I like to test all my business code.

To solve this problem JPA creates a mode that our code can react to certain events, like we talked above. JPA created **Callback Methods**.  
Let’s suppose we have the Post class bellow:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Entity  public class Post {       @Id     private Long id;     private String title;     private String text;     private Date date; //Date? Ok, terrible! Just an example } |

Now we need to persist our post:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class PostRepository {        public void save(Post post) {         entityManager.persist(post);      }    } |

Great! But we need to save our post with a specific date **before** save it. We can use the @PrePersist annotation:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Public class Post {        @PrePersist      private void changeDate() {         this.date = newDate();      }    } |

Nice! Now we have a **Callback Method** called **changeDate** that will be executed before persist our object! We have another types of Callback Methods and you can see bellow:

@PrePersist and @PostPersist  
@PreUpdate and @PostUpdate  
@PreRemove and @PostRemove  
@PostLoad

**If I have the same callback code for many Entities**

Ok! But what if we need to call a Log method (Callback Method) in all of our classes? We will copy the entire code and paste it? It’s not sounds good! For that we can create a specific class with that log code and use it in our classes. We can have our Log class like bellow using @PrePersist:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class LogListener {        @PrePersist      private void log(Object object) {          System.out.println("Your log code here");      }    } |

Did you note that we received an object? Yes, we receive the object that has being saved at the moment! After that we need to use our LogListener class using the @EntityListeners annotation:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @EntityListeners(LogListener.class)  publicclassPost {        @Id      privateLong id;        privateString title;        privateString text;        privateDate date; //Date? Ok, terrible! Just an example  } |

Awesome!

**Callback Method curiosities**

The @EntityListeners allow us to use multiple classes like @EntityListeners({Log.class, Audit.class, YourClass.class}). Nice!

Also, one method can be annotated with more than one **Callback annotation:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class LogListener {        @PrePersist      @PreUpdate      @PostRemove      privatevoidlog(Object object) {          System.out.println("Your log code here");      }    } |

**Can I have two methods annotated with the same callback annotation?**  
No! You cannot have two methods being annotated by the same callback annotation. See the next code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | publicclassLogListener {        @PreUpdate      @PostRemove      privatevoidlog(Object object) {          System.out.println("Your log code here");      }      @PreUpdate      privatevoidlogAgain(Object object) {      }  } |

The code above is not possible because we have two @PreUpdate annotation and an exception will be throws in our face : )

**Can I have two Listener Classes with methods with the same callback annotation?**

Sure! You can see that now:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | publicclassLogListener {        @PrePersist//Log class with PrePersist    privatevoidlog(Object object) {          System.out.println("Your log code here");      }    } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | publicclassAuditListener {        @PrePersist//Audit class with PrePersist    privatevoidaudit(Object object) {          System.out.println("Your audit code here");      }  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @EntityListeners({LogListener.class, AuditListener.class}) //using both Listeners  publicclassPost {        @Id      privateLong id;        privateString title;        privateString text;        privateDate date;    } |

After you persist you object you will see the follow:

|  |  |
| --- | --- |
| 1  2 | Your log code here  Your audit code here |

Yes, they will be printed following the sequence of the classes in the @EntityListeners.

**I love XML! Can I use it?**

Sure! You can indicate your callback methods in your **persistence.xml**

|  |  |
| --- | --- |
| 1  2  3  4  5 | <entity-listeners>      <entity-listenerclass="mypackage.LogListener">           <pre-persistmethod-name="log"/>      </entity-listener>  </entity-listeners> |

Mapping of java data type to DB2 data type

| **Java data type** | **Database data type** |
| --- | --- |
| short, java.lang.Short | SMALLINT |
| boolean[1](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvseta), byte[1](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvseta), java.lang.Boolean, java.lang.Byte | SMALLINT |
| int, java.lang.Integer | INTEGER |
| long, java.lang.Long | BIGINT[12](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetx) |
| java.math.BigInteger | BIGINT[11](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetu) |
| java.math.BigInteger | CHAR(*n*)[11](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetu),[5](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsete) |
| float, java.lang.Float | REAL |
| double, java.lang.Double | DOUBLE |
| java.math.BigDecimal | DECIMAL(*p*,*s*)[2](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetd) |
| java.math.BigDecimal | DECFLOAT(*n*)[3](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetm),[4](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetdf) |
| java.lang.String | CHAR(*n*)[5](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsete) |
| java.lang.String | GRAPHIC(*m*)[6](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetf) |
| java.lang.String | VARCHAR(*n*)[7](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetg) |
| java.lang.String | VARGRAPHIC(*m*)[8](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvseth) |
| java.lang.String | CLOB[9](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetl) |
| java.lang.String | XML[10](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetz) |
| byte[] | CHAR(*n*) FOR BIT DATA[5](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsete) |
| byte[] | VARCHAR(*n*) FOR BIT DATA[7](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetg) |
| byte[] | BINARY(*n*)[5](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsete), [13](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetw) |
| byte[] | VARBINARY(*n*)[7](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetg), [13](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetw) |
| byte[] | BLOB[9](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetl) |
| byte[] | ROWID |
| byte[] | XML[10](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetz) |
| java.sql.Blob | BLOB |
| java.sql.Blob | XML[10](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetz) |
| java.sql.Clob | CLOB |
| java.sql.Clob | DBCLOB[9](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetl) |
| java.sql.Clob | XML[10](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.5.0/com.ibm.db2.luw.apdv.java.doc/src/tpc/imjcc_rjvjdata.html#imjcc_rjvjdata__jvsetz) |
| java.sql.Date | DATE |
| java.sql.Time | TIME |
| java.sql.Timestamp | TIMESTAMP |
| java.io.ByteArrayInputStream |  |

| *Table 179. Mappings of DB2 data types to Java data types for retrieving data from DB2 tables* | | |
| --- | --- | --- |
| **SQL data type** | **Recommended Java data type or Java object type** | **Other supported Java data types** |
| SMALLINT | short | byte, int, long, float, double, java.math.BigDecimal, boolean, java.lang.String |
| INTEGER | int | short, byte, long, float, double, java.math.BigDecimal, boolean, java.lang.String |
| BIGINT | long | int, short, byte, float, double, java.math.BigDecimal, boolean, java.lang.String |
| DECFLOAT(*n*)[2](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.1.0/com.ibm.db2.udb.apdv.java.doc/doc/rjvjdata.htm#jvgeta),[3](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.1.0/com.ibm.db2.udb.apdv.java.doc/doc/rjvjdata.htm#jvgetdf) | java.math.BigDecimal | long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.lang.String |
| REAL | float | long, int, short, byte, double, java.math.BigDecimal, boolean, java.lang.String |
| DOUBLE | double | long, int, short, byte, float, java.math.BigDecimal, boolean, java.lang.String |
| CHAR(*n*) | java.lang.String | long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader |
| VARCHAR(*n*) | java.lang.String | long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader |
| CHAR(*n*) FOR BIT DATA | byte[] | java.lang.String, java.io.InputStream, java.io.Reader |
| VARCHAR(*n*) FOR BIT DATA | byte[] | java.lang.String, java.io.InputStream, java.io.Reader |
| BINARY(*n*) | byte[] | None |
| VARBINARY(*n*) | byte[] | None |
| GRAPHIC(*m*) | java.lang.String | long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader |
| VARGRAPHIC(*m*) | java.lang.String | long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader |
| CLOB(*n*) | java.sql.Clob | java.lang.String |
| BLOB(*n*) | java.sql.Blob | byte[][4](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.1.0/com.ibm.db2.udb.apdv.java.doc/doc/rjvjdata.htm#jvgetd) |
| DBCLOB(*m*) | No exact equivalent. Use java.sql.Clob. |  |
| ROWID | com.ibm.db2.jcc.DB2RowID | byte[] |
| XML | com.ibm.db2.jcc.DB2Xml | byte[], java.lang.String, java.io.InputStream, java.io.Reader |
| DATE | java.sql.Date | java.sql.String, java.sql.Timestamp |
| TIME | java.sql.Time | java.sql.String, java.sql.Timestamp |
| TIMESTAMP | java.sql.Timestamp | java.sql.String, java.sql.Date, java.sql.Time, java.sql.Timestamp |
| DATALINK | java.net.URL[5](https://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSEPGG_9.1.0/com.ibm.db2.udb.apdv.java.doc/doc/rjvjdata.htm#jvgetc) |  |
| **Notes:** | | |

Stored Procedure in JPA

https://en.wikibooks.org/wiki/Java\_Persistence/Advanced\_Topics

<http://www.thoughts-on-java.org/call-stored-procedures-jpa-part-2/>

<http://examples.javacodegeeks.com/enterprise-java/jpa/jpa-sql-stored-procedure-example/>

<http://dreamand.me/java/java-jee7-jpa-stored-procedure-example/>

The use of named parameters is not defined for native queries. From the JPA specification (section**3.6.3 Named Parameters**):

Named parameters follow the rules for identifiers defined in Section 4.4.1. The use of named parameters applies to the Java Persistence query language, and is not defined for native queries.**Only positional parameter binding may be portably used for native queries**.

So try the following instead:

StringqueryString = "select \* from Cell c where ST\_DWithin(c.shape, SetSRID(ST\_GeomFromEWKT('POINT(?1 ?2)'),4326), 0.1)";

Queryquery = Cell.em().createNativeQuery(queryString, Cell.class);

query.setParameter(1, longitude);

query.setParameter(2, latitude);

Note that in JPA >= 2.0 you **can use** named parameters in native queries.

<http://stackoverflow.com/questions/6277807/jpa-passing-list-to-in-clause-in-named-native-query>

@NamedNativeQuery(

name="User.findByUserIdList",

query="select u.user\_id, u.dob, u.name, u.sex, u.address from user u "+

"whereu.user\_id in (?userIdList)"

)

ist<Object[]>userList= em.createNamedQuery("User.findByUserIdList").setParameter("userIdList", list).getResultList();

http://java.boot.by/scbcd5-guide/ch08s03.html