NewbieGuideToOJB

This page describes how to use the persistance broker simple and effectively. Regardless of the object-type, there are always several database operations (set, get and delete) present.

The following generic methods could be part of your data access object or object manager that handles these database operations (and would work for all your classes).

To retrieve objects from the database:

```
public Object getObject(Object obj, String key, Object value) {
    Criteria criteria = null;
    Object result = null;
    PersistenceBroker broker = null;
    QueryByCriteria query = null;

    if(obj == null)
        return result;
    try {
        broker = PersistenceBrokerFactory.defaultPersistenceBroker();
        // Create criteria.
        criteria = new Criteria();
        criteria.addEqualTo(key, value);                                // Refine criteria
        // Create Query
        query = new QueryByCriteria(obj.getClass(), criteria);
        result = broker.getObjectByQuery(query);
    } catch (Exception e) {
        System.out.println("Error in getObject: " + e.getMessage());
    } finally {
        if(broker != null)
            broker.close();                                              // Release broker instance to the
broker-pool
    }
    return result;
}
```

Note that getObject can easily be 'extended' with more complicated matter when there are more constraints, in that case I'd suggest changing the above into getObject(Object obj, HashMap map) and add these to the criteria.

To store objects to the database:

```
public Object setObject(Object obj) {
    PersistenceBroker broker = null;

    try {
        if(obj == null)
            return obj;
        broker = PersistenceBrokerFactory.defaultPersistenceBroker();
        broker.beginTransaction();
        broker.store(obj);                                              // Store the object (make persistent)
        broker.commitTransaction();
        // At this point the object is updated with an auto-generated key, if any.
    } catch (Exception e) {
        System.out.println("Error in setObject: " + e.getMessage());

        broker.abortTransaction();                                      // Roll back
    } finally {
        if(broker != null)
            broker.close();                                              // Release broker instance to the
broker-pool
    }
    return obj;
}
```

Note that OJB makes no difference between an 'insert' and an 'update', this is all being handled by OJB. For that reason the setObject method returns the possible updated object, namely, it might have an id!

To delete objects from the database:

```
public boolean deleteObject(Object obj) {
    Collection pool = null;
    PersistenceBroker broker = null;
    boolean result = false;

    try {
        if(obj == null)
            return result;
        broker = PersistenceBrokerFactory.defaultPersistenceBroker();
        broker.beginTransaction();
        broker.delete(obj);                                                  // Delete the object
        broker.commitTransaction();
        result = true;
    } catch (Exception e) {
        System.out.println("Error in deleteObject: " + e.getMessage());
        broker.abortTransaction();                                          // Roll back
    } finally {
        if(broker != null)
            broker.close();                                                  // Release broker instance to the
broker-pool
    }
    return result;
}
```

To support this example I'll give a snippet of the repository.xml that uses a MySQL database with auto-increment on a primary key. Or simply put, the setObject mentioned above will return the assigned key by MySQL.

```
<descriptor-repository version="1.0" isolation-level="read-uncommitted">
    <jdbc-connection-descriptor jcd-alias="myDatabase" default-connection="true" platform="MySQL">
        <sequence-manager className="org.apache.ojb.broker.util.sequence.SequenceManagerNativeImpl">
            <attribute attribute-name="grabSize" attribute-value="10"/>
        </sequence-manager>
    </jdbc-connection-descriptor>

    <class-descriptor class="com.sample.Article_types" table="ARTICLE_TYPES">
        <field-descriptor name="id" primarykey="true" nullable="false" default-fetch="true" column="ID" jdbc-
type="INTEGER"/>
        <field-descriptor name="description" default-fetch="true" column="DESCRIPTION" jdbc-type="VARCHAR"/>
        <field-descriptor name="name" default-fetch="true" column="NAME" jdbc-type="VARCHAR"/>
    </class-descriptor>
    ...
</descriptor-repository>
```