Technical notes Document

Jonathan Alvarsson

July 4, 2008

Contents

1 Persistent classes **3**
1.1 Deleting 3

2 Data Access Objects (DAOs) **3**

3 Managers **5**

4 Auditing **5**

5 GUI **7**
5.1 Viewing a plate 7

6 Problems and solutions **7**
6.1 Regarding contains-method on HashSet's 7
6.2 Regarding simultaneous object editing 7

1 Persistent classes

The persistent classes, the classes that can be written and loaded from the database can be found in the `net.bioclipse.lis.pojos` package (pojo stands for plain old java object which is more or less how they are implemented). These classes contains a couple of values, getters and setters for these values and implementations of `hashcode()`, `equals()`, and `deepCopy()` and that's about what they contain.

The persisten classes all extend `AbstractBaseObject` and in some cases also `AbstractAuditableObject`.

1.1 Deleting

The persistent classes can't really be deleted. They can be marked as deleted and thus later be restored. So when one of them is said to be deleted that means that it is marked as deleted. There is a somewhat special behavior related to their `delete()`-methods see table 1.

Persistent class	Behavior
CellOrigin	Can only be deleted when no CellSample references it
DrugOrigin	Can only be deleted when no DrugSample references it
Annotation	Also deletes all related AbstractAnnotationInstances
Experiment	Also deletes all related Plates
Instrument	Can only be deleted when no Measurement references it
InstrumentType	Can only be deleted when no Instrument references it
LayoutWell	Also deletes all related LayoutMarkers
MasterPlate	Alse deletes all related Wells, PlateFunctions and AnnotationInstances
Measurement	Also deletes all related Results
PlateLayout	Also deletes all related LayoutWells, PlateFunctions and AnnotationInstances
Plate	Also deletes all related Wells, PlateFunctions, and AnnotationInstances
Project	Also deletes all related Experiments
SampleContainer	Also deletes all related samples and the related Worklist
Well	Also deletes related SampleMarkers, WellFunctions and the related SampleContainer
WorkList	Also deletes all related Operations.

Table 1: The behavior of some special persistent classes regarding delete

2 Data Access Objects (DAOs)

The data access objects handles the storing and retreaving of objects from the database. They are all instances of the class `GenericDAO` (seen in figure 2) which use *Java 5* generics and *Spring* AOP introductions to deliver the correct object without need for casting and to connect methods declared in an interface to *Hibernate* queries. It also uses the *Spring* class `HibernateDaoSupport`[1] as a base because it more or less already contains all the functionality.

Creating a new DAO consists of the following steps:

- Write an interface for the class where methods that should refer to a *Hibernate* query begins with the word "find".
- Write a *Spring* bean for the class in *applicationContext.xml*.
- Write the *Hibernate* query in namedQueries.hbm.xml

[1] `org.springframework.orm.hibernate3.support.HibernateDaoSupport`

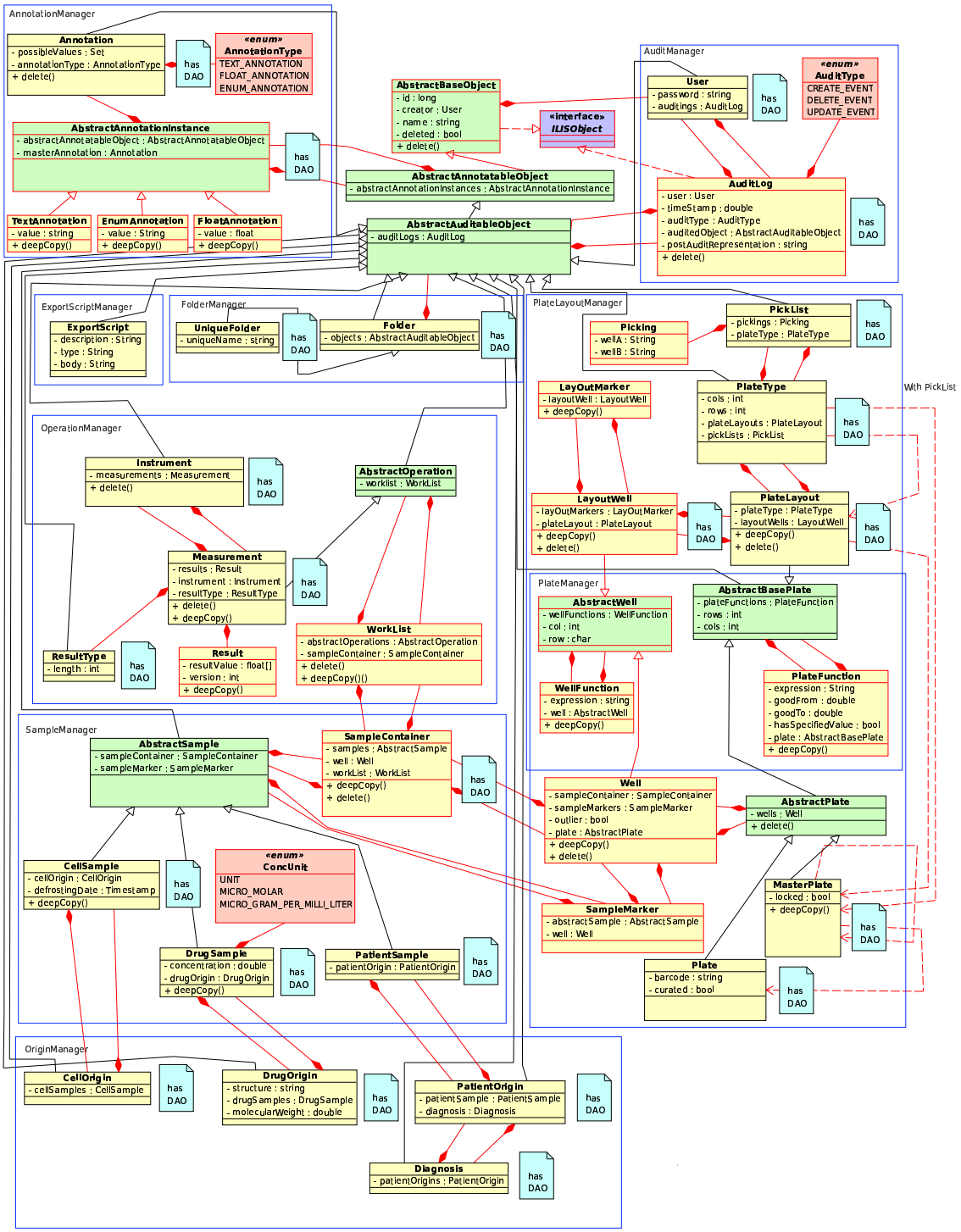

Figure 1: A class diagram of the persistent classes. All classes extend `AbstractBaseObject` (except AuditLog) and some (with black frame in figure) also `AbstractAuditableObject`. The blue boxes show which classes are handled by which managers.

3 Managers

The manager classes contains the methods that the GUI will run. They contain the needed DAOs and will forward requests to the right class so that they will be performed.

4 Auditing

When an auditable object is created, edited or marked as deleted an auditlog is created. When a non-auditable object that is a part of an auditable object is created, for example a PlateFunction, this happening will be audited as an editing of the auditable object in this case a Plate, MasterPlate or PlateLayout. Table 2 contains a list of all the methods that needs to call the AuditService. The test AuditInvokedTest checks that all of these calls takes place and the test TransactionTest checks that they are in a transaction meaning that either both the auditing and the actual operation takes place or else neither.

Manager	**Method**
AnnotationManager	newAnnotation annotate delete AbstractAnnotationInstance delete Annotation edit AbstractAnnotationInstance edit Annotation
AuditManager	newUser edit User delete User
OperationManager	newInstrument newResultType newMeasurement edit Measurement edit Instrument edit ResultType delete Measurement delete Instrument delete ResultType addResult
OriginManager	createDrugOrigin createCellOrigin edit DrugOrigin edit CellOrigin delete DrugOrigin delete CellOrigin
PlateManager	createPlate createMasterPlate createWellFunction createPlateFunction (with good between values) createPlateFunction (without good between values)
PlateLayoutManager	createPlateType createPlateLayout createWellFunction createPlateFunction (with good between values) createPlateFunction (without good between values)
ProjectManager	createProject createExperiment
SampleManager	addNewCellSampleToContainer addNewDrugSampleToContainer

Table 2: All methods needing to call the AuditService.audit method.

```
package net.bioclipse.lis.genericDAO;

import java.lang.reflect.Method;
import java.util.List;
import net.bioclipse.lis.pojos.IAbstractBaseObject;
import org.hibernate.Query;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

/**
 * This class is the basis for the daos. It uses generics to be able to return
 * daos without need for casting. Spring uses it as base to build from when
 * implementing the dao interfaces. It also contains some methods for running
 * the correspondingly named Hibernate query when methods defined in the DAO
 * interface are called.
 *
 * @author jonathan
 *
 * @param <T> The type of the persistent class the dao object should work with.
 */
public class GenericDAO<T> extends HibernateDaoSupport
                           implements IGenericDAO<T>, FinderExecutor {

        private Class<T> type;

        public GenericDAO(Class<T> type){
                this.type = type;
        }

        public void delete(long id) {
            IAbstractBaseObject obj = (IAbstractBaseObject)getSession().get(type, id);
            obj.delete();
            getSession().save(obj);
        }

        public T getById(long id) {
                return (T)getSession().get(type, id);
        }

        public void save(T instance) {
            getSession().saveOrUpdate(instance);
        }

        public List<T> executeFinder(Method method, final Object[] queryArgs) {
             String queryName         = queryNameFromMethod(method);
             Query namedQuery         = getSession().getNamedQuery(queryName);

             for( int i = 0;
                  i < (queryArgs == null? 0 : queryArgs.length);
                  i++
                ) {
                    Object arg = queryArgs[i];
                    namedQuery.setParameter(i, arg);
              }
              return (List<T>) namedQuery.list();
        }

        public String queryNameFromMethod(Method finderMethod) {
                return type.getSimpleName() + "." + finderMethod.getName();
        }
}
```


Figure 2: The `GenericDAO` class

5 GUI

5.1 Viewing a plate

In a couple of different places in the GUI something that shows data related to a plate is needed. Table 3 lists them and what are needed of them.

Needed to show	Seems to be working on	Can work on
Markers	PlateLayout	PlateLayout
Which calculation functions are defined for a certain well	PlateLayout	PlateLayout
Markers and concentrations	MasterPlate	AbstractPlate
Markers and concentrations	Plate	AbstractPlate
Results as numbers	Plate	Plate
Results as colours	Plate	Plate

Table 3: The different situations calling for some kind of plateviewer in the gui.

6 Problems and solutions

6.1 Regarding contains-method on HashSet's

During testing the following strange thing happened. A set containing an object returned false when contains was called with a reference to the actual same object. After some consideration the following cause was realised. The object was created with a hashcode method using some properties of the object. Then those properties of the object where changed and finally the contains method was called with the changed object thus causing the contains method to perform a lookup in the hash "bucket" corresponding to the new hashcode and of course not finding the object. The exact effects of this phenomenon are not yet realised but it is a good thing to know. The current solutions in the test case consists of either making all changes before the object is put in the hashSet or saving the object to the database followed by reloading the object containing the Set from the database and thus getting a new corrrect one. This phenomenon is important to have in mind when using HashSet's.

6.2 Regarding simultaneous object editing

If the original object differs from in the database when the edited object is saved a dialog should inform the user who has edited and what has changed and allow him to force overwrite or reload and redo the changes manually.