Notes on the implementation of the Clubs system

The source code of the Clubs example system is on uml.cs.uga.edu in the directory `~kochut/csx050/clubs`.

1. Entity Package

The entity classes are represented as a collection of three classes for `Person`, `Club`, and `Membership`.

1.1 Entity interface package (`edu.uga.clubs.entity`)

This package contains three interfaces for the three UML entity classes of `Person`, `Club`, and `Membership`. All three extend the `Persistable` interface defined in the persistence interface package (`edu.uga.clubs.persistence`). Therefore, all classes will be persistable in the database, as it is expected for all entity classes.

An entity class that participates in an association with another entity class and the other entity class endpoint had multiplicity 0 or 1 has a getter and setter method to return and set the other object instance. Other associations in which the entity class participates with many-type multiplicities are handled by dedicated operations in the `ObjectLayer` interface (see below).

1.2 Entity implementation package (`edu.uga.clubs.entity.impl`)

This package has 3 classes implementing the entity interfaces from the `edu.uga.clubs.entity` package. The three classes extend the `Persistent` class from the `edu.uga.clubs.persistence.impl` package, which implements the `Persistable` interface. As a consequence, the 3 classes need only implement the instance variable for the UML-defined attributes, constructors, as well as getters/setters for the attributes.

2. Persistence Layer Subsystem

The Persistence Layer subsystem has been implemented by "hand coding" the Object-Relational mapping. The MySQL schema of the clubs system is in the `db` directory, and it defines the 3 tables which implement the three classes: the `Person` class, the `Club` class, and the `Membership` association class. Each table has an automatically generated primary key (id), and a collection of attributes (columns) to represent the attributes defined in the Clubs UML data model.

2.1 Persistence interface package (`edu.uga.clubs.persistence`)

2.1.1 Persistence Layer API

The interface to the Persistence Layer subsystem is provided as the `PersistenceLayer` Java interface, which defines all of the necessary methods to store/restore/delete all entity classes and associations.

Each entity class has three methods: save, restore, and delete.

Each association between classes A and B has four methods: `save`, `restore( A )`, `restore( B )`, and `delete`, which are used to store, restore, and delete persistent links between persistent objects. The `restore(A aObject)` and `restore(B bObject)` methods return the linked object on the other side of the association (they traverse the association). For the many side, an Iterator of linked objects is returned (see below).

2.1.2 Persistable

The `Persistable` interface provides the necessary methods for getting and setting the persistent identifier (the primary key). It also provides a `boolean` method to check if an entity object has been already stored in the database. A value of -1 indicates that the entity object has not yet been stored in the database, and a non-negative value indicates that the object has already been stored in the database. However, please remember that if a previously stored entity object (with id >= 0, and for which `isPersistent()` is true) has been modified by some use case, it needs to be stored in the database again (the persistent object in the database must be updated).

2.2 Persistence implementation package (`edu.uga.clubs.persistence.impl`)

2.2.1 The Persistent abstract class

This abstract class implements the `Persistable` interface. It has a single instance variable for storing the persistent identifier (database primary key) and implements the methods defined in the `Persistable` interface. Furthermore, this class is used as the base class for all classes implementing the entity interfaces (in the package `edu.uga.clubs.entity.impl`.

2.2.2 PersistenceLayer implementation class

This class implements all methods defined in the `PersistenceLayer` interface. The implemented methods rely on the entity class-specific manager and iterator classes for handling of storage and retrieval of all entity class instances.

2.2.3 Manager classes

One manager class is implemented for each entity class and it is responsible for transferring Java objects (proxies) to and from the database by executing suitable SQL statements. It also deletes no longer needed objects.

Each manager class provides a constructor, a save method (for storing new and updating already existing objects), restore method (for retrieving persistent objects and returning iterators for them), and delete method for deleting persistent objects from the database.

A restore method accepts a model (think of it as am example) object instance to indicate what should be retrieved from the database. The following possibilities exist to supply a model object instance:

-- if a model instance is not provided (a null reference is supplied), then all persistent objects (i.e., all rows) from the database are retrieved (a suitable Iterator is created) and returned;

-- if a model instance provides an id (primary key), then the returned iterator will contain a single Java object instance

-- if a model instance does not provide an id, then all of the attributes of the model instance are used to construct the `WHERE` condition of the `SELECT SQL` statement and all rows with the specified values are retrieved; rows without the matching values are not retrieved. However, this method only works for object-values attributes (Strings, Dates, Integers, etc.) and not primitive data types, as it is not possible to specify that an `int`-valued attribute has not been provided (there is always some value of an `int` instance variable).

A restore method retrieves rows from a suitable table in the database according to the provided model object instance. A suitable `SELECT` statement is created and executed. NOTE, that if the entity class has an association to another class with a 1 or optional multiplicity, the `SELECT` statement must perform a join and retrieve the other (related) persistent object, as well.

After a successful `SELECT` statement, a corresponding Iterator object is created (see below), which delegates the `hasNext()` and `next()` methods to the suitable methods of the JDBC's `ResultSet` class.

Calling the `next()` method of the Iterator reads the values from the next row (from the `ResultSet`), creates an empty object instance of the entity class, and then sets the attributes to the values retrieved from the database row, thus completing the construction of the proxy object.

Each manager class implemented for an entity class provides the following methods:

a) a constructor
   it is used to initializes a manager object.

b) a save method
   it is used to save a given instance (the argument) to the database; if the object is not yet persistent, the method executes an SQL insert command, and if it is persistent, it performs an SQL update. After an initial insert, the value of the automatically generated primary key is retrieved and saved in the object instance.

c) a restore method
   it is used to return an iterator of proxy (Java) object instances, each representing a persistent object in the database. A corresponding iterator class instance is created and returned. The iterator provides the required `hasNext()` and `next()` methods, which can then be used to obtain the retrieved proxy (Java) objects.

d) a delete method
   it is used to delete a persistent object from the database. The argument object must be a persistent object (with the id defined).

2.2.4 Iterator classes

One iterator class is implemented for each entity class and it is responsible for iterating over all retrieved Java objects (proxies) retrieved from the database.

Each iterator class implements `java.util.Iterator` interface and provides a constructor, and implement the `hasNext()` and `next()` methods.

3. Object Layer Subsystem

The Object Layer subsystem provides create methods for all entity classes and all associations. It acts as a factory class for the entity classes by insulating the clients of the Object Layer from the specific constructors implemented for the entity classes.

Each create method is overloaded and the first version accepts all expected arguments, while the other one accepts no arguments and creates an uninitialized object instance. The argument-less create methods are used by the persistence layer to start the creation of proxy objects.

Please, note that if an entity class is associated to another class and that other class's multiplicity is 1, the create method has an argument representing the linked class object. This assures that the newly created object is always linked to the other side object (due to the 1 multiplicity).

3.1 Object layer interface package (`edu.uga.clubs.object`)

3.1.1 Object Layer API

The interface to the Object Layer subsystem is provided as the `ObjectLayer` Java interface, which defines all of the necessary methods to create/save/find/delete all entity classes and associations.

3.1.1.1 Handling of entity classes

Each entity class E has the following methods:

```
E createE()
```
which creates and returns an uninitialized object instance of class E

```
E createE( args )
```
which creates and returns an object instance of class E initialized to the provided arguments.

```
Iterator<E> findE( E modelE )
```

which returns an Iterator of all proxy (Java) objects, one for each persistent object matching the modelE object instance. The meaning of the model object instance has been explained in the section dedicated to the Persistence Layer subsystem.

`void store(E eObject)`
which stores (persists) a Java object in the data base

`void delete(E eObject)`
which deletes a Java object from the persistent data store.

3.1.1.2 Handling of associations

Each association A assoc B has the following methods (the method names are derived from concatenating the role names of the two endpoints):

`void create`*`EndpointAEndpointB`*`( A aObject, B bObject )`
which creates a link connecting the two argument objects.

`A create`*`EndpointAEndpointB`*`( B bObject )`
which returns an A object instance linked to the argument bObject (in case the A-side has multiplicity 1 or optional).

`Iterator<B> create`*`EndpointAEndpointB`*`( A aObject )`
which returns an iterator of B object instances linked to the argument aObject (in case the B-side has multiplicity of many).

`void delete`*`EndpointAEndpointB`*`( A aObject, B bObject )`
which deletes a link connecting the two argument objects. Note that for associations in which one side is mandatory (mulitplicity 1), then the delete method is not provided.

3.2 Object Layer implementation package (edu.uga.clubs.object.impl)

3.2.1 ObjectLayer implementation class

The ObjectLayer implementation class provides the implementation of the methods defined in the ObjectLayer API interface class.

The create methods for the entity classes are implemented using the calls to the corresponding constructors of the entity implementation classes in the `edu.uga.clubs.entity.impl` package.

The find, store, and delete methods for the entity classes are implemented by invoking (delegating to) the suitable Persistence Layer API methods.

The create, find, store, and delete methods for the associations are implemented by invoking (delegating to) the suitable Persistence Layer API methods.