```java
package com.ericburke.webguide.dao;

import java.io.Serializable;
import java.util.List;

/**
 * Generic Hibernate DAO interface.
 */
public interface TemplateDao <T, I extends Serializable> {
    I save(T obj);
    void saveOrUpdate(T obj);
    void update(T obj);
    void deleteAll();
    T getById(I id);
    void delete(T obj);
    void delete(I id);
    List<T> getAll();
}
```

```java
package com.ericburke.webguide.dao;

import com.ericburke.webguide.model.Category;

import java.util.List;

/**
 * Shows how to create a DAO specific to a persistent class, Category
 * in this case.
 * @author Eric M. Burke
 */
public interface CategoryDao extends TemplateDao<Category,Long> {
    List<Category> getRootCategories();
}
```

```java
package com.ericburke.webguide.business;

import com.ericburke.webguide.model.Category;
import com.ericburke.webguide.model.Resource;

import java.util.List;

/**
 * A business object that delegates to DAOs. Notice how no DAO interfaces are
 * exposed, nor are any Spring or Hibernate dependencies.
 *
 * @author Eric M. Burke
 */
public interface WebGuideBo {
    void save(Category newCategory);
    void save(Resource newResource);
    List<Resource> getUncategorizedResources();
    void update(Category category);
    void update(Resource resource);
    Category getCategory(Long categoryId);
    void dropAllData();
    void delete(Resource resource);
    void delete(Category category);
    List<Category> getRootCategories();
    void move(Resource resource,
              Category originalCategory,
              Category targetCategory);
    void move(Category categoryToMove, Category targetCategory);
    void copy(Resource res, Category targetCategory);
    void removeResourceFromCategory(Resource res, Category category);
}
```

```java
package com.ericburke.webguide.business;

import com.ericburke.webguide.dao.CategoryDao;
import com.ericburke.webguide.dao.ResourceDao;
import com.ericburke.webguide.model.Category;
import com.ericburke.webguide.model.Resource;

import java.util.List;

/**
 * Concrete implementation of a business object. The methods in this class must
 * operate inside of transactions, so we rely upon AOP for that.
 *
 * @author Eric M. Burke
 */
public class DefaultWebGuideBo implements WebGuideBo {
    private CategoryDao categoryDao;
    private ResourceDao resourceDao;

    // Spring will set the DAOs through dependency injection.
    public DefaultWebGuideBo(CategoryDao categoryDao, ResourceDao resourceDao) {
        this.categoryDao = categoryDao;
        this.resourceDao = resourceDao;
    }

    public void update(Category category) {
        categoryDao.update(category);
    }

    public void update(Resource resource) {
        resourceDao.update(resource);
    }

    public void save(Category category) {
        categoryDao.save(category);
    }

    public void save(Resource res) {
        resourceDao.save(res);
    }

    public void move(Resource resource,
                     Category originalCategory,
                     Category targetCategory) {
        if (originalCategory != null) {
            originalCategory.removeResource(resource);
            categoryDao.update(originalCategory);
        }
        if (targetCategory != null) {
            targetCategory.addResource(resource);
            categoryDao.update(targetCategory);
        }
    }

    public void removeResourceFromCategory(Resource res, Category category) {
        if (res == null || category == null) {
            return;
        }
        category.removeResource(res);
        categoryDao.update(category);
        resourceDao.update(res);
    }

    public void copy(Resource res, Category targetCategory) {
        if (targetCategory != null) {
            targetCategory.addResource(res);
            categoryDao.saveOrUpdate(targetCategory);
        }
    }

```

```
71      public void move(Category categoryToMove, Category targetCategory) {
72          if (categoryToMove == null || categoryToMove == targetCategory) {
73              return;
74          }
75          // avoid circular dependencies
76          if (targetCategory != null && targetCategory.hasAncestor(categoryToMove)) {
77              return;
78          }
79          Category origParent = categoryToMove.getParentCategory();
80          if (origParent != null) {
81              origParent.getChildCategories().remove(categoryToMove);
82              categoryDao.update(origParent);
83          }
84
85          if (targetCategory != null) {
86              targetCategory.getChildCategories().add(categoryToMove);
87              categoryDao.update(targetCategory);
88          }
89          categoryToMove.setParentCategory(targetCategory);
90          categoryDao.update(categoryToMove);
91      }
92
93      public List<Category> getRootCategories() {
94          return categoryDao.getRootCategories();
95      }
96
97      public List<Resource> getUncategorizedResources() {
98          return resourceDao.getUncategorizedResources();
99      }
100
101     public Category getCategory(Long categoryId) {
102         return categoryDao.getById(categoryId);
103     }
104
105     public void delete(Resource resource) {
106         resource.removeFromAllCategories();
107         resourceDao.delete(resource);
108     }
109
110     public void delete(Category category) {
111         category.prepareForDelete();
112         categoryDao.delete(category);
113     }
114
115     public void dropAllData() {
116         resourceDao.deleteAll();
117         categoryDao.deleteAll();
118     }
119 }
```

```
1   package com.ericburke.webguide.dao;
2
3   import net.sf.hibernate.Criteria;
4   import net.sf.hibernate.HibernateException;
5   import net.sf.hibernate.Session;
6   import net.sf.hibernate.expression.Criterion;
7   import org.springframework.orm.hibernate.HibernateCallback;
8   import org.springframework.orm.hibernate.HibernateTemplate;
9
10  import java.io.Serializable;
11  import java.util.ArrayList;
12  import java.util.List;
13
14  /**
15   * Hibernate DAO base class using Java 5 generics. This class uses Spring Framework
16   * in order to avoid writing try/catch logic for exception handling.
17   *
18   * @author Eric M. Burke
19   */
```

```
20  public class TemplateDaoImpl <T,I extends Serializable> implements TemplateDao<T, I> {
21      private Class persistentClass;
22      private HibernateTemplate template;
23
24      public TemplateDaoImpl(Class persistentClass,
25                             HibernateTemplate template) {
26          this.persistentClass = persistentClass;
27          this.template = template;
28      }
29
30      /**
31       * Store a new object in the database.
32       *
33       * @param obj the new object to make persistent.
34       * @return the generated id.
35       */
36      public I save(T obj) {
37          return (I) template.save(obj);
38      }
39
40      public void saveOrUpdate(T obj) {
41          template.saveOrUpdate(obj);
42      }
43
44      public void update(T obj) {
45          template.update(obj);
46      }
47
48      public void deleteAll() {
49          template.delete("from " + persistentClass.getName() + " x");
50      }
51
52      public T getById(I id) {
53          return (T) template.get(persistentClass, id);
54      }
55
56      public void delete(T obj) {
57          template.delete(obj);
58      }
59
60      public void delete(I id) {
61          delete(getById(id));
62      }
63
64      public List<T> getAll() {
65          return find("from " + persistentClass.getName() + " x");
66      }
67
68      protected List<T> find(String query) {
69          return new ArrayList<T>(template.find(query));
70      }
71
72      protected T getUnique(final Criterion... criterion) {
73          return (T) template.execute(new HibernateCallback() {
74              public Object doInHibernate(Session session) throws HibernateException {
75                  Criteria criteria = session.createCriteria(persistentClass);
76                  for (Criterion c : criterion) {
77                      criteria.add(c);
78                  }
79                  return (T) criteria.uniqueResult();
80              }
81          });
82      }
83
84      protected List<T> get(final Criterion... criterion) {
85          return (List<T>) template.execute(new HibernateCallback() {
86              public Object doInHibernate(Session session) throws HibernateException {
87                  Criteria criteria = session.createCriteria(persistentClass);
88                  for (Criterion c : criterion) {
89                      criteria.add(c);
```

```
                }
                return new ArrayList<T>(criteria.list());
            }
        });
    }
}
```

```
package com.ericburke.webguide.dao;

import com.ericburke.webguide.model.Category;
import net.sf.hibernate.expression.Expression;
import org.springframework.orm.hibernate.HibernateTemplate;
import java.util.List;

public class CategoryDaoImpl extends TemplateDaoImpl<Category, Long>
        implements CategoryDao {
    public CategoryDaoImpl(HibernateTemplate template) {
        super(Category.class, template);
    }

    public List<Category> getRootCategories() {
        return get(Expression.isNull("parentCategory"));
    }
}
```