Building dynamic data model

While building the data model using XML files is possible, it's advisable to use more dynamic data models in the production environment. Alternatively, ZKIB module provides a set of classes that ease interaction with data models based on JDBC databases and object-oriented layers like Hibernate or EJB. Additionally.

To make use of these dynamic data models, the developer must implement one or more instances of the DataNode object. These DataNode objects act as wrapping around the actual business bean, managing the persistence layer. In general, you should define a class for each XML file equivalent entity. Thus, in the previous example, the developer should implement classes for the root (my-data), title, country and city objects. In some cases you won't need to develop a new DataNode class as long as the data object needs not to be persisted. For those simple data objects, a SimpleDataNode object can be used.

Each class is responsible for encapsulating a business object, and for its serialization to the persistent repository, usually a database. Also, each class is responsible for retrieving child objects. The children population function is performed by the finder interface.

A DataNode object must implement the following methods:

Method	Attributes	Description
<constructor>	DataContext	Optionally, you must declare the "finders", by calling the method addFinder
doInsert		Insert a new object in the persistent storage
doUpdate		Update an object in the storage persitente
doDelete		Update an object in the storage persitente

It's noticeable, that doInsert, doDelete and doUpdate methods are always invoked in a transactional context, coordinated through the commit method of the DataSource. The DataNode derived objects can access the data of the underlying business object by calling the inherited method getInstance.

The Finder interface must implement the following methods:

Method	Result	Description

find	java.util.Collection	Retrieves a collection of business objects that are descendants of the current object. Note that this method should not construct DataNode objects that will wrap the business object later.
newInstance	Object	Creates a new business object, filling the default value of its attributes, depending on the context in which they are creating.

Thus, to implement the same countries data model, equivalent to the former XML file you should define the following business classes:

Country business class

```
package com.soffid.sample;

public class Country {
    private String name;
    private String abbreviation;
    public String getName () {return name;}
    public void setName (String name) {this.name = name;}
    getAbbreviation public String () {return abbreviation;}
    public void setAbbreviation (String abbreviation) {
        this.abbreviation = abbreviation;
    }
}
```

City business class

```
package com.soffid.sample;

public class City {
    private String name;
    public String getName () {return name;}
    public void setName (String name) {this.name = name;}
}
```

Title business class

```
package com.soffid.sample;
```

```
public class Title {
    private String name;
    public String getName () {return name;}
    public void setName (String name) {this.name = name;}
}
```

Now that we have the needed business classes, the next step is to create the a class that manages the root of the data tree. This class derives from SimpleDataNode because they do not allow the execution of the methods doInsert, doUpdate or doDelete. In its constructor defines two Finders, responsible for retrieving the list of countries (through a existing countryDAO class) and title (with code):

```
package com.soffid.sample;
import java.util.Vector;
import es.caib.zkib.datamodel.*;
import es.caib.zkib.datasource.*;
public class RootNode extends SimpleDataNode {
    public RootNode(DataContext ctx) {
        super(ctx);
        // Title
        addFinder("title",
                new Finder () {
                    public java.util.Collection find() throws Exception {
                        Title t = new Title ();
                        t.setName ( "World countries" );

                        Vector v = new Vector();
                        v.add (t);
                        return v;
                    };
                    public Object newInstance() throws Exception {
                        throw new UnsupportedOperationException();
                    }
                },
                SimpleDataNode.class);
        // Countries
        addFinder("country",
            new Finder () {
                public java.util.Collection find() throws Exception {
                    return CountryDAO.findAll ();
                };
```

```
            public Object newInstance() throws Exception {
                throw new UnsupportedOperationException();
            }
        },
        CountryNode.class);
    }
}
```

The CountryNode object class referenced in the previous class will wrap for Country objects got by DAO. This class is responsible to retrieve and instantiate City objects from the Country. It is derived from SimpleDataNode because no update, insert or delete of countries is allowed:

```
package com.soffid.sample;
import java.util.Vector;
import es.caib.zkib.datamodel.*;
import es.caib.zkib.datasource.*;

public class CountryNode extends DataNode {
    public CountryNode(DataContext ctx) {
        super(ctx);
        addFinder("city",
                new Finder () {
                    public java.util.Collection find() throws Exception {

                        Country c = (Country) getInstance();
                        return CityDAO.findByCountry (c.abbreviation);
                    };
                    public Object newInstance() throws Exception {
                        Country country = (Country) getInstance();

                        City city = new City();
                        city.setCountryAbbreviation (country.getAbbreviation());
                        return city;
                    }
                },
                CityNode.class);
    }
}
```

Finally, the CityNode class is responsible for managing City persistent objects. In this case there is no finder instance because the City has no children available:

```
package com.soffid.sample;
import java.util.Vector;
import es.caib.zkib.datamodel.*;
import es.caib.zkib.datasource.*;
public class CountryNode extends DataNode {
    public CountryNode(DataContext ctx) {
        super(ctx);
    }
    protected void doInsert() throws Exception {
        CityDAO.insert ((City) getInstance());
    }
    protected void doUpdate() throws Exception {
        CityDAO.update ((City) getInstance());
    }
    protected void doDelete() throws Exception {
        CityDAO.delete ((City) getInstance());
    }
}
```

In summary, we have had to generate four kinds of objects corresponding to the three types of element of the XML document:

- my-data: RootNode class derived from SimpleDataNode . It contains two finders, one for title, and one for country.
- title: is using SimpleDataNode class.
- country: CountryNode class derived from SimpleDataNode. It contains a finder that allows the instantiation of City objects.
- city: CityNode class derived from DataNode. Implements methods to persist City object. It has no finder.

Revision #1
Created 1 June 2021 10:18:02
Updated 1 June 2021 10:22:49