Tapestry5AnotherSelectWithObjects

NOTE: For the recommended, clustering-safe way to handle Selects with lists of objects, please see the Using Select with a List cookbook recipe.

Thiago H. de Paula Figueiredo: this approach is more complicated than needed. It has a serious flaw: You have to retrieve a list of objects when you need just one of them (wrong) or put a list in a session (very wrong). Just create one instance of OptionModelImpl for each of your options (objects) and pass them to a SelectModelImpl.

Davor Hrg: Thiago's comment should be considered, but the fact is that you should not use a select component in any situation where the number of elements is so big the performance or memory footprint becomes a problem (in that case the list is definitely too long for users to scroll through, and use AutoComplete instead). Also Thiago provides no example page of his so called simpler solution.

Tapestry5AnotherSelectWithObjects

We saw many ways of making a <SELECT> component filled with objects on other examples. This is an extension of the other methods, so if you want some more information, you may also see:

- Tapestry5HowtoSelectWithObjects
- Tapestry5 How to use the SELECT component
- Tapestry5DisplayableSelectionModel

Here, we will create an Annotation, a (not so) new feature from Java 1.5, widely known by Tapestry users, to suppress some steps from the other methods, like the instantiation of a SelectionModel and a ValueEncoder, or even create another component to do the job. Remember, this is an extension, so everything we saw in the other components are presented here. What I tried to do is an easier way to instantiate the component in the page class.

We will see 4 basic classes here:

- **GenericSelectionModel** - This is the model that the component will use to render the options for your Select component.
- **GenericValueEncoder** - This is the encoder that the component will use to get your selection back.
- **InjectSelectionModel** - This is the annotation, the only class that you will effectively use.
- **InjectSelectionModelWorker** - This is the class that will do all the dirty job for the annotation.

GenericSelectionModel.java

Put this class on any package visible to the *services* package.

```
import java.util.ArrayList;
import java.util.List;

import org.apache.tapestry5.OptionGroupModel;
import org.apache.tapestry5.OptionModel;
import org.apache.tapestry5.internal.OptionModelImpl;
import org.apache.tapestry5.ioc.services.PropertyAccess;
import org.apache.tapestry5.ioc.services.PropertyAdapter;
import org.apache.tapestry5.util.AbstractSelectModel;

public class GenericSelectionModel<T> extends AbstractSelectModel {

        private PropertyAdapter labelFieldAdapter = null;

        private List<T> list;

        public GenericSelectionModel(List<T> list, String labelField, PropertyAccess access) {
                if(list == null) list = new ArrayList<T>();
                if(labelField != null && !labelField.equalsIgnoreCase("null")){
                        if(!list.isEmpty()){
                                this.labelFieldAdapter = access.getAdapter(list.get(0).getClass()).
getPropertyAdapter(labelField);
                        }
                }
                this.list = list;
        }

        public List<OptionGroupModel> getOptionGroups() {
                return null;
        }

        public List<OptionModel> getOptions() {
                List<OptionModel> optionModelList = new ArrayList<OptionModel>();
                for (T obj : list) {
                        if (labelFieldAdapter == null) {
                                optionModelList.add(new OptionModelImpl(nvl(obj) + "", obj));
                        } else {
                                optionModelList.add(new OptionModelImpl(nvl(labelFieldAdapter.get(obj)), obj));
                        }
                }
                return optionModelList;
        }

        private String nvl(Object o) {
                if (o == null)
                        return "";
                else
                        return o.toString();
        }

}
```

GenericValueEncoder.java

Add it to the same GenericSelectionModel's package.

```
import java.util.ArrayList;
import java.util.List;

import org.apache.tapestry5.ValueEncoder;
import org.apache.tapestry5.ioc.services.PropertyAccess;
import org.apache.tapestry5.ioc.services.PropertyAdapter;

public class GenericValueEncoder<T> implements ValueEncoder<T> {

        private PropertyAdapter idFieldAdapter = null;
        private List<T> list;

        public GenericValueEncoder(List<T> list, String idField, PropertyAccess access) {
                if(list == null) list = new ArrayList<T>();
                if (idField != null && !idField.equalsIgnoreCase("null")){
                        if(!list.isEmpty()){
                                this.idFieldAdapter = access.getAdapter(list.get(0).getClass()).
getPropertyAdapter(idField);
                        }
                }
                this.list = list;
        }

        public String toClient(T obj) {
                if (idFieldAdapter == null) {
                        return nvl(obj);
                } else {
                        return nvl(idFieldAdapter.get(obj));
                }
        }

        public T toValue(String string) {
                if (idFieldAdapter == null) {
                        for (T obj : list) {
                                if (nvl(obj).equals(string)) return obj;
                        }
                } else {
                        for (T obj : list) {
                                if (nvl(idFieldAdapter.get(obj)).equals(string)) return obj;
                        }
                }
                return null;
        }

        private String nvl(Object o) {
                if (o == null)
                        return "";
                else
                        return o.toString();
        }
}
```

InjectSelectionModel.java

Now, here is the little annotation that you will use on your page class. Create an *annotations* package and put this class inside it. Further, we will see it's usage.

```
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Target(FIELD)
@Retention(RUNTIME)
@Documented
public @interface InjectSelectionModel {

        String labelField() default "null";
        String idField() default "null";

}
```

InjectSelectionModelWorker.java

Now, here comes the interesting part. This class do all the work and functionality that the annotation must do. It will add these things to your class:

- A **private PropertyAccess** variable that the model and the encoder will use on their constructors.
- A **getSelectionModel()** and **getValueEncoder()** methods to be used for the Select component.

So, this way, you don't need to add these lines on every page you create, with every Select you make. You just have to decorate the Select's list with this annotation and it will insert them for you.

Put this class on your *sevices* package:

```
import java.lang.reflect.Modifier;

import org.apache.tapestry5.ioc.internal.util.InternalUtils;
import org.apache.tapestry5.ioc.services.PropertyAccess;
import org.apache.tapestry5.ioc.util.BodyBuilder;
import org.apache.tapestry5.model.MutableComponentModel;
import org.apache.tapestry5.services.ClassTransformation;
import org.apache.tapestry5.services.ComponentClassTransformWorker;
import org.apache.tapestry5.services.TransformMethodSignature;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import br.sfiec.t5teste.annotations.InjectSelectionModel;
import br.sfiec.t5teste.data.GenericSelectionModel;
import br.sfiec.t5teste.data.GenericValueEncoder;

public class InjectSelectionModelWorker implements ComponentClassTransformWorker {

    final private Logger _logger = LoggerFactory.getLogger(InjectSelectionModelWorker.class);
    final private PropertyAccess _access;

        public InjectSelectionModelWorker(PropertyAccess propertyAccess) {
                _access = propertyAccess;
        }

        public void transform(ClassTransformation transformation, MutableComponentModel componentModel) {

                for (String fieldName : transformation.findFieldsWithAnnotation(InjectSelectionModel.class)) {
                        InjectSelectionModel annotation = transformation.getFieldAnnotation(fieldName,
InjectSelectionModel.class);

                        if (_logger.isDebugEnabled()){
                                _logger.debug("Creating selection model getter method for the field " +
fieldName);
                        }
                        String accessActualName = transformation.addField(Modifier.PRIVATE
                                , PropertyAccess.class.getName(), "_access");
                        transformation.injectField(accessActualName, _access);
```

```
                        addGetSelectionModelMethod(transformation, fieldName, annotation.labelField()
                                , accessActualName);

                                if (_logger.isDebugEnabled()){
                                        _logger.debug("Creating value encoder getter method for the field " +
fieldName);
                                }

                                addGetValueEncoderMethod(transformation, fieldName, annotation.idField(),
accessActualName);

                }

        }

        private void addGetSelectionModelMethod(ClassTransformation transformation, String fieldName, String
labelField, String accessName) {

                String methodName = "get" + InternalUtils.capitalize(InternalUtils
                        .stripMemberName(fieldName)) + "SelectionModel";

                String modelQualifiedName = (GenericSelectionModel.class).getName();
                TransformMethodSignature sig =
                        new TransformMethodSignature(Modifier.PUBLIC, modelQualifiedName
                        , methodName, null, null);

                BodyBuilder builder = new BodyBuilder();
                builder.begin();
                builder.addln("return new " + modelQualifiedName + "(" + fieldName
                        + ", \"" + labelField +"\", " + accessName + ");");
                builder.end();

                transformation.addMethod(sig, builder.toString());

        }

        private void addGetValueEncoderMethod(ClassTransformation transformation, String fieldName, String
idField, String accessName) {

                String methodName = "get" + InternalUtils.capitalize(InternalUtils
                        .stripMemberName(fieldName)) + "ValueEncoder";

                String encoderQualifiedName = (GenericValueEncoder.class).getName();
                TransformMethodSignature sig =
                        new TransformMethodSignature(Modifier.PUBLIC, encoderQualifiedName
                        , methodName, null, null);

                BodyBuilder builder = new BodyBuilder();
                builder.begin();
                String line = "return new " + encoderQualifiedName + "(" + fieldName + ",\"" + idField +"\", "
+ accessName + ");";
                builder.addln(line);
                builder.end();

                transformation.addMethod(sig, builder.toString());

        }

}
```

Bind them in your AppModule:

```
/**
     * Adds a number of standard component class transform workers:
     * <ul>
     * <li>InjectSelectionModel - generates the SelectionModel and ValueEncoder for a any marked list of
objects.</li>
     * </ul>
     */
    public static void contributeComponentClassTransformWorker
(OrderedConfiguration<ComponentClassTransformWorker> configuration, PropertyAccess propertyAccess)
    {
        configuration.add("InjectSelectionModel", new InjectSelectionModelWorker(propertyAccess), "after:
Inject*");
    }
```

So, let's use it:
First, declare a list and populate it. Put the annotation on the list and pass the **labelField** and **idField** attributes to it. Then, create the item to get the selection.

Here's the example:

```
public class Start {

        @InjectSelectionModel(labelField = "name", idField = "code")
        private List<Bank> _list = new ArrayList<Bank>();

        @Persist
        private Bank _item;

        public void onPrepare(){
                List<Bank> l = new ArrayList<Bank>();
                l.add(new Bank("Foo Bank 1", 1));
                l.add(new Bank("Foo Bank 2", 2));
                _list = l;
        }

        public Bank getItem() {
                return _item;
        }

        public void setItem(Bank item) {
                _item = item;
        }

        public List<Bank> getList() {
                return _list;
        }

        public void setList(List<Bank> list) {
                _list = list;
        }
}
```

On your template, use the core **Select** component and pass the parameters to it. The annotation will ever create the methods in these signatures: **get<AttributeName>SelectionModel()** and **get<AttributeName>ValueEncoder()**.

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
    <head>
        <title>t5 Start Page</title>
    </head>
    <body>
        <h1>t5 Start Page</h1>

        <t:form id="form1">
           <select t:type="select" value="item"
              model="listSelectionModel" encoder="listValueEncoder" />

           <input type="submit" t:type="Submit" />
           <br/><br/>
           <t:if test="item">
                  You just picked <b>${item.name}</b>, code <b>${item.code}</b>.
              <t:parameter name="else"> You have not selected yet. </t:parameter>
           </t:if>

        </t:form>
    </body>
</html>
```

Important: your bean have to override the **equals()** method from the **Object** superclass, so the component can compare your current selection with the appropriate bean inside the list. The Eclipse IDE provide a simple default implementation that solves the problem. Go to the menu **"Source/Generate hashCode() and equals()..."** to use this feature.