How to use the same Form for editing and new

Table of contents

One way

```
import wicket.markup.html.WebPage;
import wicket.markup.html.panel.FeedbackPanel;
import wicket.markup.html.form.Form;
import wicket.markup.html.form.TextField;
import wicket.markup.html.form.Button;
import wicket.markup.html.link.Link;
import wicket.markup.html.basic.Label;
import wicket.markup.html.list.ListView;
import wicket.markup.html.list.ListItem;
import wicket.model.CompoundPropertyModel;
import wicket.IFeedback;

import java.util.List;

class PersonList extends WebPage {
    private List peopleList;

    public PersonList(final List peopleList) {
        this.peopleList = peopleList;
        this.peopleList.add(new Person("Lila"));
        this.peopleList.add(new Person("Bender"));
        this.peopleList.add(new Person("Fry"));

        add(new ListView("peopleList", this.peopleList) {
            protected void populateItem(ListItem listItem) {
                final Person person = (Person) listItem.getModelObject();
                add(new Label("name", person.getName()));
                add(new Link("editPerson") {

                    public void onClick() {
                        setResponsePage(new PersonEdit(person, peopleList));
                    }
                });
            }
        });

        add(new Link("newPerson") {

            public void onClick() {
                setResponsePage(new PersonEdit(new Person(), peopleList));
            }
        });
    }
}

class PersonEdit extends WebPage {
    private FeedbackPanel feedbackPanel;
    private Person person;
    private List peopleList;

    public PersonEdit(Person person, List peopleList) {
        feedbackPanel = new FeedbackPanel("feedbackPanel");
        add(feedbackPanel);

        add(new PersonForm("personForm", person, feedbackPanel));
```

```
    }

    class PersonForm extends Form {
        public PersonForm(String s, Person person, IFeedback iFeedback) {
            super(s, new CompoundPropertyModel(person), iFeedback);
            add(new TextField("name"));
            add(new Button("save"));
        }

        public void onSubmit() {
            if (!peopleList.contains(person)) {
                peopleList.add(person);
            }
            setResponsePage(new PersonList(peopleList));
        }
    }
}


class Person {
    public Person() {
    }

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    private String name;
}
```

or another...

Very useful, thanks.

In similar vein, when you want to have a Form either editable or Read-only, and you don't want to create two different Markups, WebPages etc. One way of doing it, is to use exactly the same WebPage, Form and Markup, and add an AttributeModifyer to each form component (<input...> tag). You'd then set a boolean field to true when you want the page to be displayed/edited.

So normally you'd have something like this:

```
add(new TextField("total").add(IntegerValidator.INT));
```

(of course you don't always need an IntegerValidator)

Now create an AttributeModifyer that adds a readonly="readonly" attribute to the markup. The boolean field isReadOnly dictates whether the attribute is actually added to the output.

```
boolean isReadOnly = true; // set to true if you want the form to be readOnly; false to be editable

AttributeModifier ro =
 new AttributeModifier("readonly", isReadOnly, new Model("readonly"));  // For most Form components
AttributeModifier disabled =
 new AttributeModifier("disabled", isReadOnly, new Model("disabled"));  // For DropDowns etc.
```

And if you don't like the Gray-ish, washed-out look of read-only controls, define a style in your stylesheet and add that as well.

```
 AttributeModifier calcField =
  new AttributeModifier("class", isReadOnly, new Model("CalculatedField"));// The style class is called
CalculatedField
```

Now add the AttributeModifier to all your components, and the style/class attribute, eg:

```
 this.add(new TextField("total").add(IntegerValidator.INT).add(ro).add(calcField));
```

When you set isReadOnly to true now, the controls will be read-only, and will have the format you defined in your stylesheet

or another avoiding deprecated apis...

From version 1.5 Wicket has depricated the constructor

```
AttributeModifier(java.lang.String attribute, boolean addAttributeIfNotPresent, IModel<?> replaceModel)
```

To avoid using attribute modifiers you can sub class the TextField and override a couple of fields.

An example in Scala

```
  val commentTF = new TextField("commentTF", commentModel) {
    override def isEnabled() = !isReadOnly();
    override def onDisabled(tag: ComponentTag) = tag.put("readonly", "readonly");
  }
  form.add(commentTF);
```