

JFace Data binding

JFace permet de mettre en place un système permettant de mettre à jour les interfaces sur modification du modèle, et de mettre à jour le modèle, depuis la modification d'une interface :

-- Création du modèle

Sur le principe, chaque modèle va émettre un évènement sur modification de ses membres, évènement communiqué aux listeners associés au modèle.

-- Classe de base **AbstractModel**

La classe **AbstractModel** va permettre d'ajouter des listeners associés à des membres, et de déclencher des événements. Toute classe intégrée au modèle et utilisant le Data binding devra hériter de **AbstractModel**.

```
public abstract class AbstractModelObject {
    private final PropertyChangeSupport propertyChangeSupport = new
PropertyChangeSupport(
        this);

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        propertyChangeSupport.addPropertyChangeListener(listener);
    }

    public void addPropertyChangeListener(String propertyName,
        PropertyChangeListener listener) {
        propertyChangeSupport.addPropertyChangeListener(propertyName, listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        propertyChangeSupport.removePropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(String propertyName,
        PropertyChangeListener listener) {
        propertyChangeSupport.removePropertyChangeListener(propertyName,
            listener);
    }

    protected void firePropertyChange(String propertyName, Object oldValue,
        Object newValue) {
        propertyChangeSupport.firePropertyChange(propertyName, oldValue,
            newValue);
    }
}
```

-- Création d'un Model

Création d'une classe Utilisateur :

- Utilisateur doit être un Bean (constructeur+getters/setters)
- Hériter de **AbstractModel**
- Appeler **firePropertyChange** sur la modification de ses membres

```
public class Utilisateur extends AbstractModelObject {
    private String nom;
    private int age;

    public Utilisateur() {
    }

    public Utilisateur(String nom, int age) {
        this.nom = nom;
        this.age = age;
    }

    /**
     * @return the nom
     */
    public String getNom() {
        return nom;
    }

    /**
     * @param nom
     *         the nom to set
     */
    public void setNom(String nom) {
        String oldValue = this.nom;
        this.nom = nom;
        firePropertyChange("nom", oldValue, nom);
    }

    /**
     * @return the age
     */
    public int getAge() {
        return age;
    }

    /**
     * @param age
     *         the age to set
     */
    public void setAge(int age) {
        int oldValue = this.age;
        this.age = age;
        firePropertyChange("age", oldValue, age);
    }
}
```

-- Création d'une vue

Créer la classe FormUtilisateur, à partir de l'assistant **SWT/window Application**



-- Implémentation du code

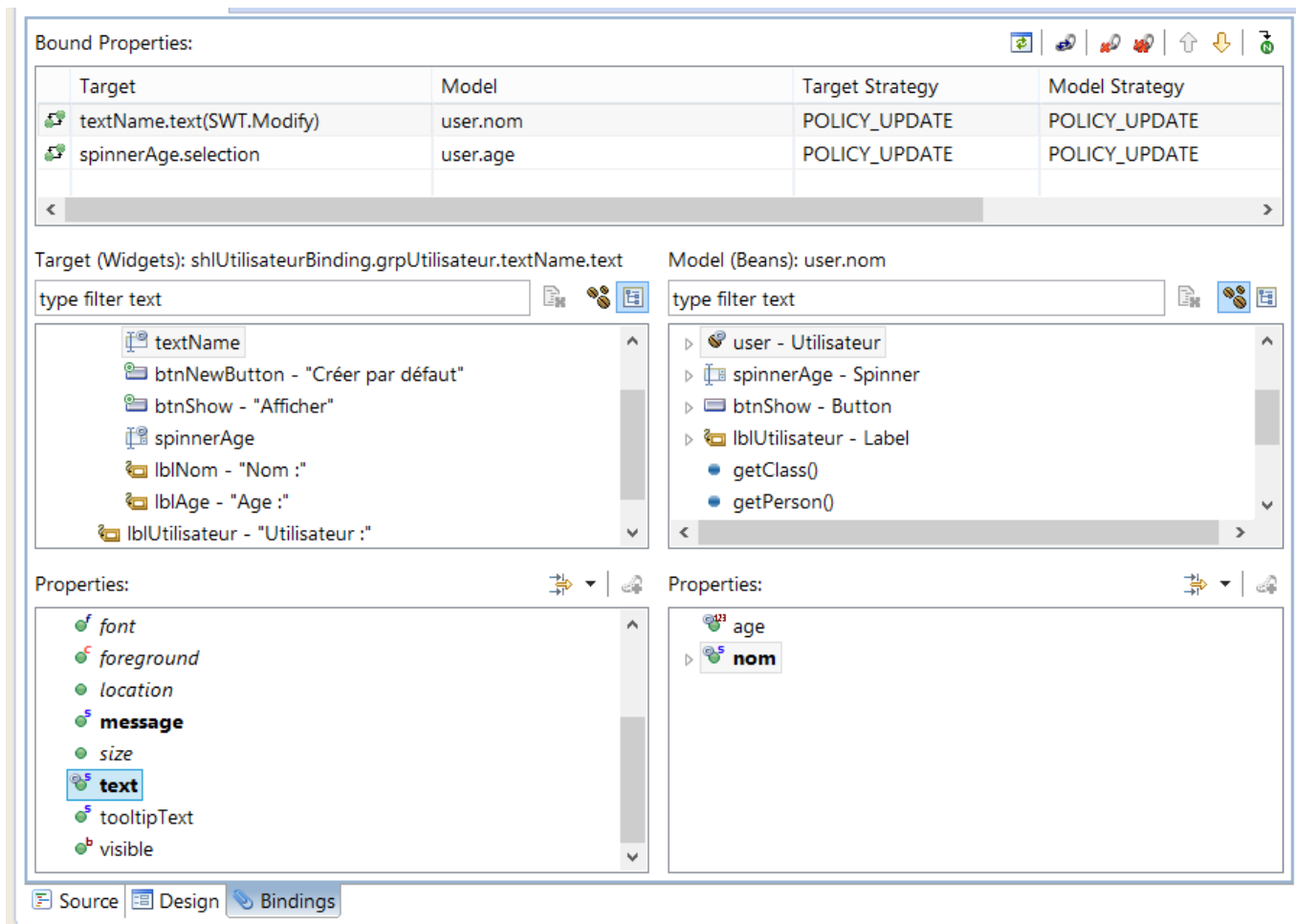
Implémenter les méthodes suivantes, puis tester l'interface

```
btnNewButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        user.setNom("DUPONT-LAJOIE");
        user.setAge(25);
    }
});
```

```
btnAfficher.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        lblUtilisateur.setText(user.toString());
    }
});
```

-- Définition du binding

Ajouter les bindings suivants :



Code généré :

```
protected DataBindingContext initDataBindings() {
    DataBindingContext bindingContext = new DataBindingContext();
    //
    IObservableValue observeTextTextNameObserveWidget =
WidgetProperties.text(SWT.Modify).observe(textName);
    IObservableValue namePersonObserveValue =
BeanProperties.value("nom").observe(user);
    bindingContext.bindValue(observeTextTextNameObserveWidget,
namePersonObserveValue, null, null);
    //
    IObservableValue observeSelectionSpinnerObserveWidget =
WidgetProperties.selection().observe(spinnerAge);
    IObservableValue ageUserObserveValue =
BeanProperties.value("age").observe(user);
    bindingContext.bindValue(observeSelectionSpinnerObserveWidget,
ageUserObserveValue, null, null);
    //
    return bindingContext;
}
```

Tester l'interface créée et son comportement

-- Binding sur listes d'objets

-- création du model

Implémenter le model Utilisateurs, permettant de gérer une liste d'instances d'Utilisateur :

```
public class Utilisateurs extends AbstractModelObject {
    private final List<Utilisateur> items = new ArrayList<>();

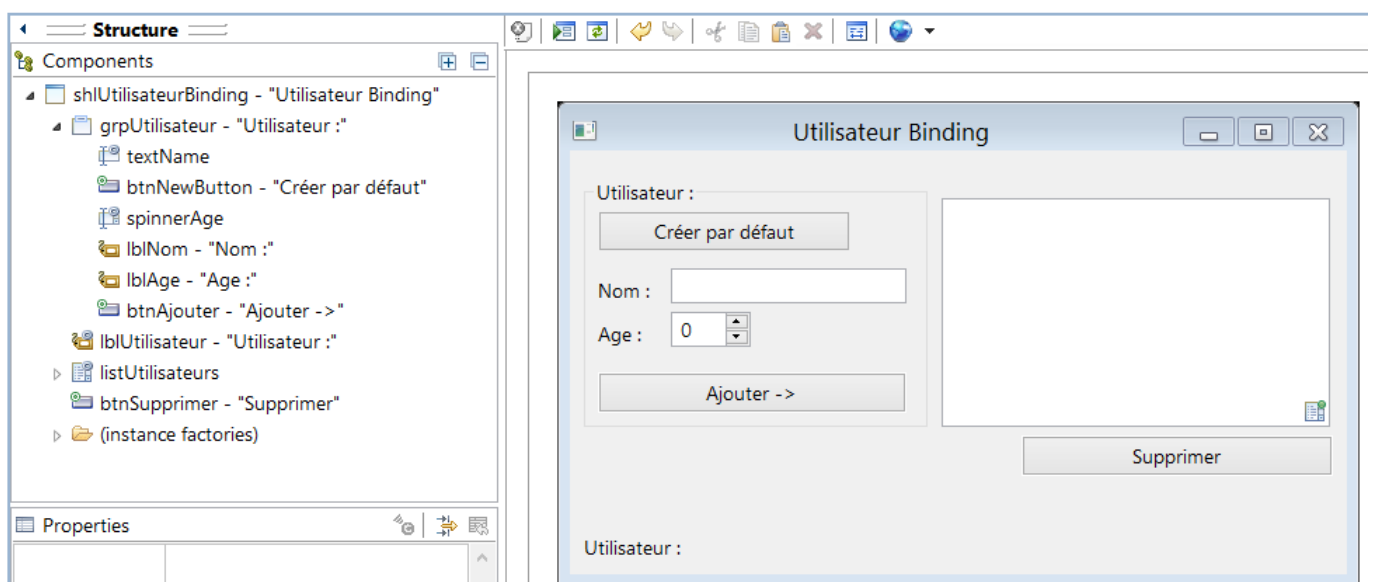
    public void add(Utilisateur utilisateur) {
        items.add(utilisateur);
        firePropertyChange("items", null, items);
    }

    public void remove(Utilisateur utilisateur) {
        items.remove(utilisateur);
        firePropertyChange("items", null, items);
    }

    /**
     * @return the items
     */
    public List<Utilisateur> getItems() {
        return items;
    }
}
```

-- Modification de la vue

- Ajouter à la vue les widgets manquants
- Ajouter le membre **Utilisateurs users**



-- Implémentation du comportement

```
btnAjouter.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        users.add(new Utilisateur(textName.getText(),
spinnerAge.getSelection()));
    }
});
```

```
btnSupprimer.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        IStructuredSelection selection = (IStructuredSelection)
listViewer.getSelection();
        if (selection != null)
            for (Iterator it = selection.iterator(); it.hasNext();) {
                Utilisateur aUser = (Utilisateur) it.next();
                users.remove(aUser);
            }
    }
});
```

From:

<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:

<http://slamwiki2.kobject.net/slam4/gui/jfacebinding?rev=1364431512>

Last update: **2019/08/31 14:39**

