Databinding

QU'EST-CE QUE LE DATABINDING?

Définition

- Moyen permettant de lier une variable Java à un composant JavaFX de manière permanente
  - Mise à jour automatique de la valeur côté Java
  - Mise à jour automatique du rendu visuel côté JavaFX
  - Détection des événements de changement et possibilité de greffer du code
- Possibilité de définir un binding unidirectionnel ou bidirectionnel
  - Le changement d'une valeur A implique un changement sur l'autre valeur B bindée
  - Le changement de B n'implique pas forcément le changement de A (paramétrable)

AVANTAGES ET INCONVÉNIENTS

Peser le pour et le contre

- L'intérêt est de diminuer le code de dynamique d'écran à écrire
  - Meilleure productivité
  - Diminution en LOC (Lines Of Code)
  - Bien maîtrisé, le databinding améliore la maintenabilité
- **Attention ! Le databinding peut aussi ajouter de la complexité !**
  - « Clarity over cleverness » s'impose ici
  - Un databinding complexe est difficile à déboguer
  - Un databinding est en général peu évolutif et il est difficile de greffer des modifications lors d'un changement de valeur
  - *Exemple : mettre la première lettre d'un mot en majuscule lors d'un databinding sur un champ String*

L'abstraction "Property"

Un standard depuis JDK 7

- La classe « Property »
  - Fournie par JavaFX
  - Sert de wrapper de valeur d'un type java (exemple : BooleanProperty)
  - Génère des événements pour JavaFX
  - Supportée par l'ensemble des composants JavaFX
- Un objet Property peut décrire
  - Un état graphique (taille du composant, style css appliqué, …)
  - Un état du modèle (texte d'un champs de saisie, position sélectionnée sur un slider, …)
- Pour utiliser un objet Property il faut définir
  - Des getters et setters
  - Une fonction nomProperty() renvoyant l'instance de l'objet Property

EXEMPLE DE CLASSE « BINDABLE »

```
public class Person {

        private BooleanProperty invited;
        private StringProperty firstName;
        private StringProperty lastName;
        private StringProperty email;

        private Person(boolean invited, String fName, String lName, String email) {
            this.invited = new SimpleBooleanProperty(invited);
            this.firstName = new SimpleStringProperty(fName);
            this.lastName = new SimpleStringProperty(lName);
            this.email = new SimpleStringProperty(email);
            this.invited = new SimpleBooleanProperty(invited);

            this.invited.addListener(new ChangeListener<Boolean>() {
                public void changed(ObservableValue<? extends Boolean> ov, Boolean t, Boolean t1) {
                    System.out.println(firstNameProperty().get() + " invited: " + t1);
                }
            });
        }

        public BooleanProperty invitedProperty() { return invited; }
        public StringProperty firstNameProperty() { return firstName; }
        public StringProperty lastNameProperty() { return lastName; }
        public StringProperty emailProperty() { return email; }

        public void setLastName(String lastName) { this.lastName.set(lastName); }
        public void setFirstName(String firstName) { this.firstName.set(firstName); }
        public void setEmail(String email) { this.email.set(email); }
    }
}
```

L'interface ObservableValue

Ajouter des listeners et des liens

- Toutes les classes de type Property implémentent ObservableValue

- Plusieurs fonctions utiles disponibles
  - addListener(ChangeListener) : évaluation stricte, quand la valeur change (empêche toute évaluation "lazy")
  - addListener(InvalidationListener) : évaluation "lazy", appelé quand la valeur est marquée comme invalide
  - bind(ObservableValue otherValue) : la propriété courante prend la valeur de otherValue dès que celle-ci change

```
TextField fieldSearch = new TextField() ;
person.lastNameProperty().bindBidirectional(fieldSearch.textProperty());
```

UTILISER DES EXPRESSIONS

Bindings avancés

- Une ObservableValue est une Expression avec des méthodes facilitant la composition de bindings
  - Exemple sur une DoubleExpression : add, divide, multiple, negate, …

```
DoubleProperty hauteur = new
SimpleDoubleProperty(40);
//largeur = hauteur / 2 + 10 = 30
DoubleExpression largeur = hauteur.divide(2).add(10);
//largeur va passer à 40 = 60 / 2 + 10
hauteur.set(60);
```

Attention !

Bien que puissantes, les expressions ajoutent de la complexité.
Il faut les réserver aux cas simples.

API statique « Bindings »

Utilisation avec des expressions

- Toolkit utile supportant les expressions

```
IntegerProperty num1 = new SimpleIntegerProperty(1);
IntegerProperty num2 = new SimpleIntegerProperty(2);
IntegerProperty num3 = new SimpleIntegerProperty(3);
IntegerProperty num4 = new SimpleIntegerProperty(4);
//approche combinée
NumberBinding total = Bindings.add(num1.multiply(num2),num3.multiply(num4));
```

- Ces méthodes permettent de mélanger les types pour les opérations arithmétiques (Integer et Long par exemple), en suivant les règles de Java pour la résolution du type final

Binding en mode « Eager »

Utilisation de ChangeListener

- L'évaluation immédiate (*eager*) est implémentée via des ChangeListeners
  - Le calcul de la valeur du binding est effectué dès que possible
  - Possibilité d'ajouter des listeners lors du changement de valeur
  - Incompatible avec l'évaluation "lazy"
- Choix par défaut de JavaFX
  - Cycle de vie plus facile à déboguer
  - Pas d'impact lourd sur les performances, tant qu'il n'y a pas d'opérations lourdes dans les bindings
  - Attention aux boucles infinies de bindings…

Binding en mode « Lazy »

Utilisation d'un InvalidationListener

- Principe d'utilisation
  - L'évaluation lazy ne calcule la nouvelle valeur que lorsque quelqu'un la demande
  - La mise à jour de la valeur s'effectue donc le plus tard possible
  - Parfois indispensable sous peine d'avoir des bindings infinis

- Exemple
  - On crée un binding "somme" additionnant par exemple 3 valeurs de type DoubleProperty
  - Une de ces 3 valeurs change, le binding somme est invalidé et ses InvalidationListeners notifiés
  - Tout changement ultérieur dans une des 3 valeurs est "ignoré" (somme est déjà invalidé)
  - L'utilisateur demande la valeur de somme (somme.getValue()), celle-ci est invalide donc recalculée au dernier moment

Binding personnalisé

De nombreuses possibilités

- Il suffit de surcharger la méthode computeValue et de renvoyer la valeur à jour choisie
- Il faut appeler la méthode bind dans le constructeur pour définir les instances de type Property observées

```
final DoubleProperty a = new SimpleDoubleProperty(1);
final DoubleProperty b = new SimpleDoubleProperty(2);
final DoubleProperty c = new SimpleDoubleProperty(3);
final DoubleProperty d = new SimpleDoubleProperty(4);

DoubleBinding db = new DoubleBinding() {
          // Constructeur de la classe anonyme
          {
                    // super est nécessaire pour profiter des
mécanismes d'invalidation
                    super.bind(a, b, c, d);
          }

          protected double computeValue() {
                    return (a.get() * b.get()) + (c.get() * d.get());
          }
};
```