Accesseurs.

```
  public JComponent getView() {
    return this.homeView;
  }

  public CatalogController getCatalogController() {
    return this.catalogController;
  }

  public FurnitureController getFurnitureController() {
    return this.furnitureController;
  }

  public void addHomeFurniture() {
```

Récupération des meubles sélectionnés dans le catalogue.

```
    List<CatalogPieceOfFurniture> selectedFurniture =
        this.preferences.getCatalog().getSelectedFurniture();
```

Si la sélection n'est pas vide, création de l'ensemble des nouvelles instances de meubles.

```
    if (!selectedFurniture.isEmpty()) {
      List<HomePieceOfFurniture> newFurniture =
          new ArrayList<HomePieceOfFurniture>();
      for (CatalogPieceOfFurniture piece : selectedFurniture) {
        newFurniture.add(new HomePieceOfFurniture(piece));
      }
```

Ajout des nouveaux meubles au logement par le biais du contrôleur du tableau des meubles.

```
      getFurnitureController().addFurniture(newFurniture);
    }
  }
}
```

Comme la vue principale n'affichera pour l'instant que l'arbre du catalogue et le tableau des meubles, son contrôleur est assez succinct, mais c'est dans cette classe qu'il faudra ajouter entre autres choses toutes les opérations qui activeront ou désactiveront les menus de l'application.

Contrôleur de la vue de l'arbre

La programmation de la classe `com.eteks.sweethome3d.swing.CatalogController` est simplissime : il suffit d'instancier la classe `CatalogTree` en lui passant le catalogue avec le contrôleur, et de transmettre la liste des meubles sélectionnés au modèle dans la méthode `setSelectedFurniture`.

Classe com.eteks.sweethome3d.swing.CatalogController

```
package com.eteks.sweethome3d.swing;

import java.util.List;
import javax.swing.JComponent;
import com.eteks.sweethome3d.model.*;

public class CatalogController {
  private Catalog    catalog;
  private JComponent catalogView;
```

```
  public CatalogController(Catalog catalog) {
    this.catalog = catalog;
    this.catalogView = new CatalogTree(catalog, this);
  }

  public JComponent getView() {
    return this.catalogView;
  }

  public void setSelectedFurniture(
           List<CatalogPieceOfFurniture> selectedFurniture) {
    this.catalog.setSelectedFurniture(selectedFurniture);
  }
}
```

Contrôleur de la vue du tableau

De façon similaire au contrôleur de l'arbre, la classe `com.eteks.sweethome3d.swing.FurnitureController` du contrôleur du tableau des meubles doit instancier la classe `FurnitureTable`.et transmettre la liste des meubles sélectionnés au logement dans la méthode `setSelectedFurniture`. Enfin, Thomas y implémente les méthodes `addFurniture` et `deleteSelection` qui prennent en charge l'ajout et la suppression de meubles dans le logement.

Classe com.eteks.sweethome3d.swing.FurnitureController

```
package com.eteks.sweethome3d.swing;

import java.util.List;
import javax.swing.JComponent;
import com.eteks.sweethome3d.model.*;

public class FurnitureController {
  private Home       home;
  private JComponent furnitureView;

  public FurnitureController(Home home,
                             UserPreferences preferences) {
    this.home = home;
    this.furnitureView =
        new FurnitureTable(home, preferences, this);
  }

  public JComponent getView() {
    return this.furnitureView;
  }

  public void addFurniture(List<HomePieceOfFurniture> furniture) {
```

Création de la vue associée à ce contrôleur.

Ajout des nouveaux meubles au logement.

Sélection des meubles ajoutés.

Suppression des meubles parmi les éléments sélectionnés dans le logement.

Sélection des meubles dans le logement.

```
    for (HomePieceOfFurniture piece : furniture) {
      this.home.addPieceOfFurniture(piece);
    }
    this.home.setSelectedItems(furniture);
  }

  public void deleteSelection() {
    for (Object item : this.home.getSelectedItems()) {
      if (item instanceof HomePieceOfFurniture) {
        this.home.deletePieceOfFurniture(
            (HomePieceOfFurniture)item);
      }
    }
  }

  public void setSelectedFurniture(
          List<HomePieceOfFurniture> selectedFurniture) {
    this.home.setSelectedItems(selectedFurniture);
  }
}
```

Notifications des modifications dans la couche métier

La programmation des contrôleurs dans Eclipse a permis à Thomas de générer toutes les nouvelles méthodes des classes `Catalog` et `Home`, sauf leurs méthodes `add...Listener` et `remove...Listener` de gestion des listeners.

Sélection des meubles du catalogue

Thomas complète tout d'abord la classe `com.eteks.sweethome3d.model.Catalog` :

1. Il ajoute les champs `selectedFurniture` et `selectionListeners` qu'il initialise avec des listes vides.
2. Il implémente les méthodes `addSelectionListener` et `removeSelectionListener` pour ajouter ou retirer un listener de l'ensemble `selectionListeners`.
3. Il modifie la méthode `getSelectedItems` pour renvoyer la sélection courante.
4. Il ajoute la méthode `setSelectedItems` pour mémoriser la sélection et émettre une notification de changement de sélection.

Classe com.eteks.sweethome3d.model.Catalog (modifiée)

```
package com.eteks.sweethome3d.model;

import java.util.*;

public abstract class Catalog {
  private List<Category> categories = new ArrayList<Category>();
  private boolean        sorted;
  private List<CatalogPieceOfFurniture> selectedFurniture =
       Collections.emptyList(); (1)
  private List<SelectionListener>       selectionListeners =
       new ArrayList<SelectionListener>();

  // Méthodes getCategories et add inchangées

  public void addSelectionListener(SelectionListener listener) { (2)
    this.selectionListeners.add(listener);
  }

  public void removeSelectionListener(
                  SelectionListener listener) { (3)
    this.selectionListeners.remove(listener);
  }

  public List<CatalogPieceOfFurniture> getSelectedFurniture() {
    return Collections.unmodifiableList(this.selectedFurniture);
  }

  public void setSelectedFurniture(
              List<CatalogPieceOfFurniture> selectedFurniture) {

    this.selectedFurniture =
       new ArrayList<CatalogPieceOfFurniture>(selectedFurniture); (4)
    if (!this.selectionListeners.isEmpty()) { (5)
      SelectionEvent selectionEvent =
          new SelectionEvent(this, getSelectedFurniture()); (6)
      SelectionListener [] listeners =
          this.selectionListeners.toArray(
          new SelectionListener [this.selectionListeners.size()]); (7)
      for (SelectionListener listener : listeners) { (8)
        listener.selectionChanged(selectionEvent); (9)
      }
    }
  }
}
```

Liste des meubles sélectionnés et des listeners.

Ajoute le listener en paramètre à la liste des listeners qui seront notifiés à chaque modification de la sélection.

Retire le listener en paramètre de la liste des listeners du catalogue.

Renvoie une liste non modifiable des meubles sélectionnés.

Sélectionne les éléments en paramètre.

Stockage d'une copie de la sélection en paramètre.

Création de l'événement correspondant à la modification.

Création d'une copie de la liste des listeners.

Notification des listeners.

JAVA 5 Collections.emptyList() vs Collections.EMPTY_LIST

À la différence de la constante EMPTY_LIST, la méthode emptyList de la classe java.util.Collections renvoie une liste générique. Le recours à cette méthode (1) évite un warning du compilateur de Java 5 émis aussitôt qu'on utilise une classe de collection sans spécifier entre les symboles < > la classe des éléments gérée par cette collection.

Une fois mémorisée la nouvelle sélection (4), la méthode setSelectedFurniture notifie à tous les listeners enregistrés dans la liste selectionListeners le changement de sélection (9). Pour permettre à un listener d'ajouter (2) ou de retirer (3) un listener (éventuellement lui-même) pendant l'appel à la méthode selectionChanged, Thomas a programmé une boucle (8) qui travaille sur une copie (7) de la liste des listeners. Sans le recours à cette copie, l'itérateur utilisé dans la boucle déclencherait une exception java.util.ConcurrentModificationException si la liste itérée était modifiée. Par ailleurs, Thomas a pris soin de créer cette copie et l'instance de SelectionEvent (6) uniquement si la liste des listeners n'est pas vide (5).

Interface du listener de sélection et classe d'événement associée

La création de la classe du logement a entraîné la création de l'interface SelectionListener et de la classe SelectionEvent dont Thomas a implémenté le constructeur et l'accesseur.

Classe com.eteks.sweethome3d.model.SelectionListener

```
package com.eteks.sweethome3d.model;

import java.util.EventListener;

public interface SelectionListener extends EventListener {
  void selectionChanged(SelectionEvent selectionEvent);
}
```

Sur la base des conventions JavaBeans, l'interface FurnitureListener dérive de java.util.EventListener et la classe FurnitureEvent est une sous-classe de java.util.EventObject.

Classe com.eteks.sweethome3d.model.SelectionEvent

```
package com.eteks.sweethome3d.model;

import java.util.EventObject;
import java.util.List;

public class SelectionEvent extends EventObject {
  private List selectedItems;

  public SelectionEvent(Object source, List selectedItems) { (1)
    super(source);
    this.selectedItems = selectedItems;
  }
```

```
  public List getSelectedItems() {
    return this.selectedItems;
  }
}
```

Bien que la source soit de type `Catalog` et la liste de type `List<CatalogPieceOfFurniture>` au moment de l'instanciation de `SelectionEvent` dans la classe `Catalog`, Thomas a utilisé les types plus généraux `Object` et `List` (1), car il va réutiliser la classe `SelectionEvent` et l'interface `SelectionListener` pour gérer les modifications de la sélection dans le logement. Dans ce cas, la source sera de type `Home` et les éléments de la liste ne seront pas typés.

Sélection et modification des meubles du logement

Thomas s'attache maintenant à compléter la classe `com.eteks.sweethome3d.model.Home` :

1. Il ajoute tout d'abord à cette classe un constructeur sans paramètre qui appelle le constructeur existant avec une liste vide.
2. Il modifie le constructeur existant pour initialiser les nouveaux champs `selectedItems`, `furnitureListeners` et `selectionListeners` qui stockent les éléments sélectionnés et les listeners.
3. Il implémente les méthodes `addFurnitureListener`, `removeFurnitureListeneraddSelectionListener` et `removeSelectionListener` pour ajouter ou retirer un listener des ensembles `furnitureListeners` et `selectionListeners`.
4. Il modifie les méthodes `addPieceOfFurniture` et `deletePieceOfFurniture` pour ajouter et retirer de l'ensemble `furniture` le meuble qu'elles reçoivent en paramètre, avant d'émettre la notification correspondante aux listeners de la liste `furnitureListeners`.
5. Il modifie les méthodes `getSelectedItems` et `setSelectedItems` pour gérer la sélection, et émettre une notification suite au changement de sélection.

Classe com.eteks.sweethome3d.model.Home (modifiée)

```
package com.eteks.sweethome3d.model;

import java.util.*;

public class Home {
  private List<HomePieceOfFurniture> furniture;
  private List<Object>               selectedItems;
  private List<FurnitureListener>    furnitureListeners;
  private List<SelectionListener>    selectionListeners;
```

Liste des meubles, des éléments sélectionnés et des listeners.