JavaFX Observable

The JavaFX *Observable* interface is the base interface for many container classes and interfaces in the JavaFX Collection Framework. For example, *ObservableList<E>* and *ObservableArray<T>* are two common implementations of the base *Observable* interface. The Java core Collection API already contains many useful container classes to represent the generic data structure of lists, set, and maps. For example, *java.util.ArrayList<E>* is a re-sizable array implementation of the *java.util.List<E>* interface to contain a list of objects. However, they are incapable of working seamlessly when synchronous functionality is required between the list model and the view component in a GUI scenario.

Before JavaFX, Swing developers relied on *ArrayList* to contain a list of objects and subsequently display them in a list-like UI control, such as *JList*. But, *ArrayList* is too generic and was not built keeping in mind the requirement of synchronization when associated with a view component. As a result, it is quite difficult to update, add, or remove objects from the model list and at the same time reflect changes in the view component. To overcome this problem, JavaFX uses observable interfaces and their implementation, such as *ObserverList<E>*.

Because *ObservableList<E>* adheres to the rules of the observable and observer paradigm of MVC, such as providing notification to its interested observer regarding any updation, addition or removal of objects from the model list, it became a de-facto container for using any lists in the JavaFX arena. Here, data representation in the model and view are synchronized seamlessly. JavaFX *ObserverList<E>* is typically used in UI controls such as *ListView* and *TableView*. Let's go through a quick example to see how *ObservableList<E>* is actually used.

A Quick Example

This is a simple example where a list of data is displayed in a JavaFX UI control called *ListView*. The underlying data model used is *ObservableList*. There are two *ListView* controls used to move data from one control to another, and two *ObservableList*s represent the data of the underlying model. The data not only moves from one control to another visually, but it also moves from one model to another. This makes data transition between the view and the model layers synchronous.

Conclusion

The Observable class and the Observer interface provide a invaluable impetus to implement sophisticated program architectures based on the Model-View-Controller paradigm. In fact, the observable and observer concept is more than just MVC. A good understanding of the underlying concept can help create more sophisticated code in Java. If observer and the observable portions of the Java core utility framework show the principle, the JavaFX observable classes and interfaces exactly show how those principles can be put to work.