21 Swinging with the Java Foundation Classes

21.1 Introduction

The Java Foundation Classes (JFC) are an advanced framework of classes aimed at simplifying the development and deployment of commercial-quality applications. Parts of the JFC are already in Java 1.1 – they include the new Delegation Event Model and lightweight components. However, the core part of the JFC is Swing\(^1\), the new GUI classes released as part of Java 1.2.

Developed by Sun and Netscape, Swing builds on the Internet Foundation Classes (IFC) to provide an unrivalled set of very flexible GUI components. It goes much further than the IFC, with a richer set of components, pluggable look and feel, and sophisticated component architecture. It also includes useful support services, such as the multi-level undo/redo framework. The new classes are 100% pure Java, and conform to the JavaBeans standard (see Chapter 4).

From the earliest stages, Swing was released to Java developers for feedback. Essentially, the aim was to give developers what they wanted. Indeed, the development team did respond to developer feedback, and Swing evolved significantly in that time.

In Chapter 19, we looked at the limitations of the Abstract Window Toolkit (AWT), and how these limitations led to the evolution of the Java UI classes through the IFC to the JFC and Swing.

In this chapter, we take at look at the important concepts behind Swing, and how they affect our application development. We start by looking at the underlying architecture of Swing (the Model–View–Controller (MVC) architecture), and how this is used to implement the pluggable look and feel. We then introduce the various GUI classes that make up Swing, before considering how easy it is to modify them, and to create new components from scratch. Finally, we look at some of the features that affect the development of applications with Swing: the new Action class, the Undo framework and Java Accessibility.

Please note that the examples in the text are compiled under the pre-release version (0.7) of Swing. In this release, Swing package names start with “com.sun.java.swing”, whereas in Java 1.2 they will start with “java.swing”. Please refer to our Web site for the most up-to-date versions of the example code.

\(^1\) Swing: apparently the choice of name was inspired by swing music playing during an early demo of the new components!

J. E. Hunt et al., *Key Java*  
© Springer-Verlag London Limited 1998


21.2 Model–View–Controller Architecture

Swing components use a modified version of the Model–View–Controller (MVC) architecture. The MVC architecture defines how the classes that make up a user interface should be structured, and was discussed in detail in Chapter 9. Its use is central in achieving the pluggable look and feel, although you do not need to understand it in order to use the new Swing components.

MVC separates a user interface (UI) into three classes (Figure 21.1). The model class encapsulates the data underlying the interface, and has no knowledge of how the data will be displayed, or how the UI will modify it. Not all components will have models—the JScrollPane is an example of one that does not.

Figure 21.1 Model–View–Controller architecture.

The view class displays the data contained by a model. In the context of Swing, the separate view class allows us to change the look of a component without changing its underlying data model.

The controller class manages the interaction with the user. It modifies the data model in order to update its contents, and refers to the view in order provide feedback to the user. By changing the controller class, we can give a component a different feel, without disrupting its views or data model. For example, we could create an MS Windows-style menu controller that displays menus in response to a mouse click, or a Macintosh-style controller that displays menus only when the mouse button is held down.

The MVC is a very powerful architecture. However, in practice things are a little more complicated. In all but the simplest interfaces, there can be a great deal of interaction between the controller and the view classes. This means that views and controllers can become complicated, and closely bound to each other. The Swing team took the approach of combining the view and controller into a single class, referred to as the delegate class (Figure 21.2). As well as solving the problem, this also made changing the look and feel more convenient, as the new delegate encapsulates the look and feel in a single class.