The Basic Ideas of Object-Oriented Software Frameworks

An *object-oriented software framework* is a collection of classes implementing the shared architecture and the common functionality of a family of applications. Each software framework provides an *Application Programming Interface* (*API*) through which an application program can either specialize the framework code or it can directly use the framework code.

Frameworks provide *domain-specific concepts*, which are generic units of functionality. Framework-based applications are constructed by writing *completion code* that instantiates these concepts. For example, a graphical user interface (GUI) framework such as *JFace* offers implementation for a set of GUI concepts, which include a text box, tree viewer, and context menu. The instantiation of such concepts requires various *implementation steps* in the completion code, such as subclassing framework-provided classes, implementing interfaces, and calling appropriate framework services.

In short, the framework classes define the architectural skeleton to which the application program must conform. Application programs can define their own classes that may extend classes from the framework and may implement interfaces from the framework. An application program may customize and interact with a framework in ways that are permitted by object-oriented programming languages such as sub-classing, implementing interfaces, overriding superclass methods, creating instances of framework classes, and calling methods of framework classes. For example, the Java source code in Fig. 1 creates a simple Java applet using the Java Swing and Java AWT frameworks. This simple applet presents a button on the bottom of the applet. Whenever the user clicks on the button, the text of the button changes from “Click Me!” to “Click Again!”. The bolded sections of the source code are references to framework classes, interfaces, and methods as described in the following:

1. Subclass framework class `JApplet`.
2. Implement framework interface `ActionListener`.
3. Hold a field of type framework class `JButton`.
4. Override framework method `init` from `JApplet`. Then, within this method:
   (a) Instantiate an object of type framework class `JButton`.
   (b) Call the framework method `addActionListener` on the the `JButton`.
   (c) Read the framework constant `BorderLayout.SOUTH`.

(d) Call the framework method `add` on the `JApplet`.

5. Override the framework method `start` from `JApplet`.
6. Override the framework method `stop` from `JApplet`.
7. Override the framework method `destroy` from `JApplet`.
8. Implement the framework-declared method `actionPerformed` from the framework interface `ActionListener`. Then, within this method:

   (a) Call the framework method `setText` on the `JButton`.

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JApplet;
import javax.swing.JButton;

public class ScrollingSimple
extends JApplet
implements ActionListener {

    JButton button;

    public void init() {
        button = new JButton("Click Me");
        button.addActionListener(this);
        add(BorderLayout.SOUTH, button);
    }
    public void start() {
        System.out.println("Applet Starting.");
    }
    public void stop() {
        System.out.println("Applet Stoping.");
    }
    public void destroy() {
        System.out.println("Destroy Method Called.");
    }
    public void actionPerformed(ActionEvent event) {
        button.setText("Click Again");
    }
}
```

Figure 1: A simple applet on top of the Java Swing and Java AWT frameworks.