Abstract Interface to Prototype Implementation with Eclipse & the WindowBuilder Plug-in

Part 1: Introduction

An abstract interface is useful for roughing out the basic capabilities of what an interaction space should do. This can easily be done using paper and post-it notes (different colors might be used to represent different types of interaction). After some more refinement of your ideas, decisions must be made about how the abstract components will actually be implemented. Screen sketches and paper prototypes are often useful at this stage. Visual Interface development tools may also be useful as the system moves towards development. Below are 2 design sketches made with the Pencil Project prototyping/sketching tool. On the left is a sketch of the original abstract interface; interaction areas are very roughly marked out. As the design is refined, the abstractions begin to be replaced by more concrete ideas of the physical screen components as shown on the right. All aspects of the design don't necessarily move towards concreteness at the same speed.

Here we show how Eclipse with the WindowBuilder plug-in (a visual interface building tool) can be used to construct a prototype implementation (coded with Java Swing). Even a code prototype can contain some areas with a high degree of abstraction. The example application, PolygonMaker, is designed to allow users to enter point values which are then converted into source code for different applications that construct polygons. As work progresses, abstractions are replaced with specific interface widgets. Even once code prototypes have been developed, it still can be quite natural to use paper, white boards etc. for low fidelity prototyping. Prototype sketches are not only useful for working out interface designs but also for discussions and insights about how the design should be represented in code. As we shall see, the code generated by visual build tools is not necessarily easy for humans to comprehend and maintain. Care must also be taken to realize if shortcuts are being taken to generate code for throw-away prototypes versus more solid code that may evolve into the final product.

Figure 1: Before code, design sketches are often used. Abstract interfaces (left) are very high-level. Eventually they are resolved to a design which uses physical interface components. A mix of abstraction and concreteness can leave more possibilities open while explore design alternatives (right).

Here we show how Eclipse with the WindowBuilder plug-in (a visual interface building tool) can be used to construct a prototype implementation (coded with Java Swing). Even a code prototype can contain some areas with a high degree of abstraction. The example application, PolygonMaker, is designed to allow users to enter point values which are then converted into source code for different applications that construct polygons. As work progresses, abstractions are replaced with specific interface widgets. Even once code prototypes have been developed, it still can be quite natural to use paper, white boards etc. for low fidelity prototyping. Prototype sketches are not only useful for working out interface designs but also for discussions and insights about how the design should be represented in code. As we shall see, the code generated by visual build tools is not necessarily easy for humans to comprehend and maintain. Care must also be taken to realize if shortcuts are being taken to generate code for throw-away prototypes versus more solid code that may evolve into the final product.
Begin by downloading the zip file containing the starting project. Unzip it and then use Eclipse to create a new project by importing its code. You may find it helpful to view the handout “Using Eclipse to Create a Project Based Upon Existing Source”. That handout gives an overview of the most important interaction areas of the Eclipse IDE and the particular example given not only shows how to import existing source into Eclipse but also gives an example of drawing with Java2D. If you need to develop a new interface component that does its own custom drawing, that should be very useful.

1: Examining the New Project

Once your project has been imported into Eclipse, open the source folders and examine the contents (Figure 2). The resource package contains an image to be used in the interface. There are five java source files provided, none of them are interface code; two of them have a red “x” indicating some sort of problem. These files actually contain unit tests for the model classes defined in the other source files. They use a library that hasn’t been added to the classpath yet. You can clear up the problem by opening one of the test files and placing the cursor on the first ‘x’ in the file (the import statement). Use the “Quick Fix” option (right menu) to “Fix Project Setup”; Add JUnit 4 to the build path.

JUnit is a testing framework used by the test files. It is common for the user interface design and software development to proceed in parallel. While the user interface is being designed, code can proceed on other aspects that are relatively independent of the user interface. Here, the developers have started to work on the model objects of the MVC (model-view-controller). Since the model does not depend upon the interface it can be developed and tested independently of the interface code. In fact, the tests may even be developed before the model using a technique called Test First Development (TFD) or Test Driven Development (TDD). These tests can be run often (even automated) and also serve as documentation for how the model object(s) work. See Figure 3.

Tests
Request model manipulations, query model, confirm expected behavior

What is the state?

Model
Underlying application objects, core logic, data, etc.

Model Answers

Question the model

View
Visualizes the model

Controller
Interprets user requests to interact with model. Makes sure view reflects current model state.

GUI

Model manipulation request

Tests Results
Request model manipulations, query model, confirm expected behavior

What is the state?

Model
Underlying application objects, core logic, data, etc.

Model Answers

Question the model

View
Visualizes the model

Controller
Interprets user requests to interact with model. Makes sure view reflects current model state.

GUI

Figure 3: The Model/View/Controller paradigm separates core logic and data from the GUI. This also allows the model to be developed and tested independently and in parallel with the development of the details of the GUI.

The JUnit framework is often used to write unit tests. In the Eclipse package explorer view, select the polygonmaker package and right-menu, “Run As…” , “JUnit Test”. Eclipse will run and visualize the results of the tests as shown to the left. Seven out of seven tests have run successfully – 3 for the POVrayConverter and 4 for the PolygonModel.
2. Begin the Prototype by Creating its JFrame and Readying the contentPane

a. Right menu on the polygonmaker package. New --> Other ... . A dialog box will come up allowing you to navigate among many items. Follow the path: WindowBuilder, Swing Designer, Application Window. Name this new class, PolygonMakerApp and Finish. Once your choice has been completed and the dialog box dismissed, the class is created and some source code is generated. A window showing the source code should appear. You should see that this new class contains a JFrame that represents the application's top level window.

b. Visualize and build the abstract prototype by switching to “Design Mode”. Click the “Design” tab near the bottom left of the Eclipse window. (Fig. 4 – callout A). Using these tabs you can switch back and forth between working with the visual designer and the source code. You should see an empty JFrame in the design area (callout B in Fig. 1). Looking at the Components in the Structure area (callout C in Fig. 1), you see there is a contentPane which will hold all the items which appear in the JFrame.

c. Visual containers like the contentPane can organize items within them in many ways. Swing manages this using objects called Layouts. Layouts use algorithms to control how the components are sized and positioned as they and the container itself are modified. A Layout good for interactive visual design, is not necessarily well suited to maintain manually. To begin with, we'll choose a Layout that is easy to work with interactively in Design mode. With the contentPane selected in the Components/Structure area (callout C in Fig. 1), look in the Properties pane (callout F in Fig. 1) for the Layout property. This shows the current Layout and to the right is a black triangle indicating a menu of alternatives. Use this menu to change the layout to “GroupLayout”. This layout manager is specifically designed to be used with visual designers. Layout managers are also indicated in the palette (callout D in Fig. 1).

3i. Add Abstract Components to the JFrame's contentPane (text labels)

a. Find the JLabel Component in the palette (callout E in Fig. 4). Click it and then click again within the frame in the designer area (the cursor appears as a green plus to show it is over an appropriate target). This is the label we'll use to represent point entry/display (callout G in Fig. 4).

b. With the label selected, look in its properties (callout F in Fig. 4) for “text” and to the right of that “...”. Click the “...” to open the editor. Label text should consist of a short one line phrase. Type in “Point Entry/Display” and click OK.

c. Another property, “horizontalAlignment” can be used to center the text. Try it.

d. The “background” property can be used to set the color. Click the “...” after the 3 rgb values and select a yellow color to match the Post-it note sketch. After hitting OK you may notice that the label background didn't actually change. Look for
the “opaque” property (enable advanced properties pointed to by callout F in Fig. 4) and make sure it is set to true.
ed. Drag the label around until you are happy with its size and placement. You may resize the top-level frame as well.
f. Copy and Paste to create labels for “Do Conversion” and “Source Code”. Edit their properties as needed.

3ii. Add Abstract Components to the JFrame's contentPane (image labels)

Images may be much more expressive then just text. Labels can easily be used to display images. Images can be created with other programs (Pencil can export objects as png files) or by scans etc. The image in the sample project was exported by Pencil. We'll assume that you already have created a label and that you now want to use it to display a picture.

a. With the “Source Code” label selected, look in its properties (callout F in Fig. 4) for “icon” and to the right of that “...”. Click the “...” to open the Image Chooser and select the “Classpath” option at the top. You should be able to expand the src and resource folders to see the image from the sample project. Select it. The image will show on the right. Once you click OK, you will see the image on the label in the design view.
b. Within the label properties, change its text to the empty string and hit enter. The label in the design view should just show the image now.
c. Edit the size and placement of the labels until you are happy with the layout.

You should have a result similar to Figure 5.

4. Run your Code.

a. With the PolygonMaker.java class created in Step 2a selected, right menu and select “Run As”. Use the “Java Application” option to launch your code. Afterwards, there is a green circle with a triangle icon near the top of the Eclipse window which launches the last run target.

In part 2, we'll look at replacing labels with other widgets.

This article mentioned Test First Development which is a technique often associated with agile development especially extreme programming. In this classic article, “Engineer Notebook: An Extreme Programming Episode” by Robert Martin and Robert Koss, the ideas of test first development and refactoring are illustrated along with pair programming as 2 developers work to complete a project.

For additional information on Java 2D graphics see Oracle's tutorial on Java 2D.