Advanced Programming

Lab Sheet 19 – Swing GUIs

Introduction

This lab reinforces your learning on how to create Java GUIs using Swing. You got some limited practice on this last year, so these exercises should build nicely on this existing knowledge. You are going to create a complete, but relatively simple, GUI-driven application. The application will be capable of creating and displaying barcodes in a small selection of standard formats, similar to the software used in inventory management systems to create labels to attach to stock. The barcodes themselves will be produced (and displayed) by a library named barbecue, which provides a simple API for generating barcodes, and allows them to be added to a Swing UI once generated.

Your application will have a single JTextField, in which the user enters the text/numbers from which they would like to create a barcode, some JRadioButtons to allow the user to specify which type of barcode to create, a button for the user to actually create the barcode, and a display area. My version of the application is depicted in Fig. 1, below.

Figure 1: Barcode Application GUI

Task 1 – Project Setup

You will need to create a new Java project in eclipse, as usual. You will also need to download the barbecue library (website here), and add the jar file to your project's build path, in the normal manner.

The barbecue jar file offers classes in the net.sourceforge.barbecue package, most notably the Barcode and BarcodeFactory classes. The Barcode class represents a generated barcode that can be added to your Swing GUI like any other widget (e.g. a JButton or JTextField). The BarcodeFactory class has various methods for creating different types of barcodes. Each of these methods has a name beginning with create, and can throw a BarcodeException if there is a problem.

You will probably be able to keep the code for this week's work in a single class, but I would advise against doing all the work in the main() method. Instead, use a constructor to set up the UI, and a method for generating the correct barcode when the user clicks the button.

Task 2 – GUI Design

You are free to duplicate the UI design was depicted previously in Fig. 1 using a BorderLayout and FlowLayout to control the northern region, or create your own design using the GridBag layout manager. In either case, you will likely spend the bulk of your development time this session getting the various components to display in the position you desire.

For now, use the barbecue library to generate a default barcode when the application starts, and add it into your UI in the correct place as a placeholder. There's a really easy to understand example of how to use the library here.

In order to get the JRadioButtons to behave correctly, you should add them to a ButtonGroup as well as adding them to the UI. This will make sure that only one radio button is selected at once. You do not need to add the ButtonGroup itself to the UI, its just there to control behaviour.

Note: The barbecue library is not perfect, and it sometimes refuses to display in the correct place and renders behind other widgets. To work around this, I always add the Barcode to my Swing UI in its own JPanel. Keep a variable reference to this JPanel for later, because it will make it considerably easier to update the UI when the user creates a new barcode.

Task 3 – The Action Listener

Once you're happy with the way your application looks, its time to actually make it work! You will need to add an ActionListener to your generate button, that calls a method in your code to create a new barcode and update the UI.

You can identify which type of barcode to generate by examining the state of the radio buttons in your application. The JRadioButton class has an .isSelected() method which returns true if the user has selected this radio button, and false otherwise. The actual generation of the barcode is done using the appropriate create method in the BarcodeFactory class.

The task of replacing your placeholder barcode with the newly-generated one is a little painful. If you placed the barcode in its own JPanel earlier, then you can call .removeAll() on the JPanel to remove the existing barcode, before calling .add() to add the new one. Once done, you will also need to add two additional calls to request that Swing refreshes your UI to display the new barcode. These are *repaint()* and *revalidate()*.

The repaint method can be called on a JPanel to mark it as "dirty" (i.e. in need of a refresh). The revalidate method can be called on a JPanel to request that swing redraws the portions of it that are marked as dirty. It is considered best practice to only request a refresh of the portions of the UI that

have changed, so I would recommend marking the JPanel containing the barcode as dirty with repaint(), but calling revalidate() on its parent layout, because if the user generates a smaller barcode than was previously present, some of the area to redraw is now outside of the barcode JPanel's bounds.

Extension Tasks

It is often more complex to create a UI design using a single GridBag layout than it is to create one using a nested structure of the simpler layouts. Try creating a version of your application using both methods, so that you can contrast the approaches.

The barbecue library supports a fair number of different barcode formats. Swap the radio buttons in your app for a JComboBox (see the documentation here), and add support for some other formats to your application.

You could even try adding support for saving the generated barcode to a file.