## Chapter 3: A Tabbed Pane

## A Tabbed Pane

The first change we make will be near the start of makeContent(), right after the two statements that set up the content pane's layout and border. We'll add a declaration for a JTabbedPane object, like this:

```
JTabbedPane tabby = new JTabbedPane();
```

This statement creates the tabbed windowpane. At this point, it has no tabs and no GUI components to display. We can add as many tabs to it as we want, but we'll only need two. We'll name them "Player View" and "Team View." The Player View tab will show us all the information about a single player, just like we set up in Lesson 10. The Team View will show us a scrollable list of all the players and their statistics.

Each tab can display a single component. Since the Player View tab has a sequence of labels and text fields to display, we'll need to define a panel to contain them and then add the panel to the tab. The Team View tab will only display a single component, the scroll area. But just to keep our look-and-feel consistent, we'll still put it in a panel with a border so that it will look similar to the Player View.

Let's create our panel and its layout and border, just like we have a number of times before:

```
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
panel.setBorder(BorderFactory.createEmptyBorder(6,6,6,6));
```

The next step is to change all the Player View components so that we add them to the panel instead of to the content pane. Here's the first one as an example; then I'll leave the rest to you. The only change to our code is that in the third and seventh lines, we replace the variable *contentPane* with *panel*.

```
nameLabel = new JLabel("Player Name:");
nameLabel.setFont(new Font("Trebuchet MS",Font.BOLD + Font.ITALIC,14));
panel.add(nameLabel);
playerName = new JTextField();
playerName.setFont(new Font("Trebuchet MS",Font.PLAIN,14));
playerName.setForeground(Color.BLUE);
panel.add(playerName);
```

Once the panel is ready, we create a tab in the tabbed pane to hold the panel. At the same time, we'll create a mnemonic key so we can switch tabs with the keyboard as well as with the mouse:

```
tabby.addTab("Player View", panel);
tabby.setMnemonicAt(0, KeyEvent.VK_P);
```

The first line does two things: It adds a tab to our tabbed pane with the text  $Player\ View\$ in it, and it puts our panel with its components in the body of the tab. The second line creates the key combination ALT+P as a keyboard mnemonic for the tab. The first argument in the setMnemonic() call, the number zero, indicates the tab we want. For this method, the tabs are referenced by the number of their position, and the first tab is in position zero. The second tab will be in position one, and so on. (By default, the addTab() method sets up the tabs from left to right in the order we create them. We can change that default sequence by using the insertTab() method instead of addTab(), but we don't have any need to do that in this lesson.)

Now we have one tab with the Player View. We're ready to set up the panel for the second view and add its tab. Since we're done with the first panel, we can reuse the variable that referred to it and create our second panel, its layout, and its border like this:

```
panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

panel.setBorder(BorderFactory.createEmptyBorder(6,6,6,6));
```

The last steps are all steps that we've done before. Here's what we'll do to create the second tab:

The first four lines set up a text area with scroll bars, like we built in Lesson 6. The fifth line adds it to the panel. The sixth line creates the tab to display the panel, with the title *Team View* in the tab. The last line sets the keyboard mnemonic for the tab as ALT + T.

We now have a tabbed pane with two tabs that contain the two views we want to see. All we need to do now is add the tabbed pane to our content pane, like this:

```
contentPane.add(tabby);
```

The program should now display both tabs and let you switch between them using either the mouse or the keyboard mnemonics. Please let me know in the Discussion Area if you have trouble making it work.

## **Creating Data for the Team View**

We have a Team View tab! But if you've tried to use it, you've probably noticed that there's no data in it. So as our last task in this part of the lesson, let's create the data display for the Team View tab and add it to the text area.

Where should we put the code? If you think about it, shouldn't it be in the same place where we add the data to the text fields in the Player View? The list first gets its data in the file open process we wrote for the actionPerformed() method in the OpenMenuItemListener inner class.

We'll add the code for the Team View data at the end of that method, and I think you'll be pleased by how easy it is. We've already set all the groundwork in the Player class. If you remember, the Player class has a toString() method in it that we used when we wanted to display a player's information. All we have to do now is get each player in the list and use its toString() method to add its information to our output.

The best part is that since we want to process every element of the list without saving our place in it, we can use Java's for-each loop. As we talked about in Lesson 2, the for-each loop processes a collection by running the code in the loop once *for each* element of the collection. This is what it looks like:

```
for (Player p : list)
{
   textArea.setText(textArea.getText() + p.toString() + "\n\n");
}
```

Pretty slick, isn't it? That's all we need to get each player and add her information to the scrolling text area.

Since it might be a little fuzzy in your memory, let me explain the format of the for-each loop and what it is doing for us. The *for* keyword tells Java we are setting up a loop. The parentheses contain the control information for the loop. First, we tell Java the type of object we want to process (Player), and then we give Java a variable name to use (p) when giving us the elements of the collection (*list*). So, in short, the first line tells Java to run the loop for every Player object in the collection list and to give us the Player object in the local variable named p.

This loop format is much simpler than the iterator process we used earlier, but each has its place. This loop format only works when we want to process all the collection elements at once. We can't do part of it now and part of it later like we can with an iterator. The for-each loop also only goes one way. We can't move backward, only forward. Finally, the for-each loop does not allow replacements or deletions during the loop process. We can't replace or delete any collection elements in the loop, or the loop's actions become unpredictable. Even though using an iterator is a little more work, it's necessary because iterators allow many actions that for-each loops don't.

Having said that, the for-each loop works perfectly in the situation we have here, so we'll keep it! The one line of code inside the loop retrieves the current text from the text area using the getText() method, adds the current player's formatted output string to it, then puts the updated text string back into the text area using the setText() method.

Once the loop finishes, the scrollable Team View is ready to display. Since that's all we need to do for the Team View, we're done with this version of the application! We can load a player file into the list collection we created for it and display the player information in both views. We can switch between the views by clicking the tabs. And in the Player View, we can move through the list using the Next and Previous menu items. Be sure to let me know in the Discussion Area if you have any problems making it all work.

I haven't given you the complete program code since the last lesson, so I'm including it here in case you would like to look at my version of it.

**Tabbed Pane Program** 

© 2022 Cengage Learning, Inc. All Rights Reserved