A More Functional Digital Clock

Here is a picture of your expanded digital clock:

![Digital Clock Applet](image)

You must change the graphical user interface by adding buttons and listeners in the digitClock class and then you must change the functional capabilities of the clock itself in the digitalClockPanel class.

(Step 1) Add those buttons and listeners in the digitalClock class

- In digitalClock constructor, change the layout from FlowLayout to a GridLayout with 2 rows and 4 columns
- Add the buttons “Hour +” through “Color” by creating then adding the buttons, in turn, before the existing “Start” button in controlPanel
- Add the action listeners by specifying
  `<put the button name here>.addActionListener(this);`

(Step 2) In the digitalClock class add the calls to the event handlers

- The actionPerformed method contains a sequence of if … else if … commands to determine which button was pushed; add an else if option for each of the new buttons where you test for the label on each button:
  `buttonLabel.equals("<put the label here>")`
- Once you have recognized a button had been pushed, you will call a method in the digitalClockPanel class to carry out the desired action. One such call already exists; when the “Reset” button is pushed then clockPanel.reset() is called. Each of the five new buttons will have a corresponding handler in the digitalClockPanel class to carry out the desired action. Choose meaningful names for these handlers since you will be writing the necessary code in the next step.

(Step 3) In the digitalClockPanel class add the event handlers

- The handler for “Hour +” through “Minute –” button presses are all very similar so we discuss them as a group. Whether you are incrementing or decrementing, you must wrap around when the extreme value is reached. For example, if you are at 59 minutes and the “Minute +” button is pressed, then the minutes value goes back to 0. In a similar manner, if you are at 0 hours and the “Hour –” button is pressed, you must reset the hours value to 23. We will use 24 hour time. In all cases, don’t forget to call repaint() to redraw the clock after a value has been changed.
• When the Color button is pressed, we will cycle through a sequence of colors. For simplicity, let’s use three predefined colors: Color.green (the current color), Color.blue, and Color.red. Using an if command, test the current color value then assign the next color value; don’t forget to “wrap around.” Remember to call repaint after changing the color.

Test your program and make sure it works correctly no matter what the user does.

**BONUS SECTION: If time permits, change the color button to a pull down menu.**

Here is the control section of our new screen:

![Control Panel Image]

Although only one button has been replaced with a pull down menu, this will require some work since handling a menu selection is much different than handling a button press.

(Step 1) Change the GUI component and add the appropriate listener

- In place of a Button, add an object of class Choice as started here:
  ```java
  Choice colorChoice = new Choice();
  colorChoice.addItem("green");
  ```
  Add the other color name to the pull down menu.
- In place of an actionListener, you will add an item listener
  ```java
  colorChoice.addItemListener(this);
  ```

(Step 2) Add the itemStateChanged(ItemEvent event) method associate with a Choice object; inside this method you will:
- Call the getSource method on event and assign the result to an object of type Choice
- If this object is not null, then test its value as indicated here:
  ```java
  if ("red".equals(event.getItem()))
  ```
  and then call a method in the digitalColorPanel class to set the display color. The parameter to this method will be Color.red, etc. depending on the alternative of the if command.

(Step 3) In digitalClockPanel, add the new method to handle color changes
- First remove the old method that was associated with the Color button press
- Then add the new method to set the color that is passed in the color value as a parameter; this is a very simple one line method

Test your program and make sure it works correctly no matter what the user does.

**CONGRATULATIONS! You have completed the lab project associated with the first session of the JavaJumpStart Tutorial.**