Carleton University
Department of Systems and Computer Engineering
SYSC 2004 - Object-Oriented Software Development - Winter 2015
Lab 10 - Introduction to Graphical User Interfaces

Objective

The purpose of this lab is to develop two programs that simulate a digital clock with a simple graphical user interface (GUI). The timekeeping hardware will be simulated by a button, which will increment the time by one minute each time it is clicked.

The program's *model* (that is, the classes that define the state and behaviour of the clock) will be implemented using a version of the `RollOverCounter` class that was introduced earlier in the term. Listener objects will be used to decouple the model from the classes that implement the GUI.

Attendance/Demo

To receive credit for this lab, you must demonstrate your work. **Also, you must submit your lab work to cuLearn**. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will grade the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before Lab 11.**

Getting Started

If you haven't done so already, download the zip file containing the *GUI-counter-v4* project that was presented in class last week. Extract the project and open it in BlueJ. Read the classes and run the program (execute the `main` method in `CounterUI`). If you understand this program, this lab should be very straightforward.

Step 1: Create a new folder named Lab 10.

Step 2: Download digital-clock.zip to your Lab 10 folder.

Step 3: Extract the *digital-clock* project from the zip file.

Step 4: Launch BlueJ and open the *digital-clock* project.

Part 1 - A Simple Digital Clock

Exercise 1 - Implementing the Model

The program you'll build will simulate a 24-hour clock; that is, a clock that counts from 00:00 (12 midnight) to 23:59 (11:59 p.m.). Let's start with the *model*. The model will use two `RollOverCounter` objects, one to keep track of the minutes and the other to keep track of the

hours.

To complete the model, you could define a `Clock` class whose methods collaborate directly with both roll-over counters (recall that this is what we did in the timer examples presented at the start of the course.) For this lab, you're going to learn a different approach. We won't have a `Clock` class; instead, the two `RollOverCounter` objects will be connected in a *chain*. The minutes counter will be incremented each time the button is clicked, and when this counter rolls over from 59 to 0, it will tell the hours counter to increment.

Step 1: Class `RollOverCounter` contains an incomplete implementation of method `toString`. Read the method's javadoc comment and complete the implementation of the method.

Step 2: Open class `RollOverCounterTest` in an editor window and read the `setUp` method. This method creates a digital clock by chaining together two `RollOverCounter` objects. Here's the relevant code:

```
private RollOverCounter minutes;
private RollOverCounter hours;
          ...
hours = new RollOverCounter(0, 23, null);
minutes = new RollOverCounter(0, 59, hours);
```

The `RollOverCounter` constructor is responsible for chaining two `RollOverCounter` objects together. The `RollOverCounter` referred to by `minutes` will store a reference to the `RollOverCounter` referred to by `hours`. The `RollOverCounter` referred to by `hours` will store the `null` reference to indicate that it is the last counter in the chain.

In effect, this code builds a linked list containing two counters. The counter that keeps track of minutes is at the head of the linked list, and is linked to ("points to") the counter that that keeps track of hours.

Step 3: In class `RollOverCounter`, change the constructor signatures to:

```
public RollOverCounter(RollOverCounter next)
```

and

```
public RollOverCounter(int minCount, int maxCount,
                       RollOverCounter next)
```

Modify both constructors so that they store the value passed in parameter `next` in a field (instance variable), which you will need to define. Make sure you understand how this causes the `RollOverCounter` object to be linked to another `RollOverCounter` object (the one referred to by parameter `next`).

Step 4: Modify the `countUp` method. When a `RollOverCounter` object's count wraps around from its maximum value to its minimum value, it should tell the next `RollOverCounter` object in the chain to increment its count. Make sure you handle the case where the

`RollOverCounter` object is the last counter in the chain.

Step 5: Compile `RollOverCounter` and `RollOverCounterTest` and run the unit tests. If any of the tests fail, make the required changes to the constructors, `toString` or `countUp`.

Exercise 2 - Constructing the User Interface

The program's user interface is very simple. It consists of a "Tick" button and three labels that display the current time in hours-minutes format (hh : mm); that is, two two-digit numbers separated by a colon.

Open class `ClockUI` in an editor window. The `main` method creates the program's window (an instance of class `JFrame`, which is part of Java's Swing library) and displays the user interface. You should not need to modify this class.

Open class `ClockPanel` in an editor window. This class is a subclass of class `JPanel`, which is part of the Swing library. Read the constructor. It creates the button (an instance of class `JButton`) and three labels (instances of class `JLabel`). Each component is placed on the panel by calling the `add` method inherited from `JPanel`. The constructor also builds the model by creating the chain of two `RollOverCounter` objects. Finally, it displays the current time by changing the text in two of the labels.

Step 1: Compile the application and interactively call the `main` method in `ClockUI`. What happens when you click the "Tick" button? Why? (Think about the examples presented during recent lectures.)

Exercise 3 - User Interface: Handling Button Clicks

Step 1: Class `TickButtonListener` is an incomplete implementation of a class that listens to `JButton` objects. This class implements the `ActionListener` interface, which means that it must provide a concrete implementation of the `actionPerformed` method.

- In class `TickButtonListener`, modify the `actionPerformed` method so that it calls `System.out.println` to print "Button clicked" on the BlueJ terminal.

Step 2: Now we're going to "wire" a `TickButtonListener` object to the "Tick" button. After we've done this, each time we click the "Tick" button, the listener's `actionPerformed` method will be called and "Button clicked" will be displayed.

- Modify the `ClockPanel` constructor to create a `TickButtonListener` object. Use `null` for both arguments passed to the `TickButtonListener` constructor. (You'll change this later).

- Register the `TickButtonListener` object as a listener of the "Tick" button. If you're not sure how to do this, review the `CounterPanel` class in the *GUI-counter-v4* project that was presented in class to see how it registers the listener for its button.

Step 3: Compile the application and call the `main` method. To display BlueJ's terminal window,

from the main menu select View > Show Terminal. Verify that every time you click the "Tick" button, "Button clicked" is displayed on the terminal. If this doesn't happen, fix your code before moving on to the next exercise.

Exercise 4 - User Interface: Connecting the Listener and the Model

The next step is to modify the program so that clicking the "Tick" button causes the time to increment.

Step 1: We need a bit of code that to verify that the `RollOverCounter`'s `countUp` method is called when the button is clicked.

- In class `RollOverCounter`, modify `countUp` to call `System.out.println` to print the counter's current value on the terminal.

Step 2: The `TickButtonListener` object needs to communicate with the `RollOverCounter` that counts minutes.

- The `TickButtonListener` constructor has two parameters. The first parameter has type `ClockPanel`, and the second parameter has type `RollOverCounter`. The code you wrote in Exercise 3 to create the `TickButtonListener` object passes `null` to both parameters. Modify the `ClockPanel` constructor so that the second argument passed to the `TickButtonListener` constructor is a reference to the `RollOverCounter` that counts minutes. (The first argument passed to `TickButtonListener()` should remain `null` for now.)

- In class `TickButtonListener`, modify `actionPerformed` to increment the time by one minute.

Step 3: Compile the application and call the `main` method. Verify that every time you click the "Tick" button, the current values of the minutes and hours counters are displayed on the terminal. Continue to click the "Tick" button until the hours counter reaches 1.

Exercise 5 - User Interface: Refreshing the User Interface

At this point, we have a program that correctly increments the clock's time by one minute each time the "Tick" button is clicked. To finish the program, all we have to do is get the listener tell the panel to update the GUI when the time changes.

Step 1: We need a method that the listener can call to update the user interface.

- In class `ClockPanel`, finish the implementation of `refreshUI` so that it displays the current time in the `JLabel` objects (call the `toString` method you implemented in Exercise 1 to get correctly formatted strings for the hours and minutes).

Step 2: The listener needs to know about the panel so that it can call `refreshUI`.

- Currently, when the the `TickButtonListener` is created, `null` is passed as the first argument. Modify the `ClockPanel` constructor so that the first argument passed to the

`TickButtonListener` constructor is a reference to the `ClockPanel` object.

- In class `TickButtonListener`, modify `actionPerformed` to request the `ClockPanel` to refresh the user interface, after the time has been incremented.

Step 3: The clock should now be up and running. Compile the program and call the `main` method. Verify that every time you click the "Tick" button, the current time shown in the GUI is incremented by one minute. Make sure that the hours field is incremented when minutes rolls over from 59 to 0.

Step 4: Remove all the `System.out.println` statements that you wrote earlier. Now that the program is finished, you no longer need them.

Part 2 - Adding a Reset Button to the Clock

Step 1: Close the *digital-clock* project. Make a copy of the digital-clock folder, and rename it as digital-clock-with-reset. Open the *digital-clock-with-reset* project in BlueJ.

Step 2: Apply what you learned in Part 1, and modify the program so that the GUI has a reset button to the right of the label that displays the minutes. Define a new class named `ResetButtonListener` to listen for clicks on this button. When the reset button is clicked, the clock's time should be reset to 00:00. (Notice that `RollOverCounter` has a `reset` method. You'll need to modify this method, similar to how you modified `countUp` in Part 1.)

When testing your program, make sure that the reset button causes both the hours and minutes to be zeroed. To do this, click the "Tick" button until the time is 01:01, then click the reset button. The clock should be reset to 00:00, and this time should be displayed in the GUI.

Wrap-Up

1. With one of the TAs watching, demonstrate your *digital-clock-with-reset* project. The TA will review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

2. The next thing you'll do is package the *digital-clock-with-reset* project in a *jar* (Java archive) file named digital-clock-with-reset.jar. (You do not have to submit your *digital-clock* project from Part 1). To do this:

    2.1. From the menu bar, select Project > Create Jar File... A dialog box will appear. Click the Include source and Include BlueJ project files check boxes. A check-mark should appear in each box. Click the down-arrow in the Main class field and select ClockUI from the list of class names in the drop-down menu.

    2.2. Click Continue. A dialog box will appear, asking you to specify the name for the jar file. Type digital-clock-with-reset or select the BlueJ icon named digital-clock-with-reset in the list of files. **Do not use any other name for your jar file** (e.g., lab10, my_project, etc.).

2.3. Click Create. BlueJ will create a file named digital-clock-with-reset that has extension .jar. (Note: you don't type this extension when you specify the filename in Step 2.2; instead, it's automatically appended when the jar file is created.) The jar file will contain copies of the Java source code and several other files associated with the project. (The original files in your digital-clock-with-reset folder will not be removed).

3. Before you leave the lab, log in to cuLearn and submit digital-clock-with-reset.jar. To do this:

   3.1. Click the Submit Lab 10 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag digital-clock-with-reset.jar to the File submissions box. Do not submit another type of file (e.g., a .java file, a RAR file, a .txt file, etc.)

   3.2. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status of your file is "Draft (not submitted)". If you're ready to finish submitting the file, jump to Step 3.4. If you instead want to replace or delete your "draft" file submission, follow the instructions in Step 3.3.

   3.3. You can replace or delete the file by clicking the Edit my submission button. The page containing the File submissions box will appear.

      3.3.1. To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

      3.3.2. To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

   3.4. Once you're sure that you don't want to make any changes to the project you're submitting, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading".