CSCU9A2 Practical 4A (Week 5) Spring 2017
Graphical User Interfaces 3 13/14 February

If you get stuck or need help at any time during the practical, ask a demonstrator.

Remember: Week 3's checkpoints (sheets 2A and 2B) must be completed by the end of this week. You have until the end of week 8 to complete this week's checkpoints.

THIS WORKSHEET:

In this worksheet, for the checkpoint, you will build a GUI for a simple Cash Register, as you might find in a shop. Then you will improvise some graphical drawing.

BUILDING A CASH REGISTER GUI

At this step you will build a new project from available Java implementing a very simple cash register, or shop checkout. This introduces new Java Swing user interface features. Then you will extend it with some more features.

- First you need to create a new basic application from an existing file as you have done before:
  - In a new folder in your CSCU9A2 working folder: Create a new BlueJ project using the file **CashRegister.java** from **Groups on Wide (V:)\CSCU9A2\Java**
- Compile the program and run it in the usual way. The cash register GUI has a text field into which you enter the price of the next item (decimal point allowed). When you click on the **Add item** button the price is obtained from the text field, is converted to a **`double`** value and is added to an internal variable **`totalPrice`**. The program then displays the new cash register total by *appending* a new line to the *text area* displayed in the window. When you have finished playing with it, close it by clicking the X at the top right.
- Open the program in BlueJ's editor: You can see the usual GUI set-up methods near the top of the program. Note that the work involved in the processing when the **Add item** button is clicked is started in the **`actionPerformed`** method which then calls a simple helper method **`addItem`** to update the internal total. **`actionPerformed`** also clears the price entry field and puts the "input focus" back into the price field for the user's convenience. Below a divider at line 84 you will find declarations for the **`totalPrice`** variable, and an item counter, and a collection of very simple methods that can be called to manage the information in those variables (**`addItem`** is one of them).
- You have not used a *text area* before: It is very much like a text field, but can have more than one line of text. The method **`setText`** can be used to instruct a text area to change its whole text (this is used in **`createGUI`** to set the text area empty initially). A useful method for text areas is **`append`**, to add more text *to the end of the currently displayed text*. To get text on multiple lines, the "new line" character "\n" must be included in the text sent to the text area. The typical use of **`append`** and new line is illustrated in method **`actionPerformed`**.
- First, give the window an appropriate title, and position it towards the centre of the screen (see earlier lab sheets.
- Now to make the application a little more interesting. **The steps below** describe some extensions for you to carry out: Adding a text field for entering item *descriptions* (for example: Cornflakes), and displaying the *item descriptions* and *prices* in the text area (*not the totals*), and having separate labels to display the item count and sales total.

- Here is a useful additional capability of the Java event framework: A text field can cause an **ActionEvent** when you are typing into it and you press the **Enter** key – this gives a more convenient alternative to clicking a button. Look at where the listener is added to the **Add item** button (in **`createGUI`**). Copy the line, paste it where **`priceField`** is created, and alter to add the listener to the price text field. Compile and test. Neat?
- Following the pattern already present: Add a new label below (that is: *after*) the text area (**itemsArea**) – this will be to display the current sales total (instead of in the text area). You will need to declare a new **JLabel** variable, and create an actual label in **`createGUI`**.
  Important: A **JLabel** can be used nicely for output – consult the online documentation for **JLabel** to find a method to change the text displayed by a **JLabel**.
- Add a new statement into **`actionPerformed`** that puts the new cash register total into the label that you have just created, prefixed with descriptive text (so the display will look like: **Sales total: 36.7**). Hint: Use the method that you found in the previous step. Compile and test.
- In **`actionPerformed`**, alter the line that appends the total to the text area, so that instead it appends the *price* of the item being added to the text area. Compile and test.
- *Before* the price entry label and text field, add a "**Description:**" label and a new text field for the user to enter the description of the item being added to the sale.
- When the **Add item** button is clicked, the item description should be fetched from the text field and appended to the text area along with the price, so that each line in the text area looks like this: **Cornflakes 1.87** Compile and test.
- The whole application GUI should now look much more like a complete record of the sale that is taking place, with all items and prices displayed as well as the overall total.

Remember: Your code must be well formatted and clearly commented.

Checkpoint [GUIs 3]: Show a demonstrator the final code for your Cash Register, and demonstrate it running. Answer any questions they ask you.

DRAWING A NICE PICTURE

In this exercise, you will take a ready made drawing application that displays a window with a drawing area that shows a simple red rectangle, and you will improvise a colourful picture in it.

- First you need to create a new basic application from an existing file as you have done before:
  - In a new folder in your CSCU9A2 working folder: Create a new BlueJ project using the file **Drawing.java** from **Groups on Wide (V:)\CSCU9A2\Java**
- Open the program in BlueJ's editor and take a look at it. Your only work will be in the **`paintScreen`** method, called from **`paintComponent`** when the drawing panel is refreshed.
- Using the **Graphics** drawing methods **setColor**, **drawRect**, **drawLine**, **fillOval**, etc and **drawString** (for adding text) improvise a colourful picture, and put labels next to some of the things in the picture (if your imagination is poor: draw a house and garden!). *If you need a reminder about the parameters required for the drawing methods, remember the **Java Class Libraries** option in BlueJ's **Help** menu – and look for the **Graphics** class at the bottom left.*
- Compile and test.

Remember: Your code must be well formatted and clearly commented.

That's all for this worksheet.

SBJ February 2017