CSCU9A2 Practical 3A (Week 4) Spring 2017
Graphical User Interfaces 1 (6/7 February)

Remember to register your practical attendance at the start of each session.

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

Remember: Week 2's checkpoints must be completed by the end of this week.

THIS WORKSHEET:

*This worksheet contains first exercises in building Java applications with Graphical User Interfaces (GUIs) – the usual way that we are all used to using computer software. In the first exercise you will experiment with a basic application with an empty window (frame), and then build a simple GUI for an application that prompts the user in BlueJ's **terminal window** for two numbers and then displays them and their sum in a GUI window. In the second exercise, you will extend the event-driven Car Park counter application from lectures.*

HOW TO FIND ESSENTIAL JAVA DOCUMENTATION

All Java programmers make frequent reference to the Java API documentation (API means: "Application Programmer's Interface"). This is a comprehensive, but sometimes hard to navigate, resource containing a detailed description of all the Java library classes, and all the methods that they contain. UNFORTUNATELY IT IS NOT A TUTORIAL ABOUT JAVA.

You will find the API documentation useful if you cannot remember precisely what a library method is called, or what parameters it requires, or just to explore for other useful methods.

To access the documentation when using BlueJ: In the **Help** menu select **Java Class Libraries**. This will launch a Web browser – you will be connected to Oracle's Java web site, and a dazzling screen of information will be displayed. Scroll down the long list in the pane at the lower left to find the *class* that you are interested in – for example look for **Scanner**, which you know quite well (you think!). Click on the class name when you find it to see the documentation in the main pane to the right. Take a look at it now – just a quick look to see what it's like – there is too much there to read! The **Scanner** class does start with a reasonably useful tutorial section, but that is not common (some descriptions contain links to Oracle's actual Java tutorials). Then there is a Method Summary that lists all the methods available to be used with a **Scanner** (for example: **hasNext**, **nextInt**, etc), and each method name is a hot-link to a more detailed description further down the page.

You can see that the documentation is extensive, detailed and complex. It is best to use it for looking up the details of something that you already know about but of which you have forgotten the details. Remember to look at the API documentation when it might help.

In the exercises below you will see another way to find out information about classes you are working with.

EXPERIMENTING WITH AN EMPTY FRAME – NO EVENTS YET!

At this step you will build a new project with a main program class that displays a simple empty frame, and then experiment with adjusting its settings. Finally you will extend it by adding three text fields to display two numbers and their sum.

- Create a new **SimpleFrame** folder in your CSCU9A2 working folder. Copy the file **SimpleFrame.java** from **Groups on Wide (V:)\CSCU9A2\Java** to your **SimpleFrame** folder.
- Launch BlueJ. Using the **Project** menu, **Open Non BlueJ...** create a BlueJ project in your **SimpleFrame** folder – remember, when you navigate to it click ***once*** on the folder name then **Open in** BlueJ – do not double click to open the folder.
- Open the **SimpleFrame** class in BlueJ's editor and take a look at the program.
- Compile the program run it in the usual way. A *very small* and rather unsatisfactory window will appear at the top left of the screen – only just large enough to hold the required buttons in the window's title bar (no space even for the title to be displayed). When you have finished inspecting it, close it by clicking the X at the top right.
- Here are some adjustments to try out – experiment with them, *recompiling and running after each change*:
  - Alter the values given to the two variables `FRAME_WIDTH` and `FRAME_HEIGHT` to give a reasonable size window. (You should then see the title displayed.)
  - Give the window a new title.
  - There is another `JFrame` library method, `setLocation,` that expects two `int` parameters for an x and y coordinate identifying the point on the screen where the top left corner of the window is to be placed. Use a call of this method to position the window somewhere central on the screen.
  - Rather than use explicit numbers for the parameters of `setLocation,` follow the pattern illustrated by the `setSize` method call: declare two new variables, assign them values, and use them as the actual parameters of `setLocation.` Make sure that you choose sensible names for the variables (simply calling them `x` and `y` is not informative enough). *[Note: The modifier keyword `final` indicates that these are not really "variables" as this the final assignment that is allowed to them. In effect, they are just names for specific values – often called "constants". Providing informative names for "key values" like this within a program is good practice.]*
  - To see what happens, move the `setVisible` method call to just after the statement that creates the `new SimpleFrame.` When you launch the program, you might see the window jumping around bit before settling down (it depends on the computer's speed and other factors – the effect may be more visible when you are adding more items to the frame later on, so try again then)! Move the `setVisible` back to where it was.
  - In method `createGUI`, alter the parameter of `setDefaultCloseOperation` to `DO_NOTHING_ON_CLOSE` When you run the program, it will now not be possible to close it by clicking the X in the top right corner! You can close the program by right-clicking on the stripy bar in BlueJ's main window and selecting **Reset Java Virtual Machine** – if you are running Java at the command prompt, then typing Ctrl-C will close the program. If you omit the `setDefaultCloseOperation` completely (e.g by placing comment bars before it), then the default default occurs which is `HIDE_ON_CLOSE` – the window disappears but the program is still running behind the scenes, which seems rather strange!

- You can play with the "hidden but still running" property like this:
  After the `setVisible` call in main, type in these lines:

```
int n = 0;
while(true)
{
    System.out.println(n++);
}
```

  That is an infinite loop that will count up continuously in BlueJ's terminal window. When the default close operation is `EXIT_ON_CLOSE`, then the counting stops when you click the X. When it is `HIDE_ON_CLOSE`, then the counting *continues* even though the window has disappeared!

  Delete, or comment out the infinite loop, and reinstate `EXIT_ON_CLOSE`.

- At the start of a new line anywhere in `main`, type `frame.` then **Ctrl Space**. BlueJ will automatically look up all the possible methods that could be used at that point (that is, `JFrame` library methods), and display them in a scrolling list – it might take a short while. Click on any method *once* and BlueJ will show a summary of its description. Click twice and a method call will be inserted in your program, with hints where the parameters are required.

- Look at the many **JFrame** methods that are available. Many are very obscure, but some interesting/useful ones are `setResizable(true/false)`, `setAlwaysOnTop(true/false)`, `setUndecorated(true/false)` – click once on each in the list to see their descriptions, and try them out. `setUndecorated(true)` has the unfortunate effect of showing a window with no X to click! To close such a window, right-click the stripy bar in BlueJ's main window and select **Reset JVM.**

- Now to make the application a little more interesting: It will input two integers from the terminal, and display their sum in the GUI window:
- Find the method from one of your previous projects which when called returns an integer from the console (terminal) input, and paste a copy of it into this program. You will need to add the **import** statement for the **Scanner** library class – look it up in a previous example.
- Just before the end of the **createGUI** method, add statements to declare two new `int` variables (say, `a` and `b`) and read values into them (with a prompt followed by two calls of the method that you pasted in the previous step). Declare one further `int` variable and assign it the sum of the first two.
- Now, following the pattern shown in the **Car Park** example on slide11 of the GUIs 1 lecture notes, add to the empty frame *three* new `JTextFields` that initially display 15 spaces as their text. Compile and run to make sure that you see the three text fields.
- Now, using the `setText` method of a `JTextField`, add *three* statements like this to display the two input values and their sum in the three text fields:

```
tf.setText("a is " + a);
```

- Make sure that window is an appropriate size, and has an appropriate title.
- Compile and test.

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

Checkpoint [GUIs 1]: Show a demonstrator your final code for the SimpleFrame, and demonstrate it running. Answer any questions they ask you.

- To see the effect of the **FlowLayout** manager at work, drag the bottom right corner of the program's window to make the window wider, and then narrower and taller.
- To see what happens if you do *not* request **FlowLayout**: Comment out the `setLayout` statement in `createGUI`. Compile and run: you will see just *one* button, the last one, and it occupies the whole window!! The default layout manager for a **JFrame** is **BorderLayout**: This places items in the NORTH, EAST, SOUTH, WEST and CENTER (CENTER is the default). If you would like to see how to use **BorderLayout**, you can look it up in the API documentation.

EXTENDING THE CAR PARK COUNTER APPLICATION

In this exercise, you will extend the functionality of an event-driven GUI:

- Create a new **CarPark** folder in your CSCU9A2 working folder. Copy the file **CarPark.java** from **Groups on Wide (V:)\CSCU9A2\Java** to your **CarPark** folder.
- Launch BlueJ. Using the **Project** menu, **Open Non BlueJ...** create a BlueJ project in your **CarPark** folder – remember, when you navigate to it click ***once*** on the folder name then **Open in** BlueJ – do not double click to open the folder.
- Open the **CarPark** class in BlueJ's editor and take a look at the program. This is the example discussed in the lecture GUIs 1.
- Compile the project and run it. A window is displayed with two buttons and a text field. When you click either button, the number displayed in the text field changes as specified in the `actionPerformed` method.
- A simple experiment: Put a comment marker at the start of *each line* `...addActionListener...` in **createGUI**, recompile and test. Now the program should *not* react when you click the button. Remove the comment marker. Those statements are necessary to configure the buttons so that they notify the program when they are clicked.
- Improvise the following extensions:
    - Add two more buttons and a text field for recording minibuses separately from cars.
    - Add a text field displaying the number of *spaces remaining* in the car park – you decide what the overall number of spaces is.
- Compile and test.
- For your interest: Adapt so that when the car park is full, no further vehicles can enter, when there are no cars in the car park, none can leave, and when there are no minibuses in the car park, none can leave (the button clicks should cause no change to the counters).
- Investigate disabling and enabling buttons as an alternative technique to prevent cars entering and leaving the car park.

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

That's all for this worksheet.

SBJ 3 February 2017