CSCU9A2 Practical 9B (week 11) Spring 2017
Object Oriented Programming 1 30th March

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

Remember: ***Week 9's checkpoints (sheets 7A and 7B) must be completed by the end of this week. You have until the end of week 12 to complete this week's checkpoint.***

THIS WORKSHEET:

This worksheet introduces you to experimenting with classes and objects on BlueJ's "object workbench", and to the simple application of classes and objects.

USING THE BLUEJ OBJECT WORKBENCH TO EXPERIMENT WITH CLASSES AND OBJECTS

Although classes such as the **StudentRecord** class from lectures cannot be "run" in the usual way, BlueJ does allow us to exercise their capabilities in a neat way – to help understand them and to test them.

- Create a new project folder called **StudentRecords** in your **CSCU9A2** folder on **H:**
- Copy **StudentRecord.java** from **V:\CSCU9A2\Java** to the new project folder.
- Launch BlueJ and create a new BlueJ project in your **StudentRecords** folder.
- Open the **StudentRecord** class in the BlueJ editor and read it carefully. This is the **StudentRecord** class studied in lectures – each instance of the class is designed to hold the information about one student. Actually, there is *one extra public method*, at the end, called **toString**. This is a standard extra method that can be added to any class: when called it simply returns one string containing a summary of the details held in the object – *we choose* what it actually returns, and here it is the name, registration number, and number of credits. This can be useful in testing: if variable **student** contains an instance of **StudentRecord**, then `System.out.println(student);` will print a summary of the student's details on the terminal, automatically calling **student.toString()**. (Without the declaration of **toString** that statement would produce something strange: a representation of the memory address *reference* in the variable **student**! Try it!)
- For the moment there is no "main program" that can be executed in the usual way, but BlueJ will allow you to "play" with **StudentRecord** on its own – in effect going through the actions that a main program might take in using the class.
- Make sure that the **StudentRecord** class has been compiled.
- Right-click the **StudentRecord** class icon in the BlueJ window. The pop up menu shows you various actions that you could choose in red, and also all of the *methods* applicable to that item. The only method is **new StudentRecord(…, …)** – this is the constructor for **StudentRecord**, indicating that we can ask BlueJ to build an instance of **StudentRecord** for us. Select **new StudentRecord(…, …)** and BlueJ will start to create a new instance of the class.
- A dialogue box appears, showing the constructor header (and the Javadoc comment if there is one), and suggesting a name to give to the new instance object (probably **studentR1**). The constructor expects parameters, so there is also a section for entering actual parameter values. Either change or accept the suggested name, enter two strings (including the quotes, e.g. "Pythagoras" and "00214159") in the two parameter value boxes, and click OK.
- A red icon representing the new object appears in the "object workbench" at the bottom of the BlueJ window, labelled with the chosen name, and the name of the class that it is an instance of.
- When you right-click on the object, a pop up menu appears offering various options, including the *public* methods of the object, plus other options *Inspect*, *Remove* and *Inherited from...* (ignore

that last one). *Remove* discards the object – try it and then re-instantiate the class. *Inspect* pops up a dialogue box with information about the object – in this case the values held in the four instance variables – look at what it displays, make sure that you understand *why* each variable has the value shown (ask if you do not understand), and then click OK.

- Instantiate two more **StudentRecord** objects – with different names and registration numbers. *Inspect* each to make sure that they have been built correctly.
- To exercise the individual methods in a **StudentRecord** object, select the method from the object’s right-click pop up menu:
- For one of the **StudentRecord** objects, select **getName**: This method expects no parameters, so BlueJ immediately calls it for that object and pops up a box showing the result. Try various "**get**" methods from the various objects.
- For one of the **StudentRecord** objects, select **setAddress**: This method expects a parameter (the student's new address), so a dialogue box appears allowing you to enter a string that will be passed to the method as its actual parameter. Enter a new address and click OK. The dialogue box will simply disappear – the method has been called, has updated the **address** instance variable in the object, but returns no result.
- To check that the address has been updated correctly: either *Inspect* the object or call its **getAddress** method – try both.
- *Inspect* the *other objects* to make sure that *their* addresses have NOT changed. This illustrates and emphasises that each object has its *own copy* of instance variables, and a method called for one object accesses *only the variables for that object*.
- In a similar way, exercise **addACredit**.
- You should be able to keep several object inspector dialogues open, moved aside from the main BlueJ window, and watch their details as you call the "mutator" methods **setAddress** and **addACredit**. Try this.
- Try calling **toString** from each object.

EXTENDING THE STUDENTRECORD CLASS

Here are some useful additions to the **StudentRecord** class for you to implement:

- Add an instance variable to hold the name of the student's degree programme (for example "Computing Science" or "Basket Weaving").
- *Either* extend the constructor with one more parameter to receive an initial degree name, *or* provide a "**set**" method to update the degree name (and make it blank initially) – you choose.
- Add a new public method, **addMultipleCredits**: this should receive an integer parameter, which should be added to the credits obtained.
- Test your new features carefully using the object workbench.

Remember that your code should be neatly formatted and commented.

Checkpoint [OOP 1]: Explain to a demonstrator your extensions to StudentRecord, and show them how you tested the extensions. Answer any questions they ask you.

CONTINUED ON THE NEXT PAGE...

A STUDENTRECORDS MAIN PROGRAM

- Now copy **Database.java** from **V:\CSCU9A2\Java** to your **StudentRecords** project folder.
- Close and then re-open the BlueJ project. The new class should appear in the BlueJ window.
- Open the **Database** class. This is a standard main program with a GUI (the one seen in lectures): It instantiates and manages one student record, displays the student's details in a text area, and allows the address to be updated and credits to be added (one at a time).
- Try out the **Database** program in the usual way. **NOTE**: If you extended the **StudentRecord** *constructor* above, then you will need to edit line 71 in **Database** where the constructor is used (add a degree programme).
- Here are some extensions to make to the **Database** class (no further changes are required in the **StudentRecord** class):
- In **displayDetails**, now include the degree programme name in the display.
- Declare two more variables to hold **StudentRecord** objects. In **setUpData** instantiate two new **StudentRecord** objects for those variables (you invent the details).
- In **displayDetails**, now include the two new students' details *below* the first student's.
- Add two buttons: when one is clicked then *two* credits should be added to the *second* student's record, when the other is clicked then *three* credits should be added to the *third* student's record (use **addMultipleCredits**). Of course, the display should update automatically.

Remember that your code should be neatly formatted and commented.

SBJ March 2017