COMP2331 Object Oriented Design and Development

Session 2 Worksheet

Introduction

Today's worksheet gives you a chance to explore classes and objects of a bit of software (the beginnings of an aquarium simulation). You will also have the opportunity to write a bit of your own code. And you'll be delighted to see some (re-worked) characters from last year!

Aims

When you have completed this worksheet, you will be able to:

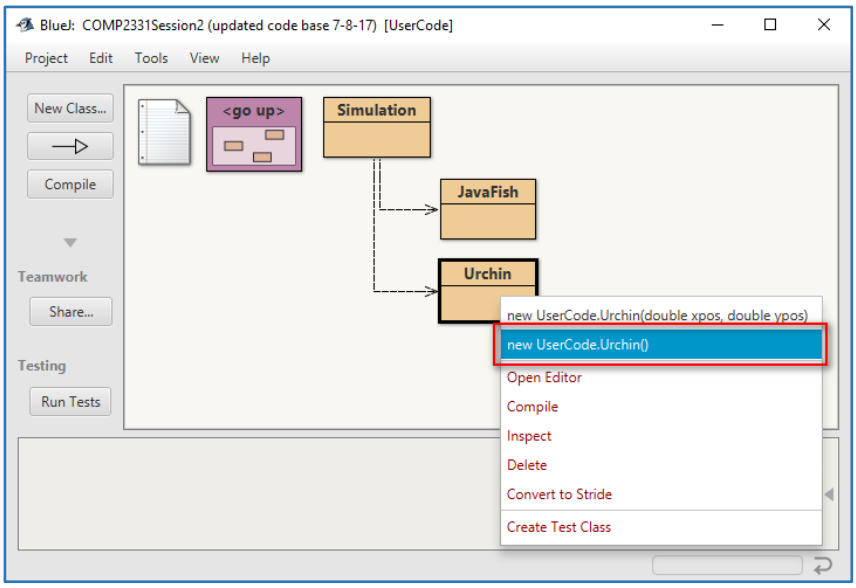
1. Explain what a class is, what an object is, and how they differ.
2. Explain what the various components of a class are and what they do.
3. Explain what accessors and mutators are and how they are used.
4. Apply the above knowledge to modify a small piece of software.

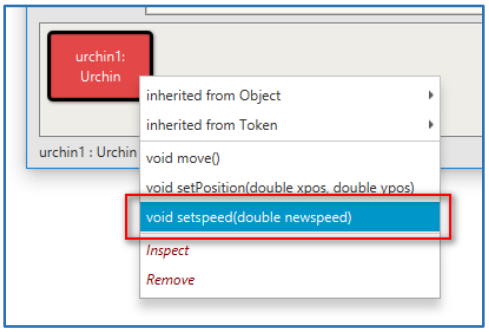
Procedure

Please follow the steps below carefully, remembering to make notes in your logbook as you go.

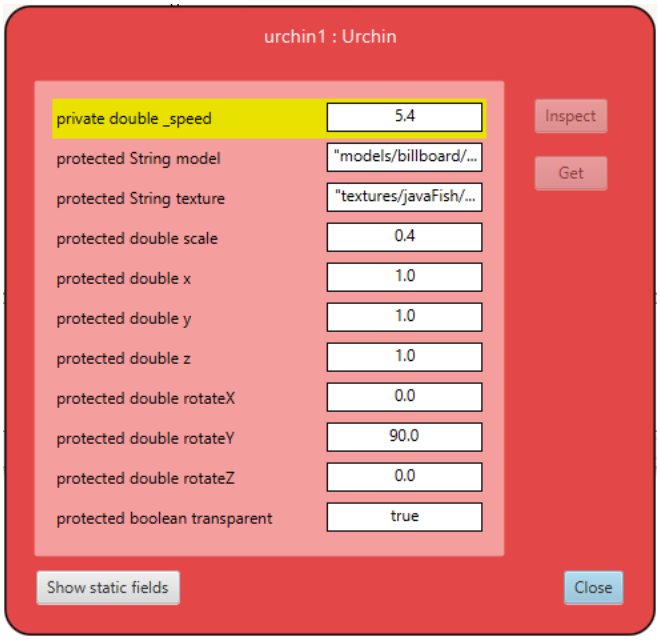
1. On Blackboard (Sessions->Session2->Code) is a file called 'Session2.7z'. Download this, extract it onto your personal storage area, and open the project in BlueJ. You will see two 'folders' in the canvas area: UserCode and Framework. Double-click on the UserCode folder, which will open a second window, and its canvas area will show three classes: Simulation, JavaFish, and Urchin.

2. Create an Urchin object on the object bench (name it urchin1). Call the method `setSpeed(double newspeed)` passing a suitable parameter for `newspeed`. What is the Type of `newspeed`? What is the full range of values that this datatype can that hold?



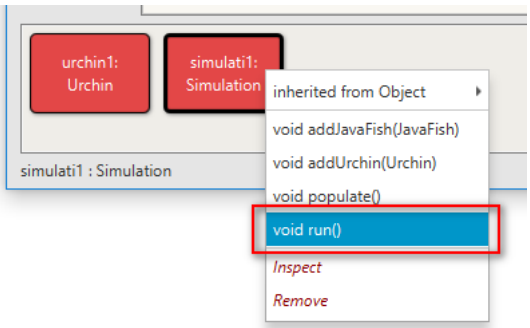


3. Inspect the urchin1 object to see which fields the `setSpeed()` method affects. (Right click object > Inspect)



4. Why isn't the urchin1 object displayed?

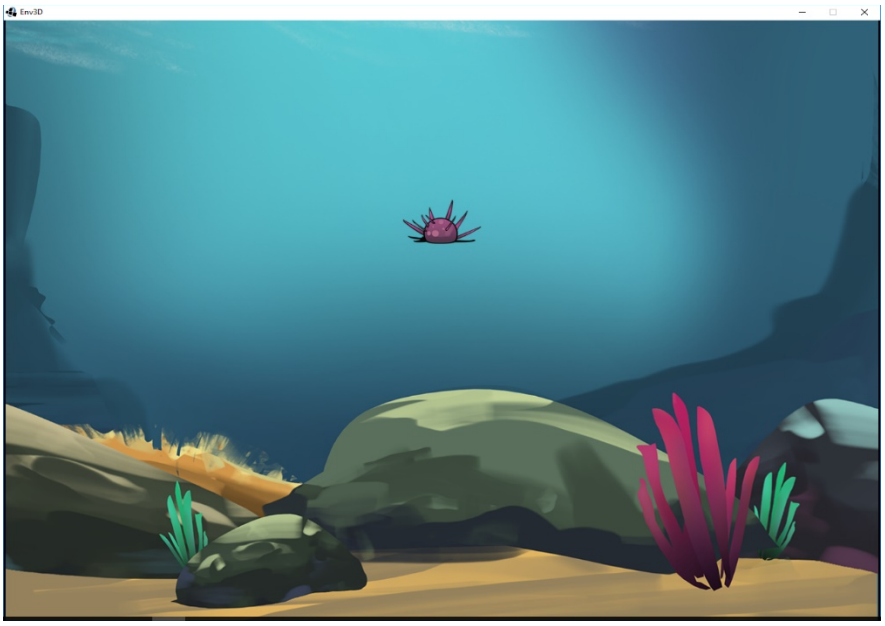
5. The Simulation class is a top-level class that runs the simulation loop. Create an instance of it (call it simulation1) and then call its `run()` method. A window will appear that shows the aquarium simulation. Why do you not see the urchin1 in there?



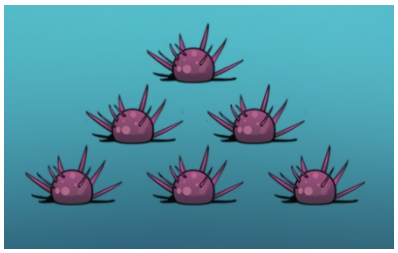
6. Quit the simulation by **pressing Esc**. As this will delete all objects from the object bench, go ahead and create an urchin1 object and simulation1 object again.

7. Call urchin1's `setPosition(double x, double y)` method, passing 5.0 to both parameters. This will tell the simulation1 object where to initially place urchin1 in the aquarium. Call the simulation1 object's `addUrchin(Urchin urchin)` method to add the urchin1 object to the aquarium.

8. Call the simulation object's `run()` method and see what happens. Repeat steps 6 and 7 with different values for the urchin's position. What is the range of **valid** values that the x and y parameters of `setPosition()` can have? (A full investigation here will help you become familiar with the aquariums coordinate system)



9. Now build an urchin pyramid by creating a number of Urchin instances and placing them appropriately in the aquarium. You might want to sketch your design on a piece of paper before you begin.



10. Try creating a JavaFish instance and place it into the aquarium – you may need to explore its methods to see what effect they have on its placement in the aquarium.

11. Double-click on the JavaFish class to reveal the code. It should begin with:

```
public class JavaFish extends DisplayObject
{
```

12. The `extends DisplayObject` part of the above is 'sub-classing' the JavaFish class from the DisplayObject class. The DisplayObject class is built into the framework you are using, which you will find in the Framework

folder. This allows us to provide some under-the-hood functionality that would otherwise over-complicate the JavaFish class. Feel free to take a look at the classes inside the Framework folder if you wish, but this is not necessary. For the time being, we can ignore `extends DisplayObject`, as this will be covered when we explore **inheritance** later in the semester.

13. Looking at the beginning of the above line of code, does it make a difference if we switch the order of the `public` and `class` keywords? Ie:

```
class public JavaFish extends DisplayObject
{
```

Edit the source code of the JavaFish class to try this out, and save it. How does the look of the editor change as a result? How does the class diagram change as a result?

14. What error message do you get if you press the 'compile' button? Does this message clearly explain what is wrong?

15. Change the class back to how it was and make sure this clears the error when you compile it.

16. Check whether or not it is possible to leave out the `public` keyword from the outer wrapper of the JavaFish class. Put back the `public` keyword and then check whether it is possible to leave out the `class` keyword by trying to compile it again. Make sure both words are put back as they were originally before continuing.

17. Make a list of the names of the fields, constructors, and methods in the JavaFish class.

18. What are the two features of the constructors that make them look significantly different from the methods of the class?

19. In the following field declaration from the JavaFish class,

```
private double _speed;
```

    does it matter which order the three words appear in? Edit the source code to try different orderings, and notice how the appearance of the editor canvas and the class diagram change. Does this change in appearance give you a clue as to whether or not other orderings are possible? Make sure you reinstate the original version after your experiments!

20. Is it always necessary to put a semicolon at the end of a field declaration? Experiment with the editor to find out.

21. Write an accessor method called `getSpeed` in the JavaFish class. The new method should return the value of the `_speed` field. Compile and test it. (Hint: to test make a JavaFish object and call your new method from the list of available methods)

22. How can we tell from just its header that `setSpeed` is a method and not a constructor? What kind of method is it?

23. Complete the empty `accelerate` method in the JavaFish class so that it adds the value of the parameter to the `_speed` field. Test this by making use of your getSpeed method.

24. Observe the swim behaviour of the simulated fish. Do you notice anything odd about it? What might you do to correct it? (Remember that the aquarium is 3D and the fish is a **flat** square ☺)