LAB 1: CREATING OBJECTS

In this lab you will use BlueJ to create and manipulate some Java objects. In an object-oriented program objects often model some kind of real-world "things". The objects in this lab represent shapes (circles, rectangles, etc.). These objects can also draw visual a representation of themselves so that you can see the effect of anything you do with them.

You should attempt the steps for each task and try to answer the ***questions*** that follow some of the steps. It is recommended that you keep a lab note book as you work through the labs and take note of your answers to the questions in each lab.

Related reading

Lecture Notes - *1: Introduction to Object-Oriented Programming*

Working with BlueJ and lab code

The software, including BlueJ, that you need for the lab exercises in this module is installed in a Windows 8.1 Virtual Machine (VM) on each PC in the computer labs. Once you have logged in to the PC you can start the appropriate VM as follows:

- click on the **VMware** icon on the desktop – this should open the folder containing the VMs that are on the PC. Each VM is in its own folder.
- find the VM folder whose name begins with *SEBE-VM-Win8.1x64Ent*
- open this VM folder and find the file whose type is *VMware virtual machine configuration*. double-click on this file. VMware Player should start and load the VM. You can view the VM full screen once it has started if you wish.

BlueJ is available on the VM from the Programs item in the Start menu.

You may also want to install BlueJ on your own computer for your personal use. BlueJ is free to download from **www.bluej.org**, and runs on Windows, MacOS and Linux. You will probably need to download and run the version of the BlueJ installer that includes the JDK (Java Development Kit – you'll learn more about that later in the module) as that is not usually present on Windows or MacOS by default.

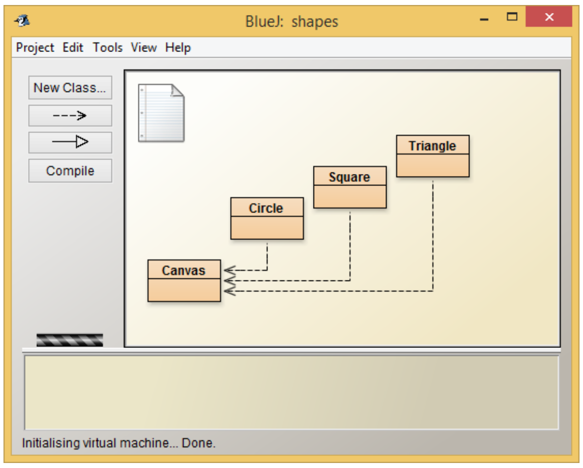
Most lab exercises will require you to download starter code from GCULearn. In most cases the code will be in the form of a ZIP archive containing one or more BlueJ projects. Once you have downloaded the appropriate ZIP archive you can extract the contents by right-clicking and selecting **7-zip > Extract files**... from the pop-up menu. You can then start BlueJ and use the **File>Open Project** option to navigate to the folder where you extracted the project and open it.

Note that no hard copies of lab tasks will be given out, so you will need to access them through GCULearn while working in the labs. You can do so in the same VM as you are running BlueJ. Alternatively you can view the lab task on the host PC, switch between host and VM as required.

Task 1 : Creating and manipulating objects

Getting started

1. Start BlueJ and use the **File>Open Project** option to open the project *shapes*. You can find this project in the examples folder inside the BlueJ application folder. On the lab VMs this is *C:\Program Files (x86)\BlueJ\examples*. Alternatively, you can download the code from GCULearn.



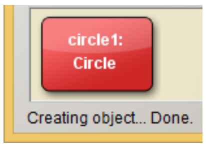
Initialising virtual machine... Done.

   When the project is opened, BlueJ should look like the screenshot above. There are four **classes** shown in the main area of the BlueJ window.

2. You may see the classes cross-hatched. This means that they need to be compiled before use. Click the Compile button to compile all the classes. You should ignore any warning dialog that is shown[1].

Creating objects

3. Right-click on the **Circle class** and select *new Circle()* from the menu. Accept *circle1* as the name of the instance in the dialog which is shown. You should now see a **Circle object named circle1** in the Object Bench.



Creating object... Done.

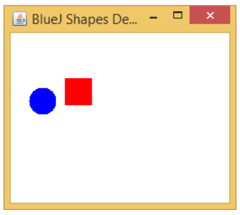
4. Similarly, create a **Square object** in the Object Bench with a suitable name.

[1] You can switch off this warning by selecting Tools>Options>Preferences, choose Miscellaneous tab, and uncheck "Show compiler warnings when….."

5. Right-click on *circle1* and choose *Inspect* from the menu. You should see the Object Inspector dialogue which shows the **fields of the object**.

   Q1.1 What colour is the circle? What colour is the square?

6. You can see the objects in the Object Bench, but now you will get each one also to create a **visual representation** of itself. Call the *makeVisible* method of each object – to call a method you right-click on the object and select the method name from the menu. You should see a new "picture" window that displays the representations of the shapes.



Calling methods

7. Call the *moveDown* method of *circle1*.

   Q1.2 What change do you see when you inspect the object? What happens in the picture?

8. Call a method or methods to move the circle object so that it lines up directly below the square.

   Q1.3 What method(s) did you use? How many times? Could you have achieved this without the picture being visible?

9. Call the *moveHorizontal* method of *square1*. This method needs some additional information – how far you want to move. Enter a value of 100. Observe the effect by inspecting and on the picture.

10. Move the square 10 pixels down and 50 pixels to the left, with one method call for each.

    Q1.4 What method(s) did you use? How did you specify the direction (up or down, left or right)?

11. Call the *changeColor* method of *circle1 (note US spelling of color!).* Enter the value "green" and observe the effect.

Q1.5 What is the data type of the parameter for this method call? What happens if you enter green, without quotes? What happens if you enter a number, again without quotes?

Creating more objects

12. Create another circle object, named *circle2*, in the Object Bench, and make it visible in the picture.

    Q1.6 How many Circle classes do you see in BlueJ? How many circle objects? How many circle objects do you think it would be possible to create?

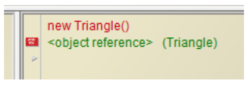
13. Inspect the objects in the Object Bench. Look at the field names and field values.

    Q1.7 What do the circle objects have in common? How do they differ? What makes a circle different from a square?

14. You can also create objects using code. Select the View>Show Code Pad option. The Code Pad area is displayed to the right of the Object Bench. Enter the following code in the Code Pad

    new Triangle()

    You should see the following. Drag the new triangle object to the Object Bench using the object symbol in the margin, and accept the instance name *triangle1*. Make the triangle object visible in the picture.

    ```
    new Triangle()
    <object reference>   (Triangle)
    ```

    Q1.8 What key word in code is used when creating an object?

Task 2 : Working with interacting objects

Getting started

1. Close the shapes project and open the project *picture*. You can find this project in the same *examples* folder inside the BlueJ application, or download from GCULearn. Compile the project if necessary.

   This project contains the same shape classes as the previous one, but there is also an additional class *Picture*.

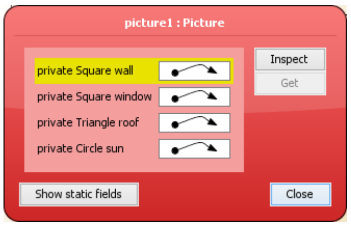
Creating interacting objects

2. Create an instance of the class *Picture* with name *picture1* in the Object Bench.

3. Call the draw method of *picture1*. You should see a very simple picture of a house made up from simple shapes. These shapes are instances of the *Circle*, *Square* and *Triangle* classes you saw in Task 1.

   Q2.1 How many objects, including the picture object are there? How many objects did you directly create? How do you think the other objects were created?

4. Inspect the picture object in the Object Bench. Its fields are all instances of one or other of the shape objects.



picture1 : Picture

private Square wall
private Square window
private Triangle roof
private Circle sun

   Q2.2 How would you describe the relationship between the picture object and the other objects?

Calling methods

5. With the Object Inspector for the picture object open, select the square object that represents the wall of the house and click the Inspect button. You should see another Object Inspector, for that object.

   Q2.3 What colour is the wall (square) object?

6. Call the *setBlackAndWhite* method of the picture object. Inspect the picture object again, and from there inspect the wall object.

   Q2.4 What field of that object has changed? What method does a square object provide to change its colour? What method did you call? How did the square's method get called?

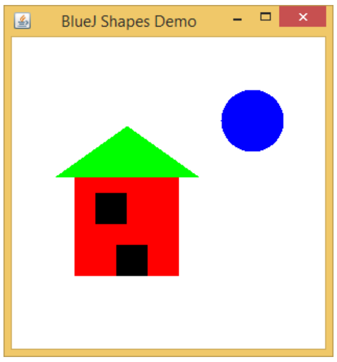
Task 3 : Modifying source code

In this final task you will try to make some changes to the Java code in the *shapes* project and observe the effect of the changes. You will learn much more about how to write code as you progress through the module, but you can try this task now if you are feeling adventurous.

1. Continue to work with the *shapes* project. Open the code editor for the *Picture* class (right-click on the class and select **Open Editor**). Look for the code that defines the colour of the sun in the picture. This code will be inside a block of code that defines a method called *draw*. Change the code so that the **sun will be blue** rather than yellow.

2. Compile the class by clicking the **Compile** button in the code editor, and **test your change** by creating a new picture object and calling its *draw* method, i.e. the method you have just modified. Note that every time you compile code, all existing objects are removed from the object bench and the picture window is closed.

3. Add a door to the house. This should be a square the same size and color as the window, and should be at the bottom of the wall, like this:

   To do this you will need to add a new line:

   ```
   private Square door;
   ```

   immediately after the similar line which declares the window field, and then add code into the *draw* method to create and position the door. **Test your change** - it may take some trial and error to get the position right.

4. Add code at the end of the *draw* method to make the sun set. It should move slowly downwards until it disappears from the picture. *Hint*: use the method *slowMoveVertical* of *Circle*. **Test your change** by creating a picture and calling the draw method.

5. Separate out the code to make the sun set into a new method *sunset* so that it can be called at any time. **Test your change** by creating a picture and calling the *draw* method and then the *sunset* method.