EECS 1710
LAB 2 :: Defining and using methods

__Prerequisite__ (Lab 0/Lab1) – please ensure that you have setup your EECS account, and can log into the lab machines prior to starting this lab, and you are familiar with submit/websubmit.

STEP 1: Importing and unzipping the starter code

Download and extract the following lab2 starter code:
http://www.eecs.yorku.ca/course_archive/2022-23/F/1710/labs/lab2/lab2.zip

There are two files to submit for this lab. Assuming the above zip file is downloaded and extracted into the `1710/labs/` folder in your home directory, you should be able to navigate to those folders and submit with the following commands:

```
cd
cd 1710/labs/lab2/lab2_q1/
submit 1710 lab2 lab2_q1.pde

cd
cd 1710/labs/lab2/lab2_q2/
submit 1710 lab2 lab2_q2.pde
```

This will submit each file independently. You may use websubmit to submit/delete submitted files (see link at the end of this document). To list files submitted, type:

```
submit -l 1710 lab2
```

STEP 2: Exercises

Question 1: (defining and using methods) [20 marks]

(a) No argument methods

Create a program (in file ***lab2_q1.pde***) that renders (draws) a simple, non-animated stick-figure object (e.g. stick figure, stick tree, or something similar (examples below).

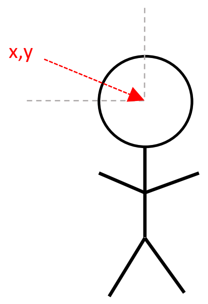
There are a number of approaches to doing this; for e.g., one possible algorithm (for a stick figure), could be:

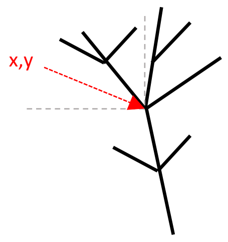
- setup:
  - Set the output window size;
  - Set the background colour;
- draw:
  - Call the method that renders the stick figure (see below), we will call this method `renderObject();`
- renderObject:

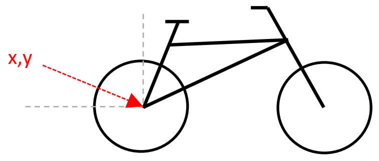
- Create a method called `renderObject()` that receives **no arguments**.
- Set local float variables x and y to the centre of the output (application) window; (i.e. variables are local if declared within the current method; variables are global if defined at top of sketch outside all other methods)
- Create a local variable called `radius` (the radius of the figure's head) to some initial (constant) value – e.g. 50;
- Draw the figure's trunk (body);
- Draw the figure's arms;
- Draw the figure's legs;
- Draw the figure's head;
- This method returns nothing (i.e., `void`) to its calling program.

It is suggested that you centre the figure's head/starting point at coordinates (x , y) and draw all the rest of figure relative to these coordinates. For example, the trunk is a single vertical line from (x , y + radius) to (x , y + radius * 3), or similar. **This will help parametrize the object in part (b).**

There is no requirement for colours to be used at this stage, nor for your object to be a stick figure. It could be some other kind of stick object (e.g. tree, car, building, face – try to keep it to a simple set of strokes/shapes that are easy to express in terms of the local variable you begin with in your method: (x , y). You can use the variable `radius` to control the lengths of lines. Here are some example stick objects (you need only create one):





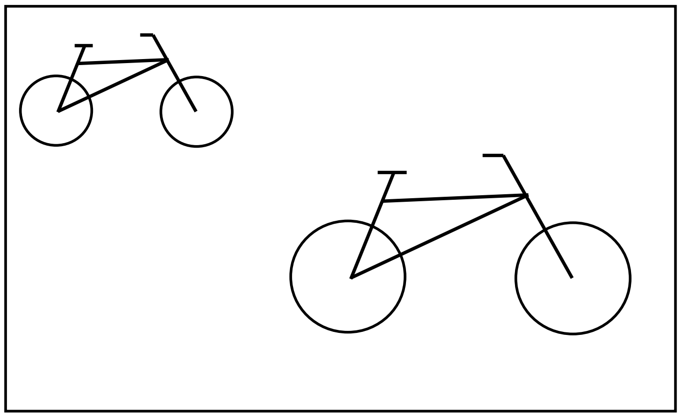


(b) Parametrized methods

In this part, you will add a **new** method of the **same name** (also called `renderObject`), that will parametrize some of the properties of your stick object from part (a). This method will include parameters/arguments for the two float variables you created in `x` and `y`, and a third for the `radius` variable. Again, this method will not have anything to return, so should be declared as `void`.

If you have created your part (a) by drawing each component in relation to `x,y`, then you should not really need to modify your code significantly from part (a) – only, remove the statements declaring and initializing the variables `x,y` and `radius` - you will now use those declared in your method header instead.

Include two new calls to your new version of `renderObject(..)` method, each with a different set of arguments (values passed to the method for x, y, and radius), so that two versions of the object are created on the application window (each at a different position, and with a different size). Both objects should have the same aspect ratio as one another (i.e. one should only differ from the other in terms of scale). Example output (using the 3rd stick bike object from previous example):



QUESTION 2: (create a generative crowd/park/landscape scene) [20 marks]

Create a program (in file ***lab2_q2.pde***) that will draw a scene with two layers (background and foreground). In the background you will have static elements, while in the foreground, you will generate/spawn several repeated parametrized elements (preferably using the methods you created in Question 1) to populate your scene.

Each time you click the mouse, your program should clear the foreground elements from the scene, and re-populate the scene with a pre-set number of randomly generated (positioned, coloured & scaled versions of your foreground object(s)).

The idea here is to re-use your parametrized `renderObject` method from Question 1, part (b). However if you would like to create a different method to draw something more exotic, you are free to do so. Make sure it is parametrized (in a manner similar to that described in question 1), so it draws relative to a starting position x,y and a size/radius.

There are two methods you will be expected to create (alongside the standard blocks you need for a dynamic sketch):

- `renderBackground() {   }` - populates a scene with some static background elements (these do not need to be parametrized). For instance, some shapes for clouds, sky, ground, mountains, skyline, etc. Depending on the scene you are creating.

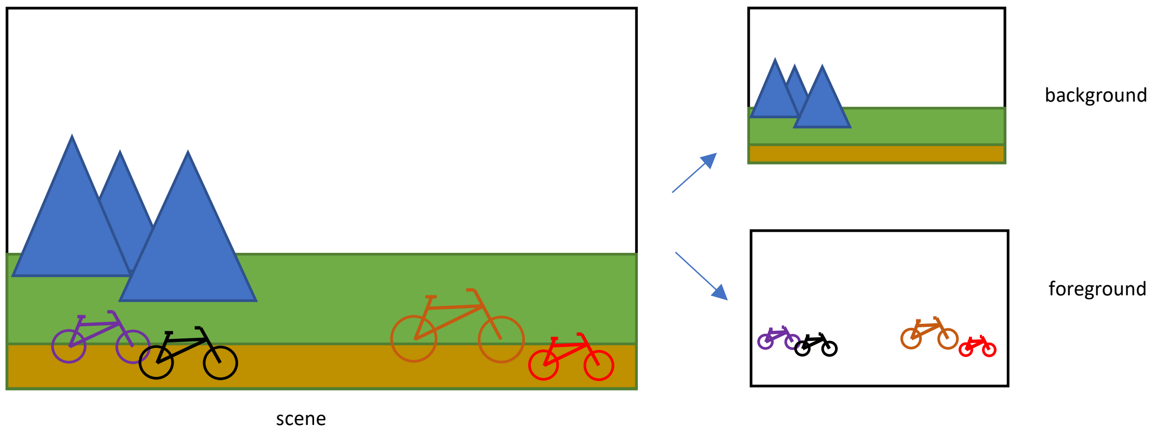
- `renderForeground() {   }` - populates a scene with a number of randomly positioned stick objects (from question 1, or if you prefer to design some new parametrized objects). Copy and paste your parametrized method from Question 1, into your code for Question 2 is the first step if you are re-using those methods (or write a new method to create a parametrized object).

Possible Algorithm:

- setup:
    - Set the output window size;
    - Call to renderBackground();
- draw:
    - Make a decision to draw a new foreground object or not
        - if yes, call renderForeground() with a randomly assigned set of position/size arguments and/or stroke/colour.
        - if not, don't call renderForeground();
    - Increment/Decrement a global variable to track the number of foreground elements left to render to the scene
- mousePressed:
    - clear the foreground elements by re-rendering the background scene
    - reset variables used to count number of foreground objects left to render

Your scene does not have to be anything too complex.

For example.. it may be a street scene – where there is a "crowd" of stick people, or a park, with a "forest" of stick trees… or a bike stand or bike trail, with a "bunch" of stick bikes (see figure below). The scene is up to you, however you must use only the draw() method to generate the foreground part of the scene (i.e. the part with repeated stick objects). Here is an example of a simple park/trail scene with a number of repeated stick bikes (4 objects in total).

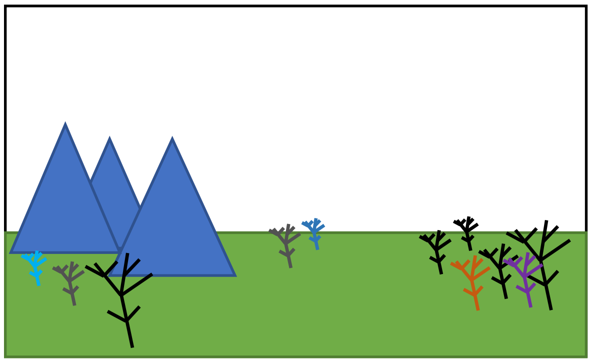


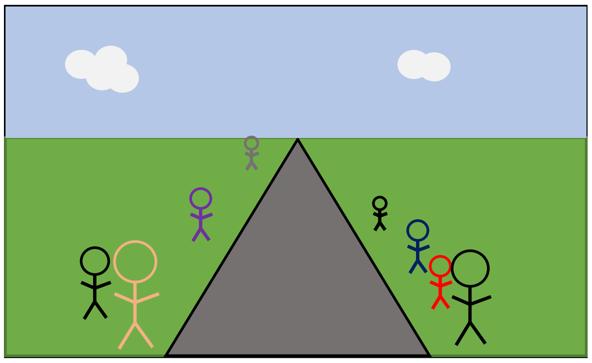
To do this, consider creating some variables that act as *counters* for the number of objects to draw, and some parameters to constrain where you want to restrict their random positioning/size. Alternatively, you may just explicitly make several separate calls to your renderObject method (each with different randomly assigned arguments for its parameters).

E.g. if using a counter (such as: `int numBikes = 4;`) inside the draw() {} block, you can keep drawing as long as your numBikes variable is not zero or less (use a relational operator/conditional expression here), and after each draw call, you can decrement/increment your counter to indicate the number of objects remaining. You will need to use an IF statement to decide if you need to keep drawing objects or not. Use the counter variables to test whether you have drawn enough of that object or not.

Add modifications to your code so that some property of the renderedObject is generated randomly. For example this may be as simple as the stroke colour (for the overall object), or it may be some slight change in geometry (e.g. the tree branches are slightly different, or the wheels on the bike are different sizes, etc). *If you are running out of time, keep this modification as simple as possible (e.g. make tree lean different angle)*

Other example scenes (based on the example stick objects outlined in question 1):





STEP 3: SUBMISSION (DUE: Tuesday Oct 4, 2022 – 5:00pm)

You will formally submit the ***.pde** files associated with lab2 -> Questions1-2

Submit each file from terminal:

```
cd
cd 1710/labs/lab2/lab2_q1/
submit 1710 lab2 lab2_q1.pde

cd
cd 1710/labs/lab2/lab2_q2/
submit 1710 lab2 lab2_q2.pde
```

Check submitted files:

```
submit -l 1710 lab2
```

NOTE: REMEMBER, you can choose to use the web-submit function (see Lab 0 for walkthrough). You will need to find and upload your lab1 files independently. Web-submit can be used to check your submission from the terminal, and can be found at the link: https://webapp.eecs.yorku.ca/submit/