CS 1133, LAB 5: SUBCLASSES

http://www.cs.cornell.edu/courses/cs1133/2013fa/labs/lab05.pdf

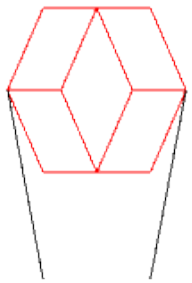
First Name: _____ **Last Name**: _____ **NetID**: _____

This lab demonstrates the power of subclasses, particularly in GUI applications. Subclasses are a great way to customize visual behavior of a GUI object. In addition to basic subclasses, you this lab will also give you some experience with the use of the `super` function.

This is another fairly short lab to finish out the course. In fact, you will find you spend more time reading the lab than doing it.

Lab Materials. We have created several Python files for this lab. You can download all of the from the Labs section of the course web page.

http://www.cs.cornell.edu/courses/cs1133/2013fa/labs



When you are done, you should have the following three files.

- `drawapp.py` (the primary script to run the application)
- `lab05.py` (the module that you must modify)
- `shapes.py` (a support module, which you will not touch)

You should create a *new* directory on your hard drive and download all of the files into that directory. Alternatively, you can get all of the files bundled in a single ZIP file called `lab04.zip` from the Labs section of the course web page.Open a command prompt and type

```
python drawapp.py
```

A figure like that on the right should appear in the window. Your command prompt will fill with the usual messages from Kivy. In addition, you should see some test ouput about the various shapes that are drawn in the window.

0.1. **Getting Credit for the Lab.** This lab will involve the modification of two of the above files: `lab05.py` and `drawapp.py`. There is no unit test for this lab. When you are done, simply show these two files your lab instructor. As always, you should try to finish the lab during your section, especially given how short this is.

This is the last lab in the last week of class. If you do not finish this week, you will need to schedule a time with the instructor after Fall Break to get credit.

Course authors: D. Gries, L. Lee, S. Marschner, W. White

1. Understanding the Application

Before you get started on this lab, we should first describe a bit how `drawapp` works. If you open it up you will notice that it contains two classes. The first is `DrawApp`; it is a subclass of the Kivy class `App` and has a single important method: `run()`. This `run()` method is called in the application code of this module, and is what opens up a new Window.

By itself, `DrawApp` just draws a blank window. The second class, `Panel` represents the contents of the Window, and it is responsible for drawing the figure above. When the application is built, a `Panel` object is "placed inside" the Window. The constructor for `Panel` calls the method `drawShapes()`, which draws the figure you see.

The method `drawShapes()` creates several objects, all of which are subclasses of class `Shape`. The `Shape` class, which is defined in the module `shapes.py`, contains information about the position of the shape, as well as its size and geometry. When the method `draw()` is called, the shape is added to the panel and remains there until you quit the application.

Like the Turtle in Aassignment 4, the values of `x` and `y`, as well as the side lengths, are given in pixels. In this application, the pixels are indexed by integer coordinates *starting at the bottom left corner*, as follows:

```
...
(0,2)  (1,2)  (2,2)  ...
(0,1)  (1,1)  (2,1)  ...
(0,0)  (1,0)  (2,0)  ...
```

In a pixel coordinate (x,y), x is the horizontal coordinate; it increases from left to right. Similarly, y is the vertical coordinate; it increases from bottom to top.

All of the shapes in this lab are subclasses of the class `Shape`. The class `Shape` has several attributes (many of which it inherits from `Widget`, a Kivy class) that position it correctly inside of the `Panel`. However, the primary method for drawing a shape is called `draw()`. You will notice that the body of this method in `Shape` says `pass`; which means it is undefined. This is why we never create an object whose type is simply `Shape`. All of the drawing is done in the subclasses, which are listed below.

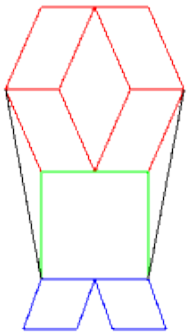
Class	Subclass Of	Description
`Parallelogram`	`Shape`	Four sided shape where each pair of opposing sides is parallel.
`Line`	`Shape`	Line segment between two points.
`Rhombus`	`Parallelogram`	Parallelogram where all four sides are equal length.
`Square`	`Rhombus`	Rhombus where all angles are right angles.

You should now look at the documentation of `Parallelogram` (which is in `shapes.py`) to see how it works. You can either look at the source code directly, or import the module and use the `help()` function as shown in class. You should pay particular attention to the concept of the "leaning factor" in the definition of a parallelogram.

2. Lab Instructions

This lab is broken up into two tasks, the first one being quite short.

Task 1: Complete the Constructors. The image in the window is incomplete. Only `Parallelogram` and `Line` draw properly. That is because the constructors in `Rhombus` and `Square` are unfinished. If you open the `lab05.py` file, you will see that both of their `__init__` methods are empty.

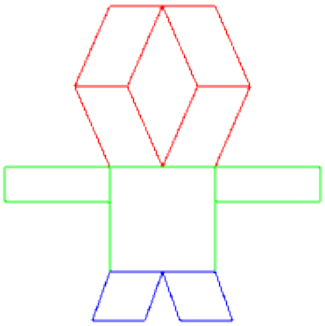


This is unacceptable. At the very least, each of these subclasses must call the `__init__` method in their parent class as a helper function. In this case, this is all we need. The class `Parallelogram` does all the work for drawing, so we just need to be sure to (eventually) invoke its `__init__` as a helper.

We showed how to do this in class. You simply call the method `__init__` as an explicit function in the parent class. If you are unclear on how to do this, open up `shapes.py` and look at the class `Line`. See how it works?

Use this technique to complete the constructors of `Rhombus` and `Square`. Each constructor should be a single line using and explicit call to `__init__` in the parent class. When you are done, run `drawapp.py` as a script again. You should now get the image on the right.

Task 2: Add Some Arms. The shape above looks almost like a person. It has a head (the `Parallelograms`), a body (the `Square`), and some legs (the `Rhombuses`). You will give the person arms. All the changes you will make will be in class `Panel` in module `drawapp.py`. Read through method `drawShapes()` of `Panel`.



But before you do anything else, you should comment out the code that produces the two black lines (in DemoShapes).

Hint: Look for where the color is set to black.

Each arm is should be a green rectangle that is 60 pixels long and 20 pixels high. Its leaning factor (field d of class Parallelogram) is 0, which means that it is a rectangle. The leaning factor is defined in the specification of `Parallelogram`. Later, when you get the program going with leaning factor 0, you can try a different leaning factor, say 15, and see what it looks like.

The arms should be attached at the top right and top left of the square that makes up the body. The tops of the arms should be parallel to the top line of the square. When done, it should look like the image to the right.

In writing the code that draws rectangles, use the variables that are defined at the top of method `drawShapes()`. Use variables to contain all the constants that you need, as we did in method `drawShapes()`. You should avoid using numbers directly in the constructor `Parallelogram`.

Hint: To determine the arm coordinates, look at the position of the green square. For debugging purposes, you may want to include print statements for the objects you create, as we did.

When you are finished, show `drawapp.py` and `lab05.py` to your instructor.