CSci 1133, Spring 2015
Lab Exercise 12

Simple Inheritance and Polymorphism

Graphical programs are natural applications for object-oriented programming. In this lab exercise, you will construct a number of classes to manage objects on a graphical display. We'll begin by extending our working knowledge of the built-in Turtle Graphics interface.

Warm-up

Using Python in interactive mode, start by constructing a Turtle object using the class constructor:

```
>>> import turtle
>>> myt = turtle.Turtle()
```

The "turtle" is the object that is represented by the little symbol and does the "drawing". For some of the more advanced functions, we also need to interact with the "window" object on which the turtle is drawing. This is referred to as a "screen" in turtle lingo, and we construct one using the following:

```
>>> scr = turtle.Screen()
```

To enable animation (moving graphics objects), the screen has an instance variable that delays the graphics operations by some number of milliseconds. We can see what the default delay is by calling the screen's `.delay()` method without any arguments:

```
>>> scr.delay()
```

If we are going to use turtle graphics to do something a bit more sophisticated, we want it to run as quickly as possible, so we can set the delay to zero by providing an argument to the `.delay()` method:

```
>>> scr.delay(0)
>>> scr.delay()
```

In a similar way, the turtle object has a default "speed" between 1 and 10 (slow to fast) which typically defaults to some slow value so the user can follow what the turtle is doing. Again, we want it to draw as quickly as possible, so we'll speed it up using the `.speed()` method of the turtle class. Calling the `.speed()` method with no arguments will return its current value:

```
>>> myt.speed()
```

To change it, we provide an integer value from 0-10. Although 0 is the lowest number, it will cause the turtle to go at *maximum* speed:

```
>>> myt.speed(0)
>>> myt.speed()
```

Although the little turtle symbol is cute, we won't need "cute" anymore… to remove the turtle symbol, we can use the `.hideturtle()` method:

```
>>> myt.hideturtle() [ENTER]
```

Nice. The "turtle" is still there, we just made it invisible…

You already know how to draw lines and turn the turtle, but a number of other objects can be drawn using turtle methods in one step. Circles are one example, complex polygons are another. First, move the pen (turtle) to a specific x,y location on the screen using the `.goto(x,y)` method. Note: to avoid drawing a line as you move you first need to "lift" the pen off the "paper". You do this using the `.penup()` method:

```
>>> myt.penup()
>>> myt.goto(40,40)
```

Now, let's draw a circle with radius = 100 (pixel units):

```
>>> myt.circle(100)
```

Huh?? Where's our circle? Oops… we forgot to put the pen back down! Let's try it again:

```
>>> myt.pendown()
>>> myt.circle(100)
```

Voila! We have a circle at 40,40. But it isn't very colorful… In fact, it's *transparent*. The only thing that shows is the outline of the circle. In order to have a colored circle we have to do a few more things, including filling it with a color. Creating a "filled" color object, requires four steps in turtle graphics:

1). Set the fill color to what ever you want ("red", "blue", "yellow", etc.)
2). Indicate the start of a sequence of drawing operations that will create an "enclosed" figure
3). Perform the sequence of drawing operations
4). Indicate the end of the drawing sequence, and fill the object with the fill color

It's easier to do than describe! Let's redraw our circle filled with red:

```
>>> myt.fillcolor("red")
>>> myt.begin_fill()
>>> myt.circle(100)
>>> myt.end_fill()
```

We now have a nice red circle. Note that the `.fillcolor()` method takes a string. Most of the obvious colors are represented as you might think ("red", "blue", etc.). Let's do it again with a rectangle. There are turtle methods for drawing polygons that you might like to explore on your own. For now, we'll just use the line drawing methods we've used in the past:

```
>>> myt.penup()
>>> myt.goto(-50,-50)
>>> myt.pendown()
>>> myt.fillcolor("blue")
>>> myt.begin_fill()
>>> for i in range(4):
        myt.forward(100)
        myt.right(90)
>>> myt.end_fill()
```

There are a number of interesting turtle methods that you should try. You can find a decent summary in Appendix C of your textbook. If you'd like a complete description of the entire module, consult the "official" Python reference online: `http://docs.python.org/3.2/library/turtle.html`

Functions are Objects!

Yes, its true. Function names in Python are object references and can be used in lists, dictionaries and even as arguments in other functions. In order to provide input to our graphics programs, we need to exploit this capability to provide our own function that will be called when a mouse "click" occurs.

Mouse tracking and input ("clicking" a mouse button) are handled *asynchronously* to your program's operation. When the user "clicks" a mouse button, Python calls a function that you provide and passes the x and y coordinates of the mouse location as they were when the button was pressed. Turtle graphics "knows" what function to call because you "tell" it by giving the name of the function as an argument to a screen method, `.onclick()`. Again, it's probably easier to demonstrate than describe. First, write a short function definition:

```
>>> def mouseInput(x,y):
        print(x,',',y)
```

The `mouseInput` function will be called when a button is "clicked" by the user. Note that it *must* have two arguments for the x and y coordinates of the mouse location corresponding to the button press (Note that if this is a *member* function, you will also need to supply the "self" argument as always). In this case, we are simply printing out the x and y coordinate values on the terminal.

Next, we have to instruct the turtle screen object that the `mouseInput()` function is the one that should be called when the mouse button is clicked. We do that by registering the function using another *screen* method: `.onclick()`

```
>>> scr.onclick(mouseInput)
```

Note that the single argument to the `.onclick()` method is the name of the function that we've written to process mouse button-press events. The method just "sets up" the mechanism, we have to do one more thing to actually make Python start processing button presses. It involves one more *screen* method that takes no arguments:

```
>>> scr.listen()
```

Position the turtle window and the python shell so that you can see both. Now move the mouse around in the turtle window and press the left button on the mouse and watch the output in the python shell display. You should see x and y values printed each time you press the button.

You are now ready to construct more sophisticated graphics programs. Let's use objects and inheritance to do something fun.

Stretch

1). **Random Colors**

We are going to need a function that will return a random color value (string) from the following colors:

["red", "yellow", "green", "blue", "purple", "orange"]

Write a function named `randColor` that takes no arguments and randomly returns one of the colors in the list. [Hint: use `random.randint` to generate a suitable random index value]

2). **Shape Class**

Construct a base class named `shape` that will maintain common "shape" information such as location, fill color, etc. The shape class should have instance variables for the x and y location of the shape (type int), the fill color (type str) and a boolean indicating if the shape should be filled or not. The location should default to 0,0. The fill color should default to "" (Null string) and filled should default to False. Provide the following `shape` methods:

- `setFillcolor(str)` Mutator to set the fill color to the value of the string argument
- `setFilled(bool)` Mutator to set the filled boolean to the value of the boolean argument
- `isFilled()` Accessor that will return the value of the filled instance variable

3). **Circle Class**

Now, create a new class named `circle` that inherits the `shape` base class. The constructor for the circle class should have arguments for the x and y location (default to zero) and the radius of the circle (default to 1). Be sure to properly call the super-class constructor! Provide the following `circle` methods:

- `draw(turtle)` An accessor method that will draw a circle using turtle graphics. `draw()` will take a single turtle object as an argument and draw the circle at its specified x,y location. If the circle is "filled", be sure to implement the proper sequence of operations to draw a filled circle of the appropriate color. Otherwise draw an unfilled circle.

Test your two class definitions by creating and drawing several circles

Workout

1). **Graphics Display Object**

Create another class named `Display` that will maintain a graphics environment involving shapes and mouse input. The constructor for the Display class will instantiate a number of instance variables and set up the turtle environment to enable the input of mouse button selection events. Be sure to import the turtle module in your class definition file.

The Display constructor should do the following:

- Initialize a turtle object member using the Turtle() method
- Initialize a screen object member using the Screen() method
- Initialize an instance variable named `elements` (a null list)
- Set the speed and delay of the turtle and screen to zero and hide the turtle

Create a `Display` *method* named `mouseEvent(x,y)` that will process mouse button presses. For this version of the program, the `mouseEvent()` method will do the following:

- Create and draw a `circle` object at the x,y location indicated by the mouse press (passed as arguments to the function). The `circle` should be filled and have a random radius between 10 and 100. It should also have a random color as determined by the `randColor()` function created in the Stretch exercise.

You will also need to add statements to the `Display` constructor to register the `mouseEvent()` method as the "on-click" routine (don't forget to enable it using the `listen()` screen method!)

If you've done this correctly, you should be able to simply instantiate a Display object, then use your cursor to roam about the window. Each time you press the mouse button, a randomly sized/colored circle should appear at the mouse location.

Challenge

Congratulations! You've created a fairly sophisticated graphics program using the simplest of all graphics interfaces: turtle graphics.

Use your imagination and modify the program to do something creative and interesting!