Week 14 - Monday

COMP 1800

Last time

- What did we talk about last time?
- Inheritance

Questions?

Assignment 10

Inheritance

Inheritance

- The idea of inheritance is to take one class and generate a child class
- This child class has everything that the parent class has (members and methods)
- But, you can also add more functionality to the child
- The child can be considered to be a **specialized** version of the parent

Creating a subclass

- All this is well and good, but how do you actually create a subclass?
- Let's start by writing the **Vehicle** class

```
class Vehicle:
  def travel(self, destination):
    print('Traveling to', destination)
```

Extending a superclass

- We use put the superclass name in parentheses when making a subclass

```
class Car(Vehicle):
  def __init__(self, model):
     self.model = model

  def getModel(self):
     return self.model

  def startEngine(self):
     print('Vroooooom!')
```

- A **`Car`** can do everything that a **`Vehicle`** can, plus more

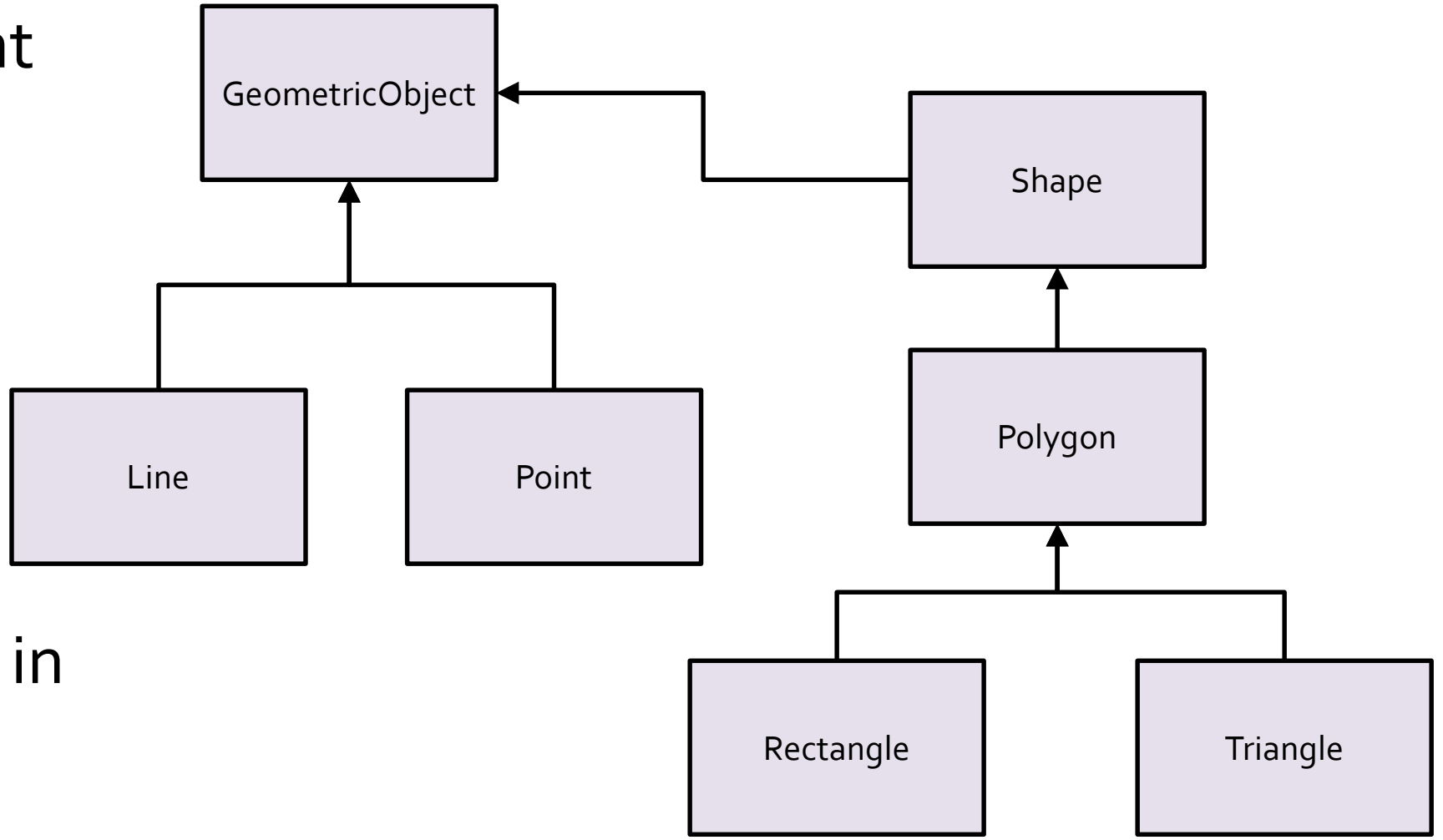
Shapes

Inheritance hierarchies

- In large, object-oriented systems, it's common for there to be many classes with many children (and grandchildren, and great-grandchildren…)
- This kind of arrangement is called an **inheritance hierarchy**
- Using UML, we can draw inheritance relationships between classes with arrows
- Although it is counterintuitive, the UML standard is for the arrow to point from the child to the parent

Shapes

- Drawing different kinds of shapes can be a useful task for inheritance
- Consider the following inheritance hierarchy shown in UML



Drawing shapes

- The classes shown in the previous slide have an inheritance relationship with **`GeometricShape`**
  - The *is-a* relationship, since each of those shapes is a **`GeometricShape`**
- We also need a place to draw those shapes
- We can create a **`Canvas`** class to draw them
- A **`Canvas`** is *not* a **`GeometricShape`**
- Instead, it provides a turtle that **`GeometricShape`** objects can use to draw themselves

Canvas class

- Since it's not important to the inheritance hierarchy, here's the code for **`Canvas`**
- It sets up a turtle and a screen in its constructor
- It also handles the turtle in the draw code

```
class Canvas:
    def __init__(self, w, h):
        self.turtle = turtle.Turtle()
        self.screen = turtle.Screen()
        self.screen.setup(width = w, height = h)
        self.turtle.hideturtle()

    def draw(self, shape):
        self.turtle.up()
        self.screen.tracer(0) # animation off
        shape.draw(self.turtle)
        self.screen.tracer(1) # animation back on
```

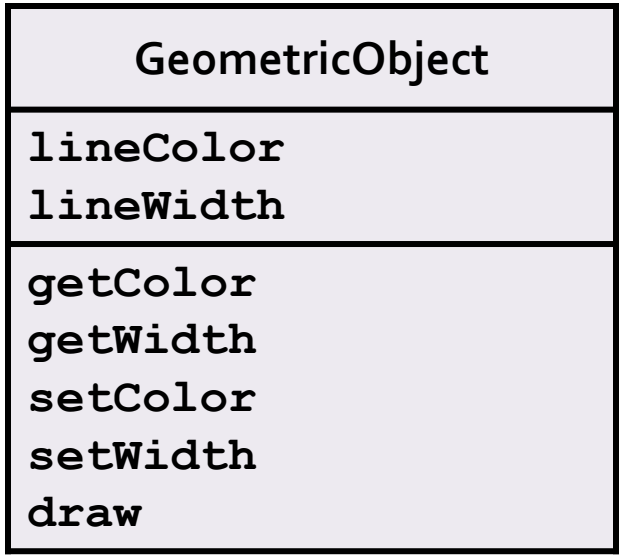
One final bit of Python syntax

- You can't have a function (or an **`if`** statement or a loop) with nothing in it
- For these rare circumstances, there's a special keyword that means do nothing
  - The **`pass`** keyword

```
def doNothing():
  pass # would have errors otherwise
```

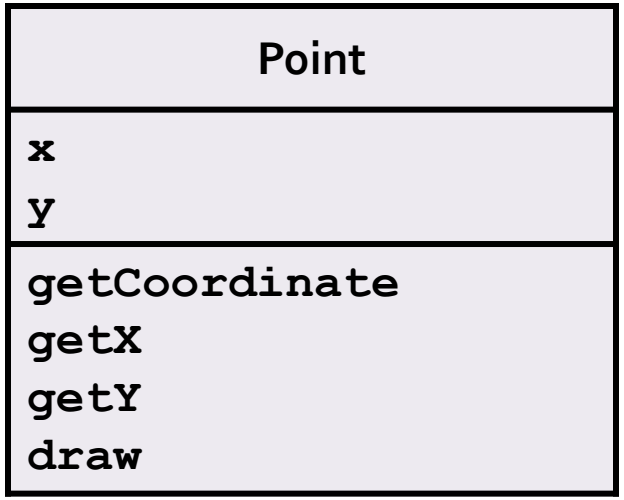
GeometricObject class

- Use the UML diagram to create the **GeometricObject** class
- The **draw()** function should do nothing
  - Use **pass**!
  - It takes in a **turtle** as well as **self**
- The constructor should:
  - Set **lineColor** to **'black'**
  - Set **lineWidth** to **1**
- A **GeometricObject** will give us the basic code for setting the color and the width of the lines we'll draw in child classes



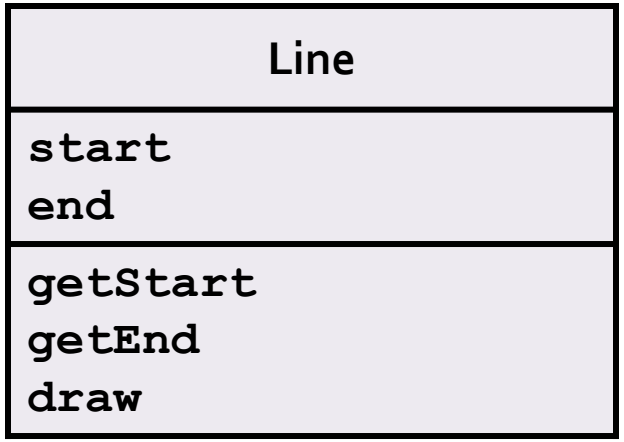
Point class

- Use the UML diagram to create the **Point** class
  - Remember that **Point** is a child of **GeometricObject**
  - Its constructor takes an *x* and a *y* (and calls the **super()** constructor)
- The **getCoordinate()** function gives back a tuple containing *x* and *y*
- The **draw()** method will:
  - Go to the given location with the turtle
  - Use the turtle's **dot()** method to draw a point
    - It takes a size (the width) and a color



Line class

- Use the UML diagram to create the **Line** class
  - Remember that **Line** is a child of **GeometricObject**
  - Its constructor takes two **Point** objects (**start** and **end**) (and calls the **super()** constructor)
- The **draw()** method will:
  - Set the turtle's color
  - Set the turtle's width
  - Go to the starting point
  - Put the turtle's tail down
  - Go to the ending point



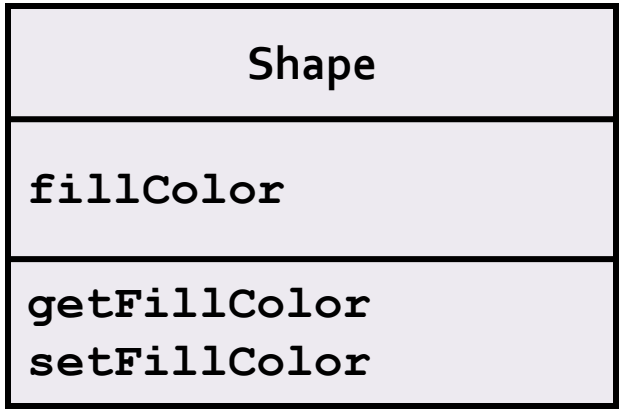
Using what we have

- Now we can draw a line using the classes we have
- The following code will create a red line with a thickness of 2, from (-100, -100) to (100, 100)

```
canvas = Canvas(500, 500)
line = Line(Point(-100,-100), Point(100, 100))
line.setWidth(2)
line.setColor('red')
canvas.draw(line)
```

Shapes

- In addition to points and lines, we could have polygons
- The turtle module allows us to create polygons that are filled in
- Thus, we can add another class that inherits from **`GeometricShape`**, adding a fill color
- Use the UML diagram to create the **`Shape`** class
  - Remember that **`Shape`** is a child of **`GeometricObject`**
  - Its constructor sets its fill color to **`None`**

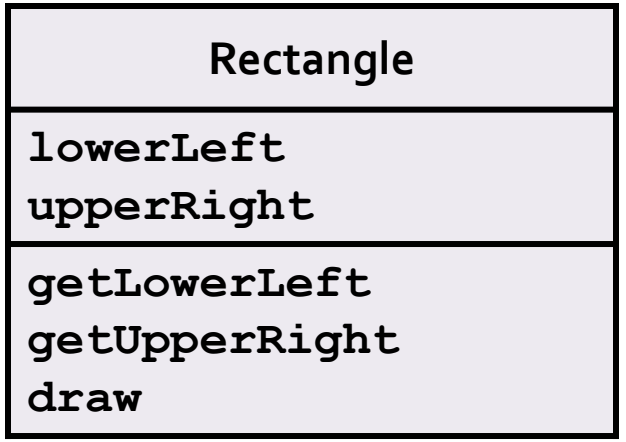


Polygons with turtle

- To make a polygon with the turtle module, you have to do the following steps:
  - Set the turtle's color to the color you want to fill the polygon
  - Go to the starting corner of the polygon
  - Call the **`begin_fill()`** method on the turtle
  - Visit all the corners of the polygon, returning back to the starting point
  - Call the **`end_fill()`** method on the turtle
- Important: You have to visit the points on the polygon in counterclockwise order
  - Otherwise, it might fill your shape incorrectly

Rectangle class

- Use the UML diagram to create the **Rectangle** class
  - Remember that **Rectangle** is a child of **Shape**
  - Its constructor takes two **Point** objects (**lowerLeft** and **upperRight**) (and calls the **super()** constructor)
- The **draw()** method will use the approach described on the previous slide to fill in the rectangle



Moving on from here

- The book describes ways for the **`Canvas`** to keep a list of **`GeometricShape`** objects
- When one of them is changed, it can clear the screen and redraw everything, keeping everything updated
- By extending **`Shape`** with other classes, you could make the following classes:
  - **`Ellipse`**
  - **`Circle`**
  - **`Triangle`**
  - **`Square`**
  - Even more …

Quiz

Upcoming

Next time…

- No class Wednesday or Friday because of Thanksgiving
- Next Monday we will review up to Exam 1

Reminders

- Work on Assignment 10
  - **Due next Friday**
- Review chapters 1 through 4