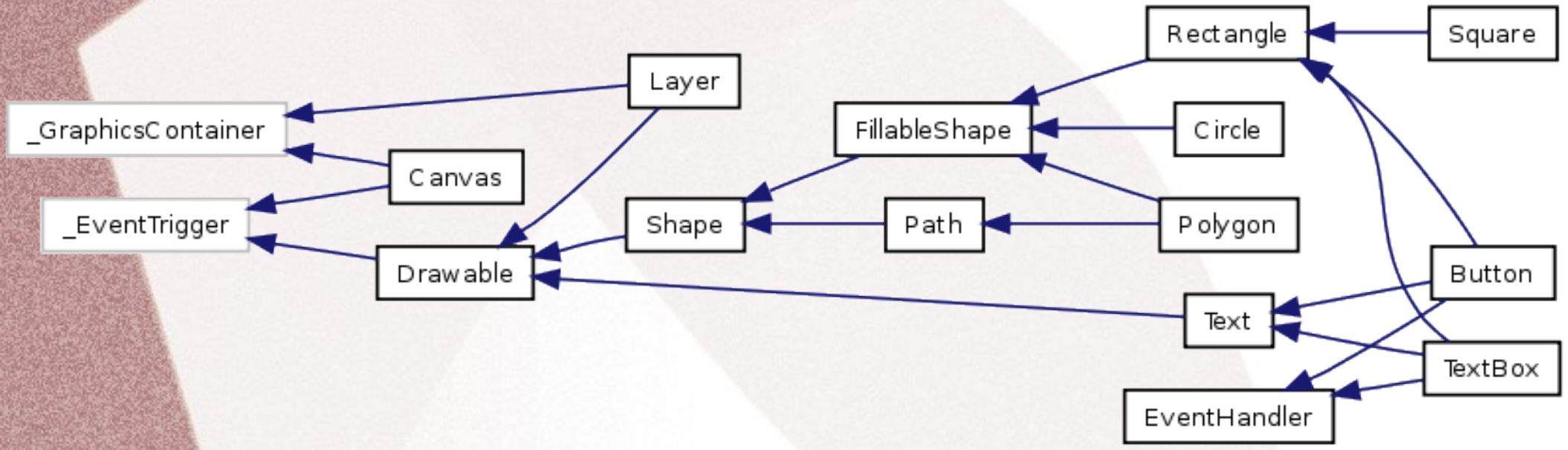
# Lecture 13

- Chapter 9 Inheritance

9.4 Class Hierarchies and cs1graphics

# 9.4 Class Hierarchies and cs1graphics

Here is the rich hierarchy in cs1graphics module.




This graph can be found on Documentation page of www.cs1graphics.org

# 9.4 Class Hierarchies and cs1graphics

We can recall a 'star' (defined as a polygon) that I used in some of the examples while we were covering Chapter 3.

Why don't we try to design a **Star** class (new drawable class)?

Pros: no need for future users to deal with geometry of stars, *(i.e. all the user will need is to say how many rays he/she wants the star to have)*

Since a star is a special case of a **Polygon**, let the Polygon be a parent class of the Star class.

```
class Star(Polygon)
```

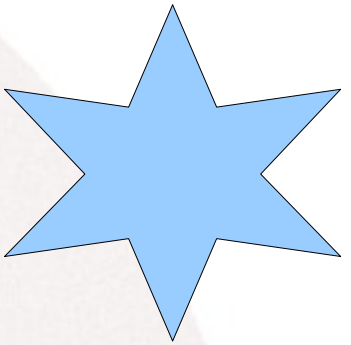
we will inherit: many useful behaviors (like *drawing* a polygon, *adjusting the border*, *fill color*, *moving*, *scaling*, and so on).

# 9.4 Class Hierarchies and cs1graphics

Our biggest challenge will be implementing a constructor that establishes the initial geometry of a star.

4-point star

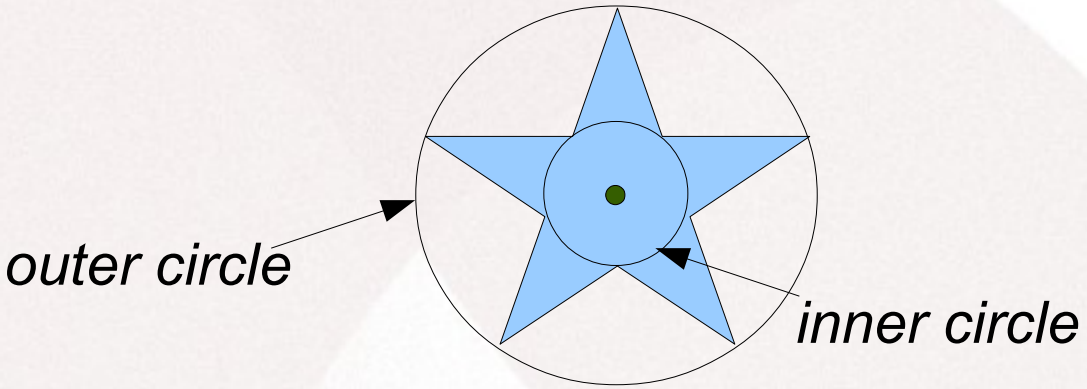
5-point star



6-point star

# 9.4 Class Hierarchies and cs1graphics

Our biggest challenge will be implementing a constructor that establishes the initial geometry of a star.




5-point star

A star with n rays has 2n points.
Also we have two radii (outer and inner):
outer - measured from the center of the star to the tips of the rays,
inner - measured from the center of the star to the place where two neighboring rays meet.

# 9.4 Class Hierarchies and cs1graphics

A decision:

Let's allow the user to specify the *outer radius* and the *ratio between the inner and outer radii*.

The center of the star will be the reference point, and we can allow the user to specify the point on the Canvas to attach the star to.

So, the constructor will be:

```
__init__(self,numRays=5,outerRadius=30,
         innerRatio=0.5, center=Point(0,0))
```

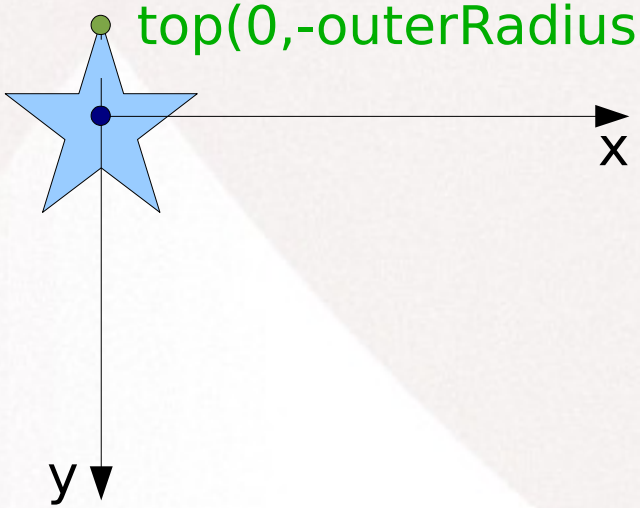
$$InnerRatio = \frac{(inner\ radius)}{(outer\ radius)} < 1$$

The smaller the ratio (closer to 0), the “thinner” the star,
The larger the ratio (closer to 1), the “puffier” the star.

# 9.4 Class Hierarchies and cs1graphics

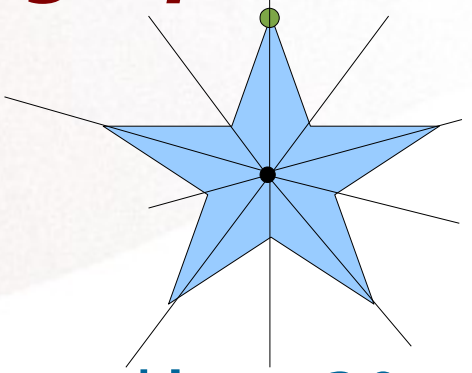
*We will put the top point of the star outerRadius units above the origin.*

*And we'll be building a star as if it is centered around the origin*




There is a reason for centering the star around the origin: we will see it later.

# 9.4 Class Hierarchies and cs1graphics



Let's begin to build the class Star:

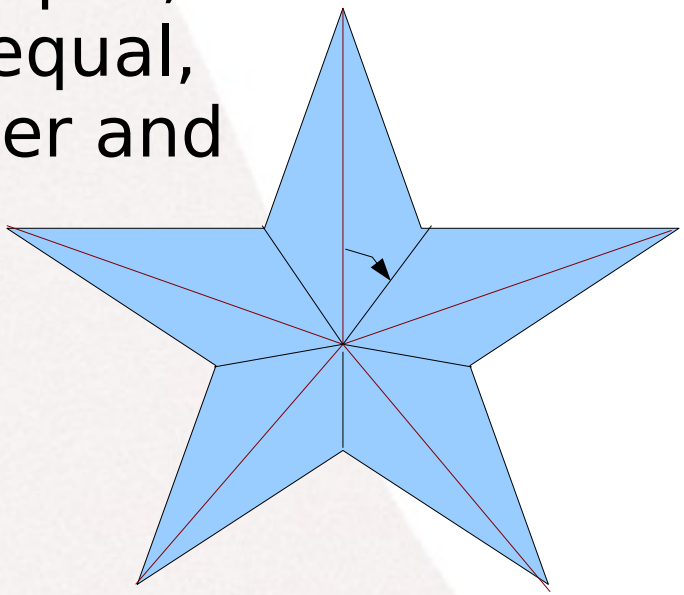
```
class Star(Polygon):
 def   __init__(self,numRays=5,outerRadius=30,
                innerRatio=0.5,
center=Point(30,30)):
   Polygon.__init__(self) # call the parent
constructor, a polygon with 0 points is
created

   top = Point(0,-outerRadius) # top point of
the star
   print("Top point:",top)

   angle = 360.0/(2*numRays) # angle between
adj. points
   print("Angle:",angle)
```

# 9.4 Class Hierarchies and cs1graphics

to calculate the angle:

```
#angle between adjacent points
 angle = 360.0/(2*numRays)
print("Angle:",angle)
```

The angles between outer point are equal, the angles between inner points are equal, and the angles between adjacent outer and Inner points are equal too.



Therefore each small angle is $\frac{360}{(2*n)}$, where $n$ is the number of star rays.

Why do we need this?

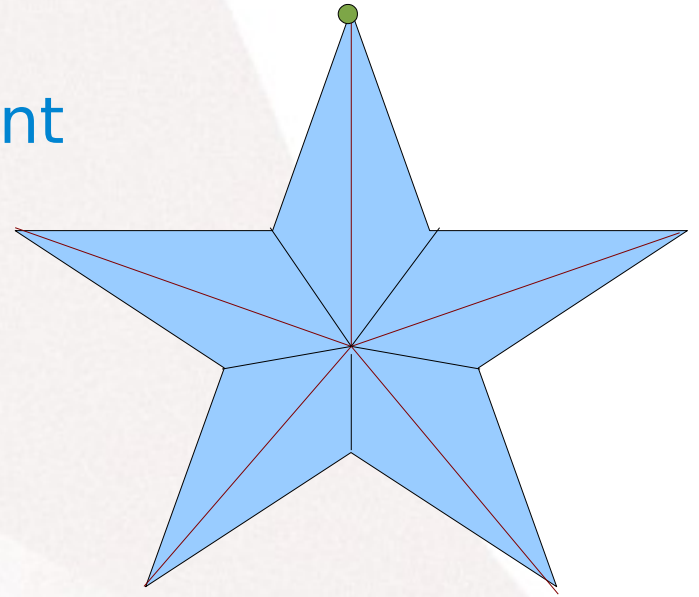
# 9.4 Class Hierarchies and cs1graphics

Answer:
We'll start with the top point, and will be calculating all the other points by dully rotating and scaling vector, represented by the origin and the top point.

For this will use operations * and ^:

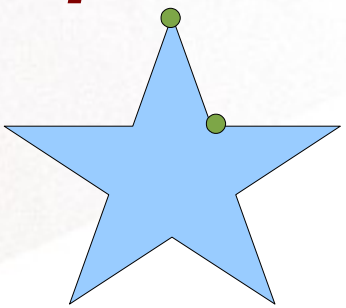
Point * n - multiplies vector with the Point as the end-point by the scalar n

Point^angle – rotates the vector with Point as the end-point of the vector to angle degrees clockwise about the origin.



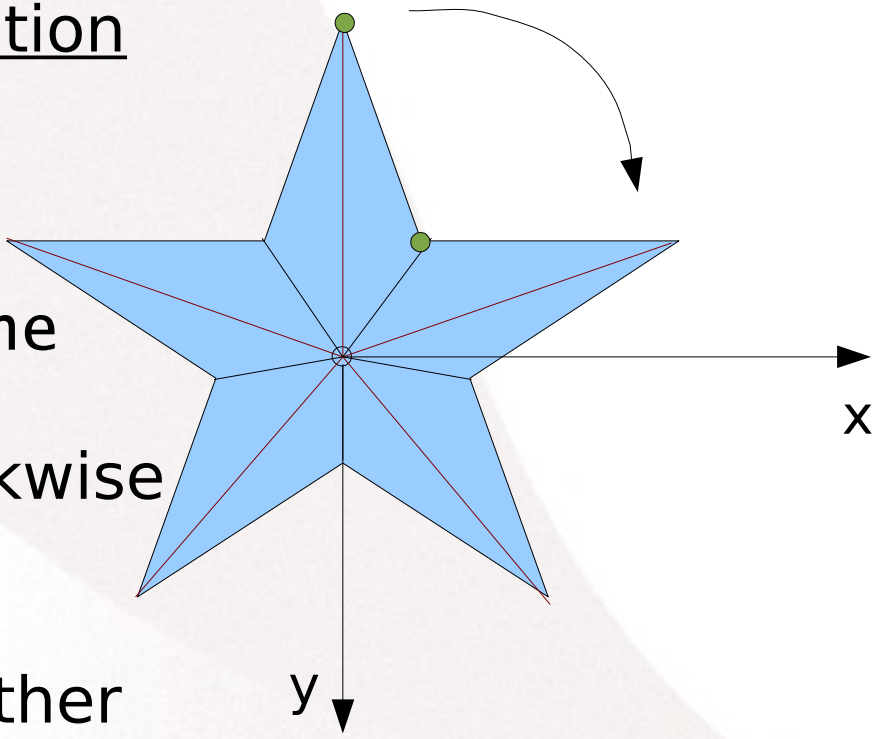

See methods of Point class:
__mul__ (self,operand) and
__xor__ (self,angle)
in the documentation of cs1graphics

# 9.4 Class Hierarchies and cs1graphics



Let's proceed:

```
for i in range(numRays):
    # adding ray point
    elf.addPoint(top^(angle*2*i))

    # adding inner point
    self.addPoint(innerRatio*top^(angle*(2*i+1)))
```




Two points are added at each iteration of this for loop – one *outer point*, one *inner point*.

And the reason for centering the star around the origin is that operator ^ rotates the vector clockwise **about the origin**.

See what happens if you choose other top point (most likely the star will be tilted)

# 9.4 Class Hierarchies and cs1graphics

Let's proceed:

```
# move Ref. Point from the top point
# to the center of the star
self.adjustReference(0,outerRadius)

 # move the star to the location given by
 # user, or to the default location
self.moveTo(center.getX(),center.getY())
```

# 9.4 Class Hierarchies and cs1graphics

One more addition:

We can allow user/programmer to change the innerRatio. In this case we need to create one more attribute: `_innerRatio`, and then write a method that will allow to update the innerRatio

```
# record the inner ratio as attribute
self._innerRatio=innerRatio

 def setInnerRatio(self,newRatio):
    factor=newRatio/self._innerRatio
     self._innerRatio=newRatio

     #we will modify only inner points
     for i in range(1,self.getNumberOfPoints(),2):
       self.setPoint(factor*self.getPoint(i),i)
```

# 9.4 Class Hierarchies and cs1graphics

See the program star_class.py

Then run a slightly improved version:

star_class2.py