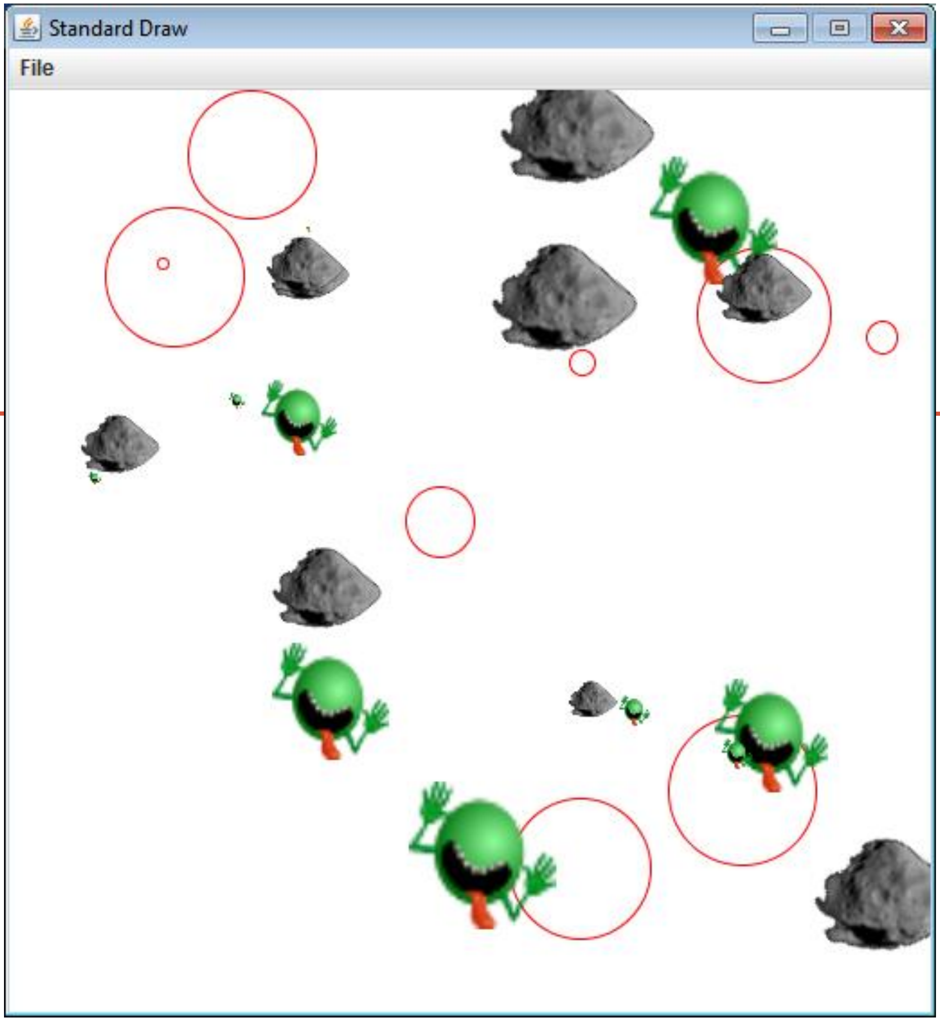
INHERITANCE AND OBJECTS



Outline

- Inheritance
  - Sharing code between related classes
  - Extremely common in modern OOP languages
- Managing many objects
  - Create class holding a collection of other objects
  - Let's you simplify your main program
  - Hides details of how you store things

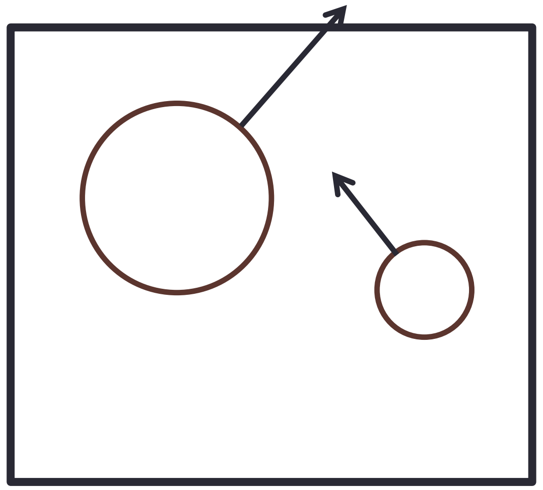
Inheritance

- One class extends another
  - Parent class: shared attributes/methods
  - Child class: more specific attributes/methods
    - Class declared extends the parent class

- Why? Lets you share code
  - Repeated code is evil

Inheritance Example

- Goal: Animate circles that bounce off the walls
  - What does an object know?
    - x-position, y-position
    - x-velocity, y-velocity
    - radius
  - What can an object do?
    - Draw itself
    - Update its position, check for bouncing off walls



Circle

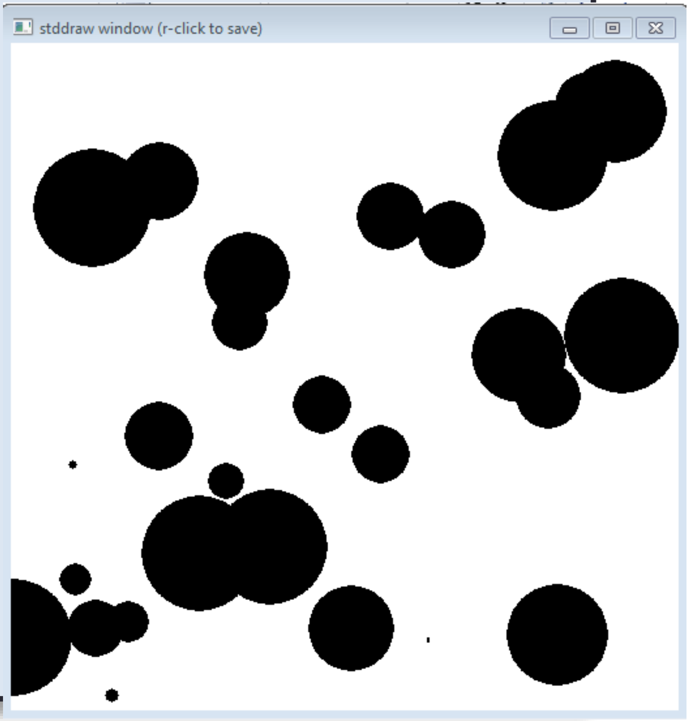
```
import color
import math
import StdDraw

class Circle:

    def __init__(self, x = 0.0, y = 0.0, r = 0.0):
        self.posX   = x
        self.posY   = y
        self.radius = r
        self.color = color.Color(0, 0, 0)

    def addToX(self, deltaX):
        self.posX += deltaX

    def overlap(self, other):
        deltaX = self.posX - other.posX
        deltaY = self.posY - other.posY
        d = math.sqrt(deltaX**2 + deltaY**2)
        if d < (self.radius + other.radius):
            return True
        return False

    def setColor(self, r, g, b):
        self.color = color.Color(r, g, b)

    def draw(self):
        StdDraw.setPenColor(self.color)
        StdDraw.filledCircle(self.posX, self.posY, self.radius)

    def toString(self):
        return "(" + str(self.posX) + ", " + str(self.posY) + ") r = " + str(self.radius)
```

Bouncing Circle Class

```
from Circle import Circle

class BouncingCircle(Circle):

    def __init__(self, x, y, vx, vy, r):
        super.__init__(x, y, r)
        self.vx = vx
        self.vy = vy

    def updatePos(self):
        self.x += self.vx
        self.y += self.vy
        if self.x < 0.0 or self.x > 1.0:
            self.vx *= -1
        if self.y < 0.0 or self.y > 1.0:
            self.vy *= -1
```

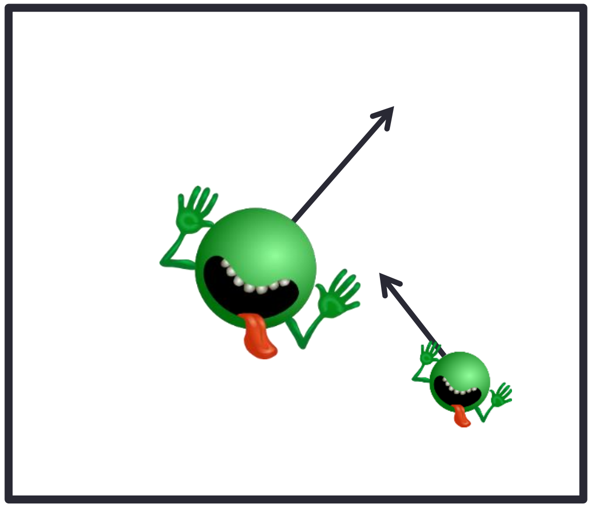
Bouncing Circle Client

```
from BouncingCircle import BouncingCircle as BC
import random
import StdDraw

circles = []

for i in range(0, 30):
    circles.append(BC(random.random(),
                      random.random(),
                      0.002 - random.random() * 0.004,
                      0.002 - random.random() * 0.004,
                      random.random() * 0.1))
while True:
    StdDraw.clear();
    for i in range(0, len(circles)):
        circles[i].updatePos()
        circles[i].draw()
    StdDraw.show(10)
```



Inheritance Example

- Goal: Add images that bounce around
  - What does an object know?
    - x-position, y-position
    - x-velocity, y-velocity
    - radius
    - image filename
  - What can an object do?
    - Draw itself
    - Update its position, check for bouncing off walls



```
from Circle import Circle
import picture
import StdDraw

class BouncingImage(Circle):

    def __init__(self, x, y, vx, vy, r, image):
        super().__init__(x, y, r)
        self.vx = vx
        self.vy = vy
        self.image = image
        self.pic = picture.Picture(self.image

    def updatePos(self):
        self.posX += self.vx
        self.posY += self.vy
        if self.posX < 0.0 or self.posX > 1.0:
            self.vx *= -1
        if self.posY < 0.0 or self.posY > 1.0:
            self.vy *= -1

    def draw(self):
        StdDraw.picture(self.pic, self.posX, self.posY)
```

All this code appeared in the BouncingCircle class!

Repeated code is evil!

Inheritance: Bouncing Circular Images!

```
from BouncingCircle import BouncingCircle
import picture
import StdDraw

class BouncingImage(BouncingCircle):

    def __init__(self, x, y, vx, vy, r, image):
        super().__init__(x, y, vx, vy, r)
        self.image = image
        self.pic = picture.Picture(self.image)

    def draw(self):
        StdDraw.picture(self.pic, self.posX, self.posY)
```

This class is a child of the BouncingCircle class

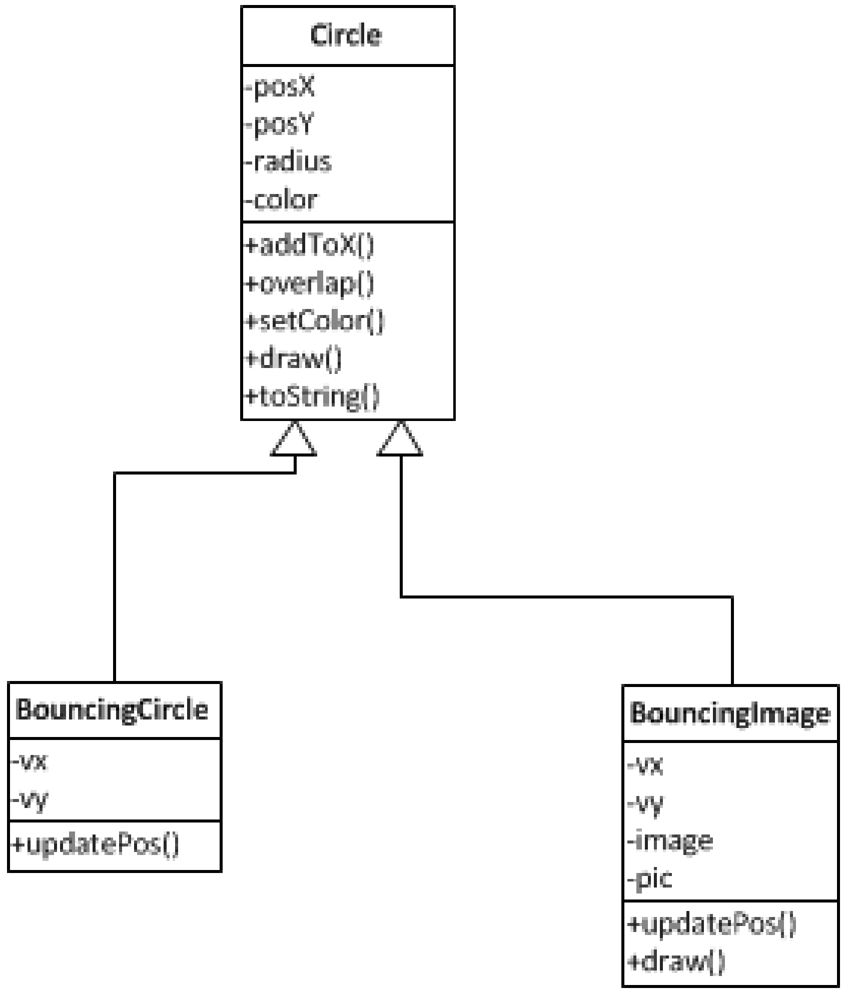
Calls the BouncingCircle constructor which sets all the other instance variables.

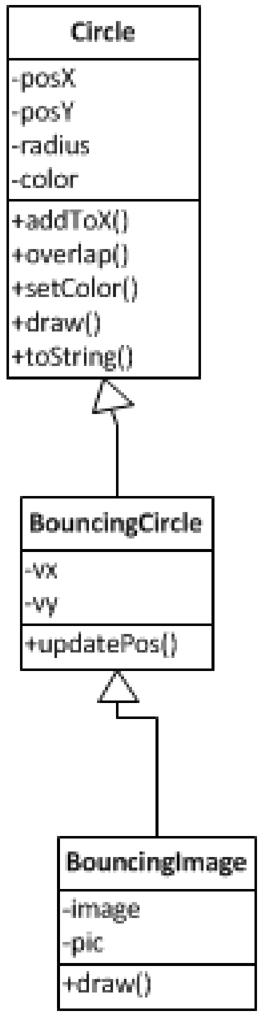
We only need our additional instance variable, others inherited from BouncingCircle

Overridden version of `draw()` method, this one draws a picture

Override = method with same method name as parent's method

Unified Modeling Language (UML) Class Diagram

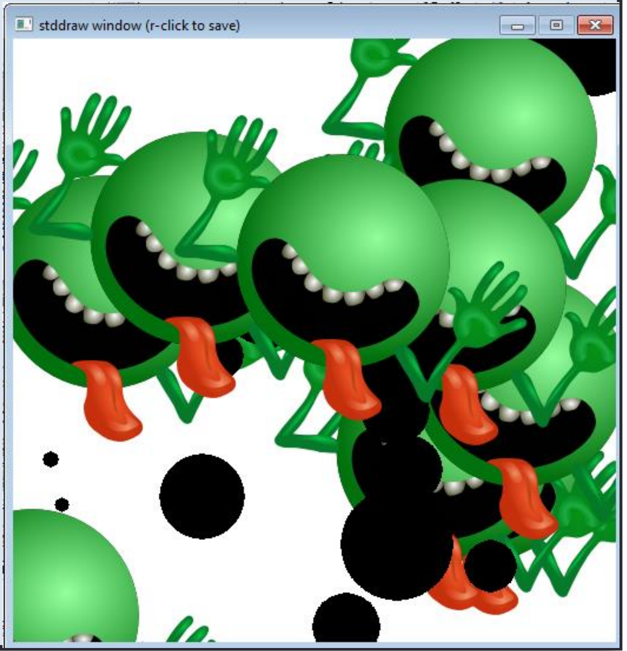




Client, 3 Object Types, *With* Inheritance and Polymorphism

```
from BouncingImage3 import BouncingImage as BI
from BouncingCircle import BouncingCircle as BC
from Circle import Circle as C
import random
import StdDraw

circles = []
for i in range(0,30):
    rand  = random.randint(0,2)
    x  = random.random()
    y  = random.random()
    vx = 0.002 - random.random() * 0.004
    vy = 0.002 - random.random() * 0.004
    r  = random.random() * 0.1

    if rand == 0:
        circles.append(C(x, y, r))
    elif rand == 1:
        circles.append(BC(x, y, vx, vy, r))
    else:
        circles.append(BI(x, y, vx, vy, r, "dont_panic.png"))

while True:
    StdDraw.clear()
    for i in range(0, len(circles)):
        if isinstance(circles[i], BC) or isinstance(circles[i], BI
            circles[i].updatePos()
        circles[i].draw()
    StdDraw.show(10)
```

Put them all together in one list!



What Method gets Run?

```
while True:
    StdDraw.clear()
    for i in range(0, len(circles)):
        if isinstance(circles[i], BC) or isinstance(circles[i], BI):
            circles[i].updatePos()
        circles[i].draw()
    StdDraw.show(10)
```

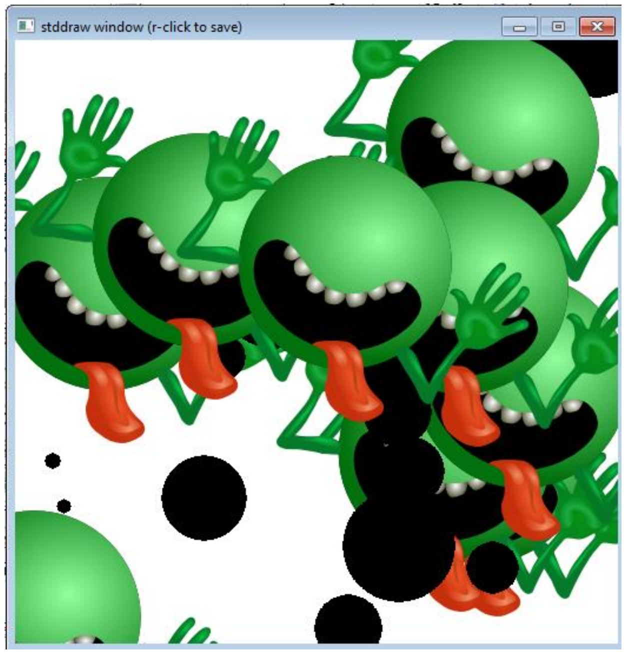
`circles[i]` could be: `Circle`, `BouncingCircle` or `BouncingImage` object

Circle	BouncingCircle	BouncingImage
`x, y, r`	`vx, vr`	`image`
`draw()`		`draw()`
	`updatePos()`	

<u>Rule:</u> Most specific method executes. If the subclass has the desired method, use that. Otherwise try your parent. If not, then your parent's parent, etc.

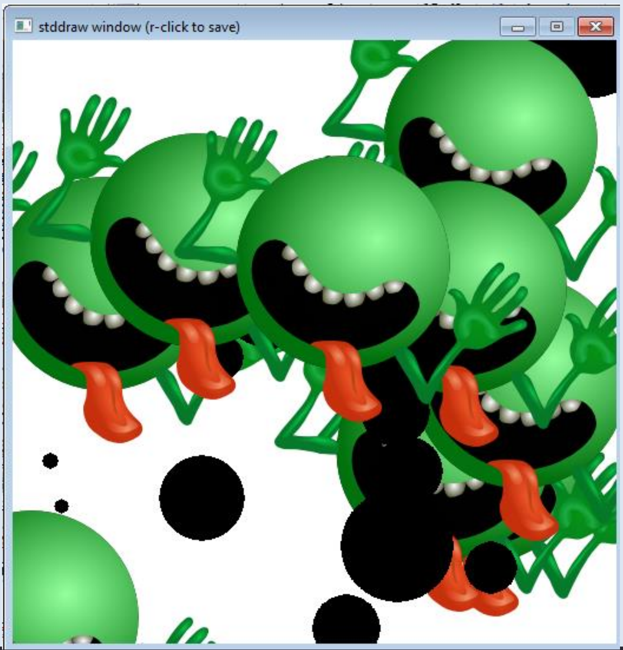
Object Collections

- Goal: Simplify main, offload work to object that manages a collection of objects
  - Helps hide implementation details
    - You can change how you store things later
- Let's fix up the bouncing program
  - Introduce new class `Bouncers`
  - Holds all the Circle type objects
  - Update and draw them all at once



Simplified Program

```
from Bouncers import Bouncers
import StdDraw

bouncers = Bouncers()
for i in range(0,30):
    bouncers.add()

while True:
    StdDraw.clear()
    bouncers.updateAll()
    bouncers.drawAll()
    StdDraw.show(10)
```



```
class Bouncers
------------------------------------------------------------------------
     __init__()    // Create an empty collection of bouncing objects
     add()         // Add a random type of bouncing object with a
                   // random location, velocity, and radius
     updateAll()   // Update the position of all bouncing objects
     drawAll()     // Draw all the objects to the screen
```

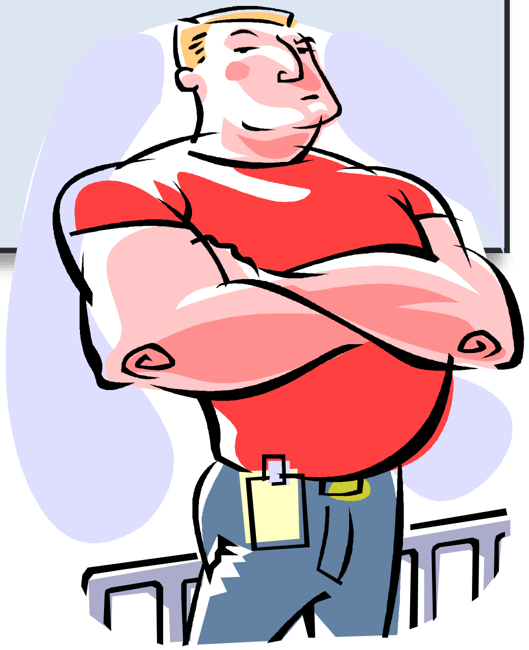
Application Programming Interface (API) for the Bouncers class.

Bouncer Implementation, 1/2

```
from BouncingImage3 import BouncingImage as BI
from BouncingCircle import BouncingCircle as BC
from Circle import Circle as C
import random
import StdDraw

class Bouncers:

    def __init__(self):
        self.circles = []

    def add(self):
        rand  = random.randint(0,2)
        x   = random.random()
        y   = random.random()
        vx = 0.002 - random.random() * 0.004
        vy = 0.002 - random.random() * 0.004
        r   = random.random() * 0.1

        if rand == 0:
            self.circles.append(C(x, y, r))
        elif rand == 1:
            self.circles.append(BC(x, y, vx, vy, r))
        else:
            self.circles.append(BI(x, y, vx, vy, r, "dont_panic.png"))
...
```

Bouncer Implementation, 2/2

```
def updateAll(self):
  for obj in self.circles:
    if isinstance(obj, BC) or isinstance(obj, BI):
        obj.updatePos()

def drawAll(self):
  for obj in self.circles:
      obj.draw()
```



Summary

- Inheritance
  - Sharing code between related classes
  - Extremely common in modern OOP languages
- Managing many objects
  - Create class holding a collection of other objects
  - Let's you simplify your main program
  - Hides details of how you store things