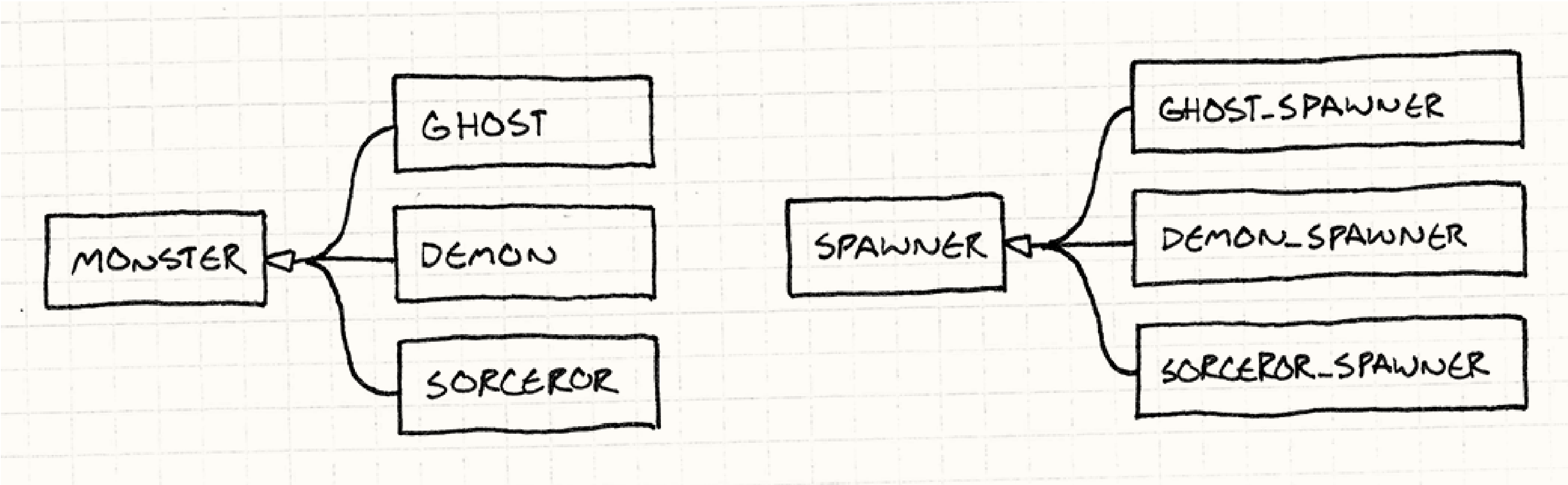
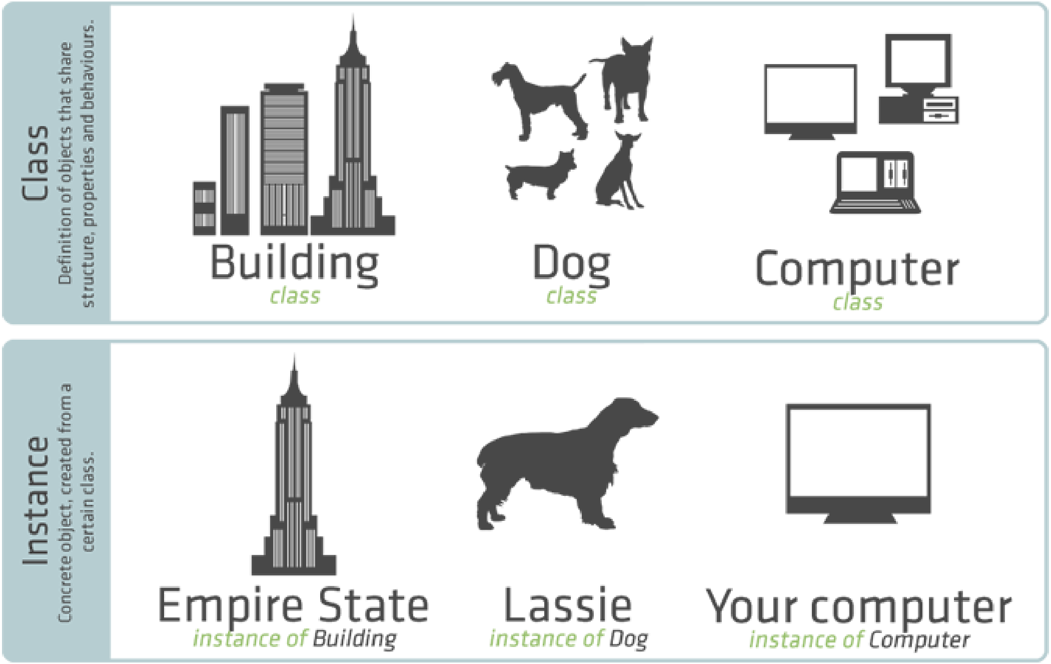
Object Oriented Programming

Motivation


MONSTER
GHOST
DEMON
SORCEROR
SPAWNER
GHOST_SPAWNER
DEMON_SPAWNER
SORCEROR_SPAWNER

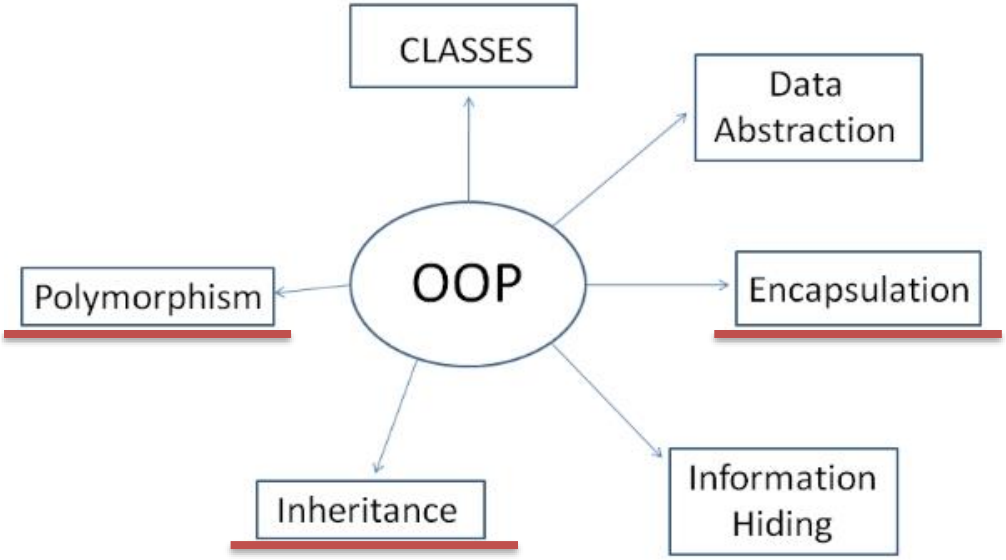
Class

- A class specifies the set of instance variables and methods that are "bundled together" for defining a type of object.



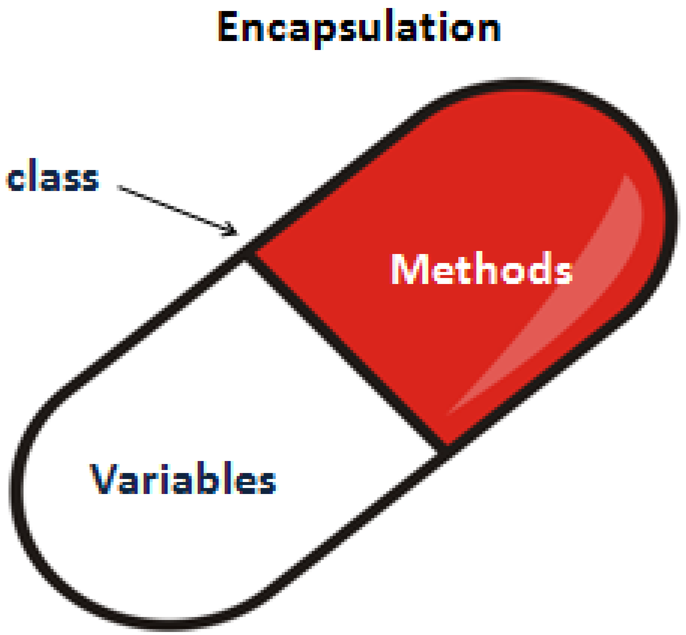
Fundamental features of OOP

- Three fundamental features supporting the design of object-oriented programs are referred to as encapsulation, inheritance, and polymorphism.

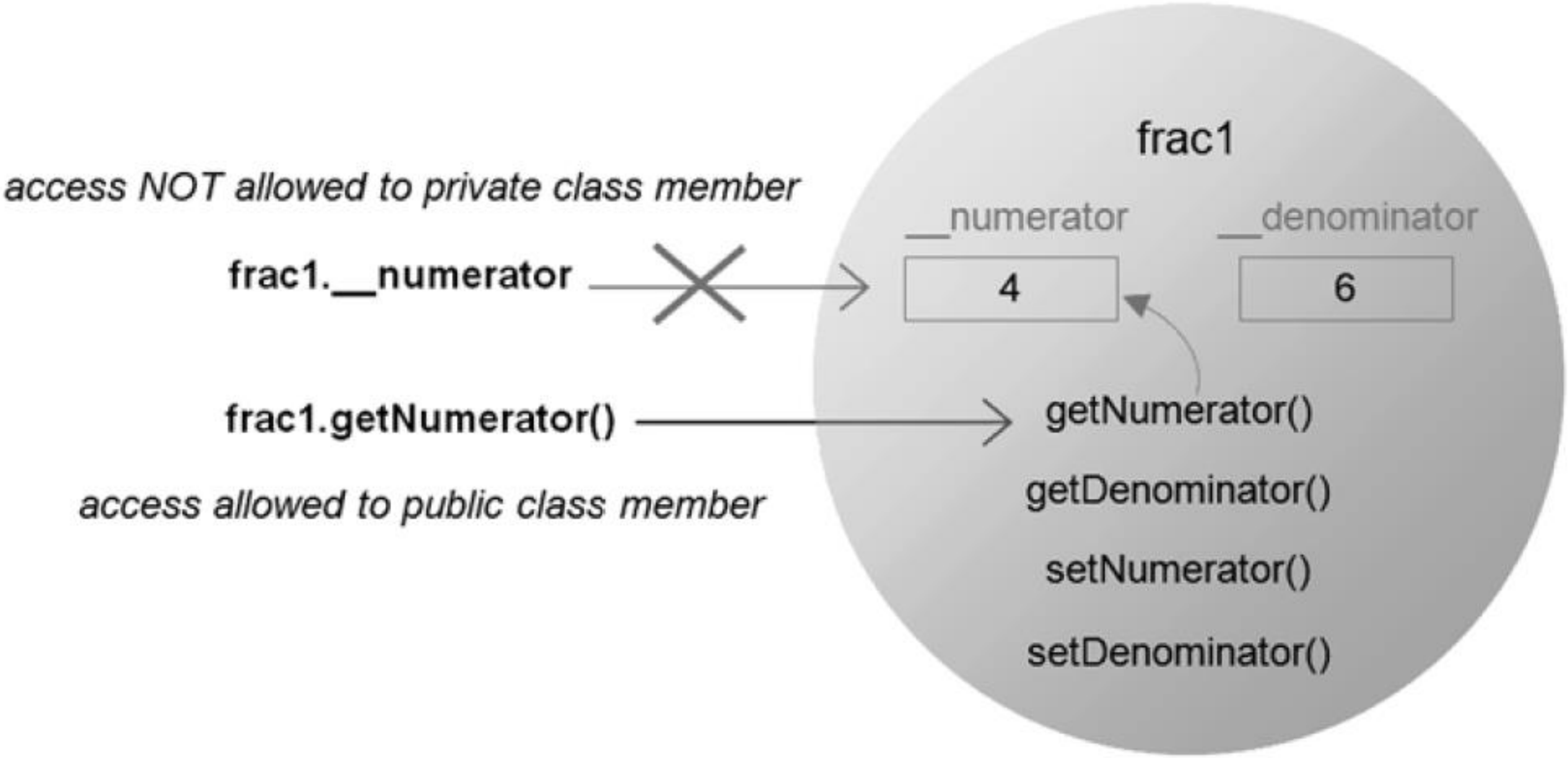


Encapsulation

- Encapsulation is a means of bundling together instance variables and methods to form a given type, as well as a way of restricting access to certain class members.

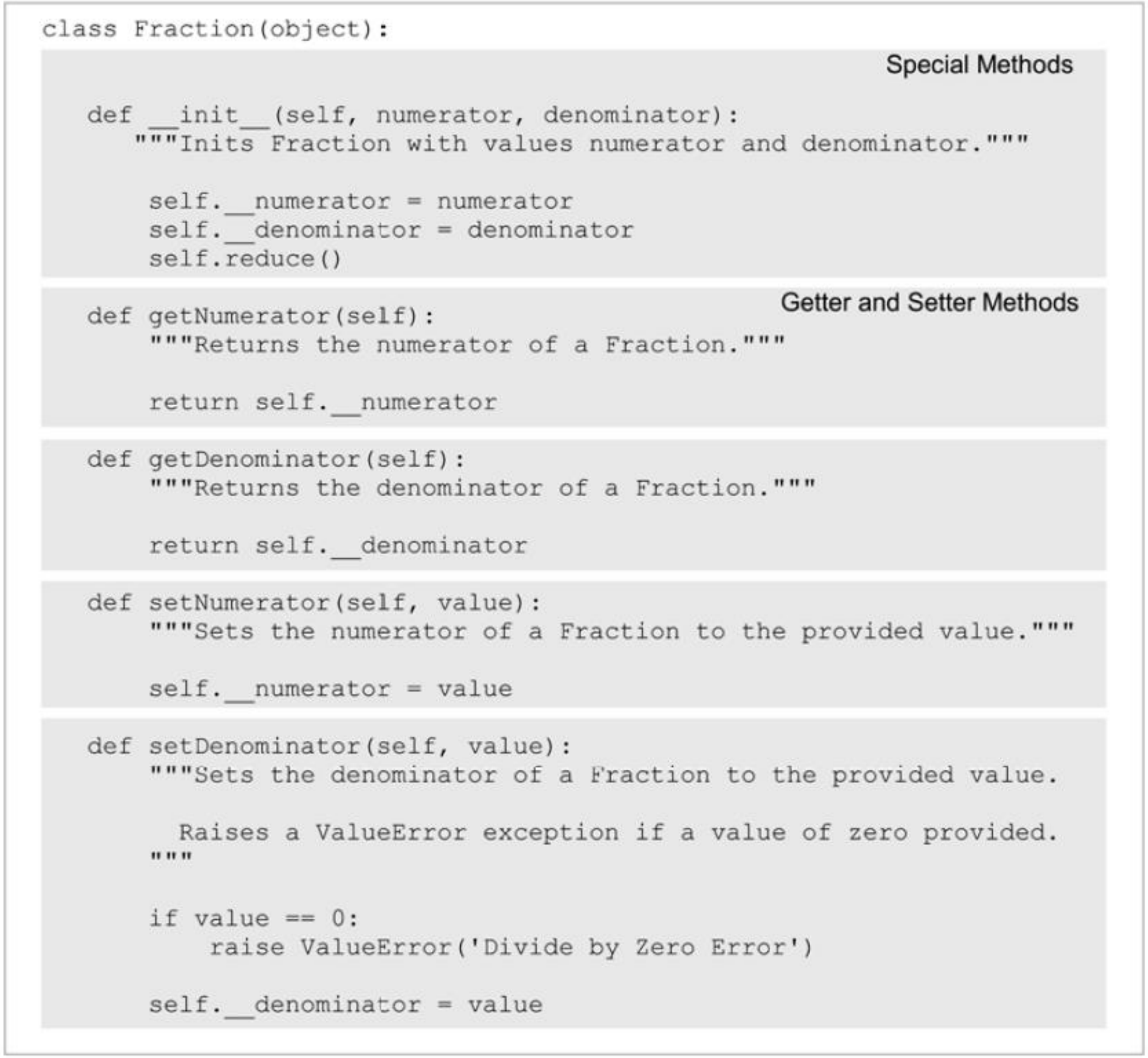


Encapsulation



getters and setters allow to get (return) and set (assign) private instance variables of a class

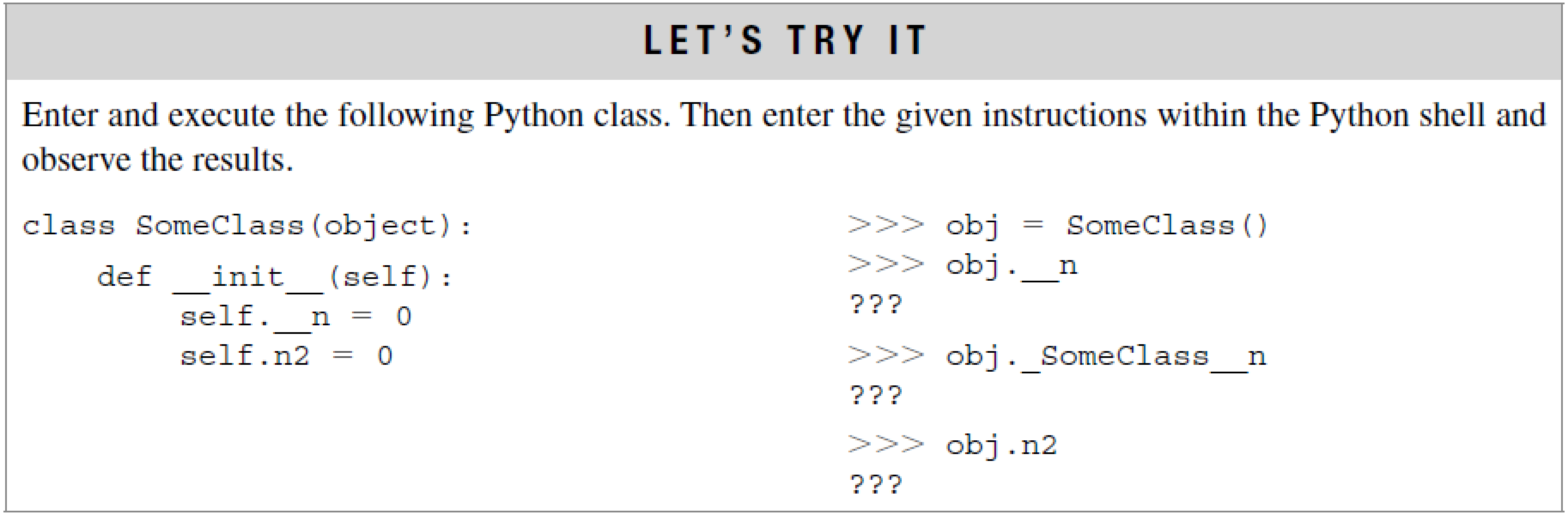
Defining Classes



Special Methods
Getter and Setter Methods

```
class Fraction(object):

    def __init__(self, numerator, denominator):
       """Inits Fraction with values numerator and denominator."""

        self.__numerator = numerator
        self.__denominator = denominator
        self.reduce()

    def getNumerator(self):
        """Returns the numerator of a Fraction."""

        return self.__numerator

    def getDenominator(self):
        """Returns the denominator of a Fraction."""

        return self.__denominator

    def setNumerator(self, value):
        """Sets the numerator of a Fraction to the provided value."""

        self.__numerator = value

    def setDenominator(self, value):
        """Sets the denominator of a Fraction to the provided value.

          Raises a ValueError exception if a value of zero provided.
        """

        if value == 0:
            raise ValueError('Divide by Zero Error')

        self.__denominator = value
```

Defining Classes



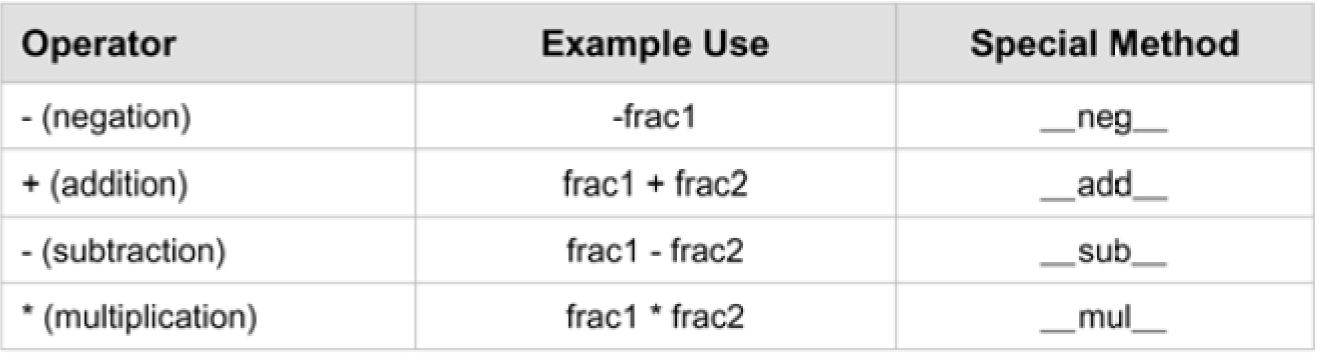
LET'S TRY IT

Enter and execute the following Python class. Then enter the given instructions within the Python shell and observe the results.

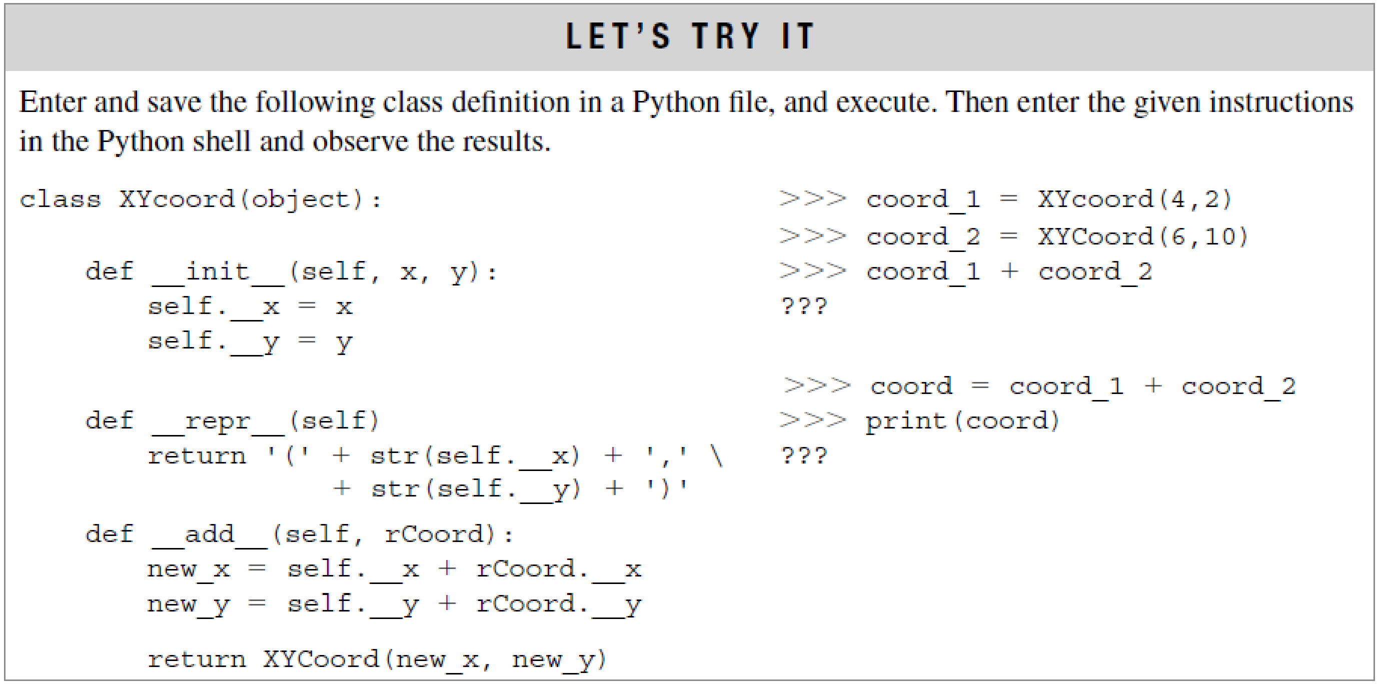
```
class SomeClass(object):

    def __init__(self):
        self.__n = 0
        self.n2 = 0
```

```
>>> obj = SomeClass()
>>> obj.__n
???

>>> obj._SomeClass__n
???

>>> obj.n2
???
```

Artimetic special operators



Operator	Example Use	Special Method
- (negation)	-frac1	__neg__
+ (addition)	frac1 + frac2	__add__
- (subtraction)	frac1 - frac2	__sub__
* (multiplication)	frac1 * frac2	__mul__

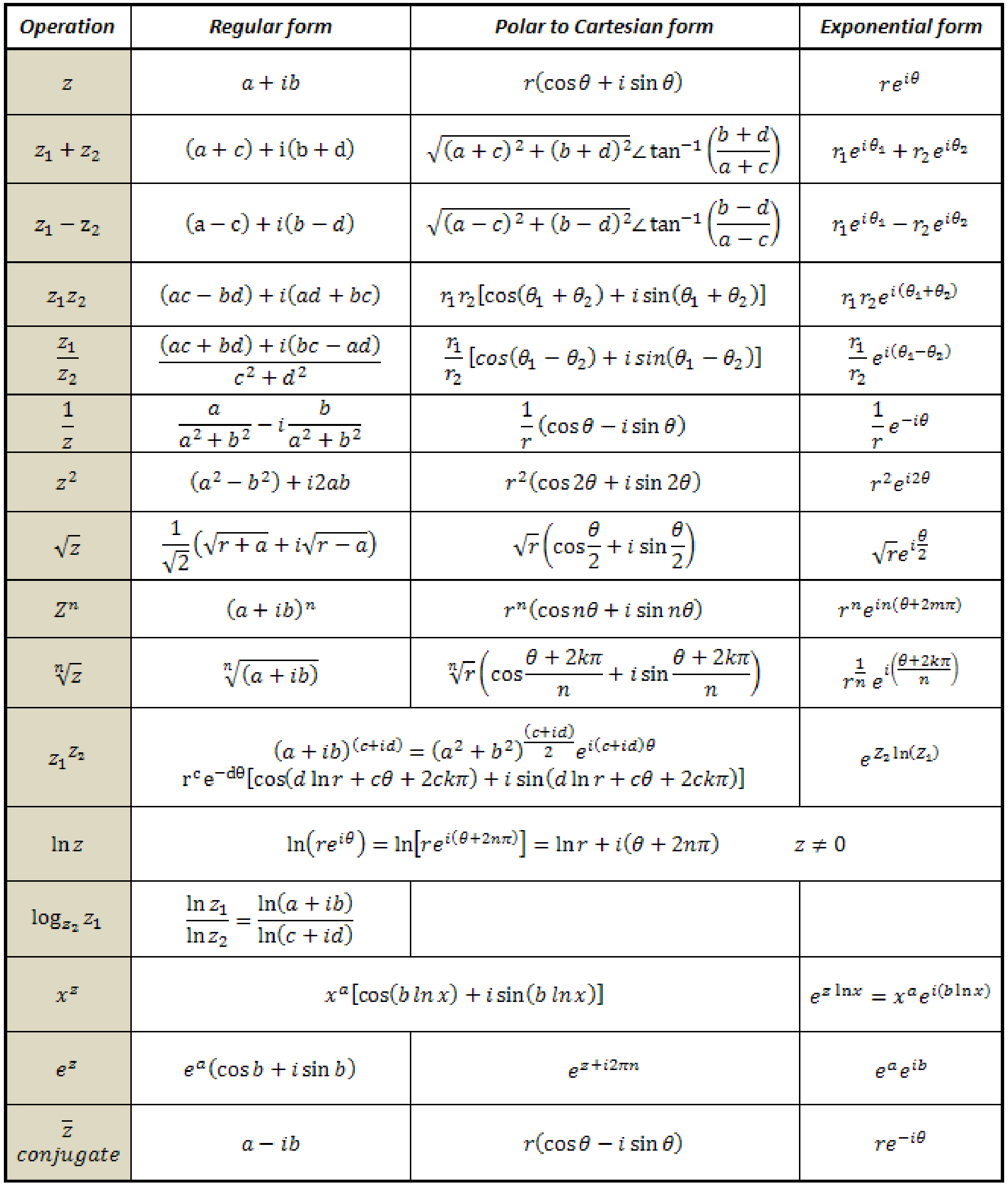


LET'S TRY IT

Enter and save the following class definition in a Python file, and execute. Then enter the given instructions in the Python shell and observe the results.

```
class XYcoord(object):

    def __init__(self, x, y):
        self.__x = x
        self.__y = y

    def __repr__(self)
        return '(' + str(self.__x) + ',' \
                   + str(self.__y) + ')'

    def __add__(self, rCoord):
        new_x = self.__x + rCoord.__x
        new_y = self.__y + rCoord.__y

        return XYCoord(new_x, new_y)
```

```
>>> coord_1 = XYcoord(4,2)
>>> coord_2 = XYCoord(6,10)
>>> coord_1 + coord_2
???

>>> coord = coord_1 + coord_2
>>> print(coord)
???
```



Operation	Regular form	Polar to Cartesian form	Exponential form
z	$a+ib$	$r(\cos\theta+i\sin\theta)$	$re^{i\theta}$
z_1+z_2	$(a+c)+i(b+d)$	$\sqrt{(a+c)^2+(b+d)^2}\angle\tan^{-1}\left(\frac{b+d}{a+c}\right)$	$r_1e^{i\theta_1}+r_2e^{i\theta_2}$
z_1-z_2	$(a-c)+i(b-d)$	$\sqrt{(a-c)^2+(b-d)^2}\angle\tan^{-1}\left(\frac{b-d}{a-c}\right)$	$r_1e^{i\theta_1}-r_2e^{i\theta_2}$
z_1z_2	$(ac-bd)+i(ad+bc)$	$r_1r_2[\cos(\theta_1+\theta_2)+i\sin(\theta_1+\theta_2)]$	$r_1r_2e^{i(\theta_1+\theta_2)}$
$\frac{z_1}{z_2}$	$\frac{(ac+bd)+i(bc-ad)}{c^2+d^2}$	$\frac{r_1}{r_2}[cos(\theta_1-\theta_2)+i\,sin(\theta_1-\theta_2)]$	$\frac{r_1}{r_2}e^{i(\theta_1-\theta_2)}$
$\frac{1}{z}$	$\frac{a}{a^2+b^2}-i\frac{b}{a^2+b^2}$	$\frac{1}{r}(\cos\theta-i\sin\theta)$	$\frac{1}{r}e^{-i\theta}$
z^2	$(a^2-b^2)+i2ab$	$r^2(\cos 2\theta+i\sin 2\theta)$	$r^2e^{i2\theta}$
$\sqrt{z}$	$\frac{1}{\sqrt{2}}(\sqrt{r+a}+i\sqrt{r-a})$	$\sqrt{r}\left(\cos\frac{\theta}{2}+i\sin\frac{\theta}{2}\right)$	$\sqrt{r}e^{i\frac{\theta}{2}}$
Z^n	$(a+ib)^n$	$r^n(\cos n\theta+i\sin n\theta)$	$r^ne^{in(\theta+2m\pi)}$
$\sqrt[n]{z}$	$\sqrt[n]{(a+ib)}$	$\sqrt[n]{r}\left(\cos\frac{\theta+2k\pi}{n}+i\sin\frac{\theta+2k\pi}{n}\right)$	$r^{\frac{1}{n}}e^{i\left(\frac{\theta+2k\pi}{n}\right)}$
${z_1}^{z_2}$	$(a+ib)^{(c+id)}=(a^2+b^2)^{\frac{(c+id)}{2}}e^{i(c+id)\theta}$ $\mathrm{r^c e^{-d\theta}}[\cos(d\ln r+c\theta+2ck\pi)+i\sin(d\ln r+c\theta+2ck\pi)]$		$e^{z_2\ln(z_1)}$
$\ln z$	$\ln(re^{i\theta})=\ln[re^{i(\theta+2n\pi)}]=\ln r+i(\theta+2n\pi)$ $\quad z\neq 0$		
$\log_{z_2}z_1$	$\frac{\ln z_1}{\ln z_2}=\frac{\ln(a+ib)}{\ln(c+id)}$		
x^z	$x^a[\cos(b\,ln\,x)+i\sin(b\,ln\,x)]$		$e^{z\ln x}=x^ae^{i(b\ln x)}$
e^z	$e^a(\cos b+i\sin b)$	$e^{z+i2\pi n}$	e^ae^{ib}
$\bar{z}$ *conjugate*	$a-ib$	$r(\cos\theta-i\sin\theta)$	$re^{-i\theta}$

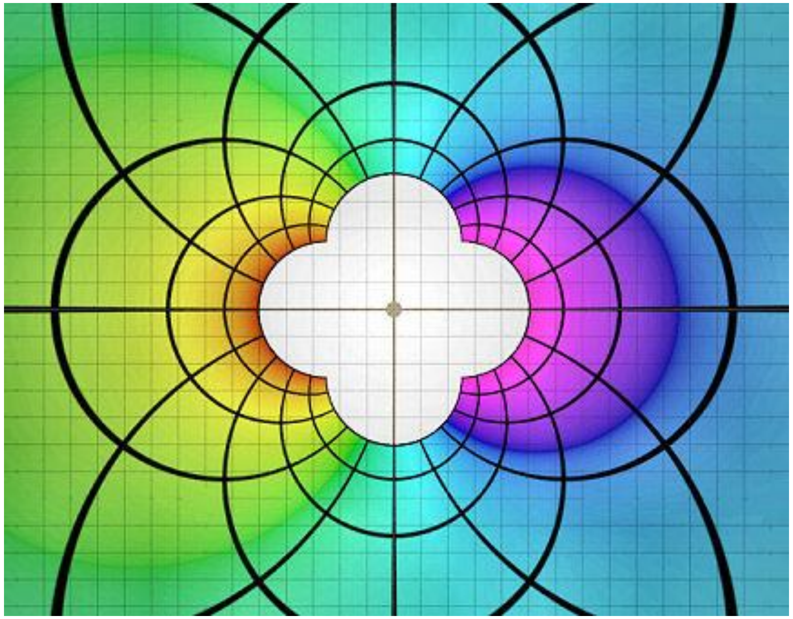
$z_1=a+ib \qquad z_2=c+id \qquad \arg(z)=\theta=\tan^{-1}\left(\frac{b}{a}\right)+2n\pi \qquad r=\sqrt{a^2+b^2} \qquad k=0,1\ldots.n-1$

$m,n=0,1,2\ldots$*any integer*

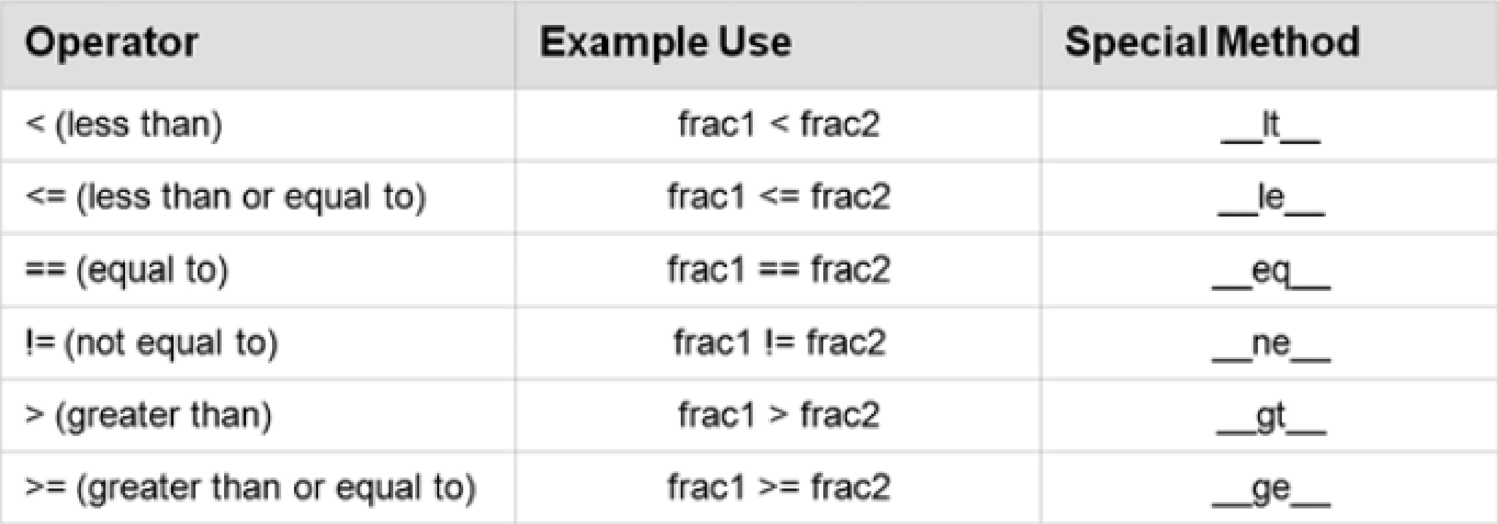
In geometry and complex analysis, a **Möbius transformation** of the complex plane is a rational function of the form

$$f(z)=\frac{az+b}{cz+d}$$

of one complex variable *z*; here the coefficients *a*, *b*, *c*, *d* are complex numbers satisfying *ad* − *bc* ≠ 0.



Relational special operators



Operator	Example Use	Special Method
< (less than)	frac1 < frac2	__lt__
<= (less than or equal to)	frac1 <= frac2	__le__
== (equal to)	frac1 == frac2	__eq__
!= (not equal to)	frac1 != frac2	__ne__
> (greater than)	frac1 > frac2	__gt__
>= (greater than or equal to)	frac1 >= frac2	__ge__

Inheritance

- Inheritance in object-oriented programming is the ability of a subclass (also “derived class” or “child class”) to **inherit members of a superclass** (also “base class” or “parent class”) as part of its own definition.

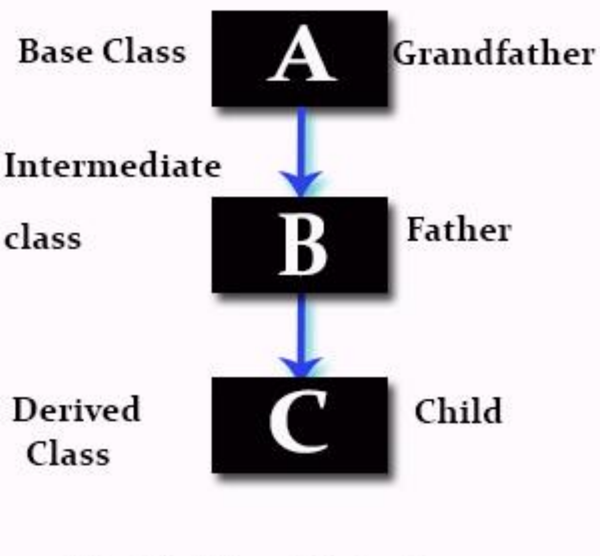
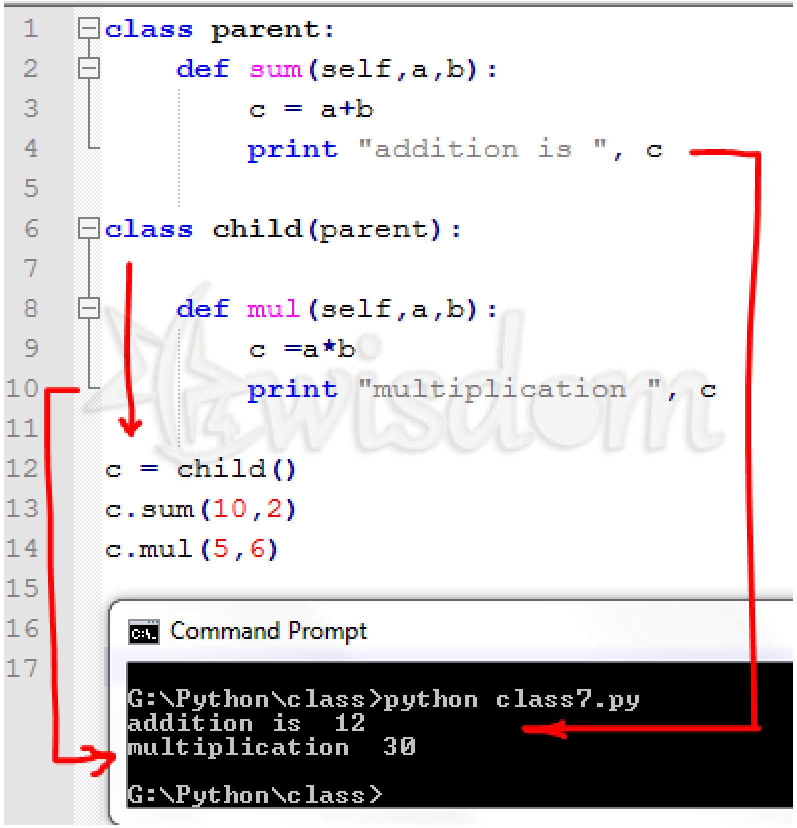


Fig: Multilevel Inheritance

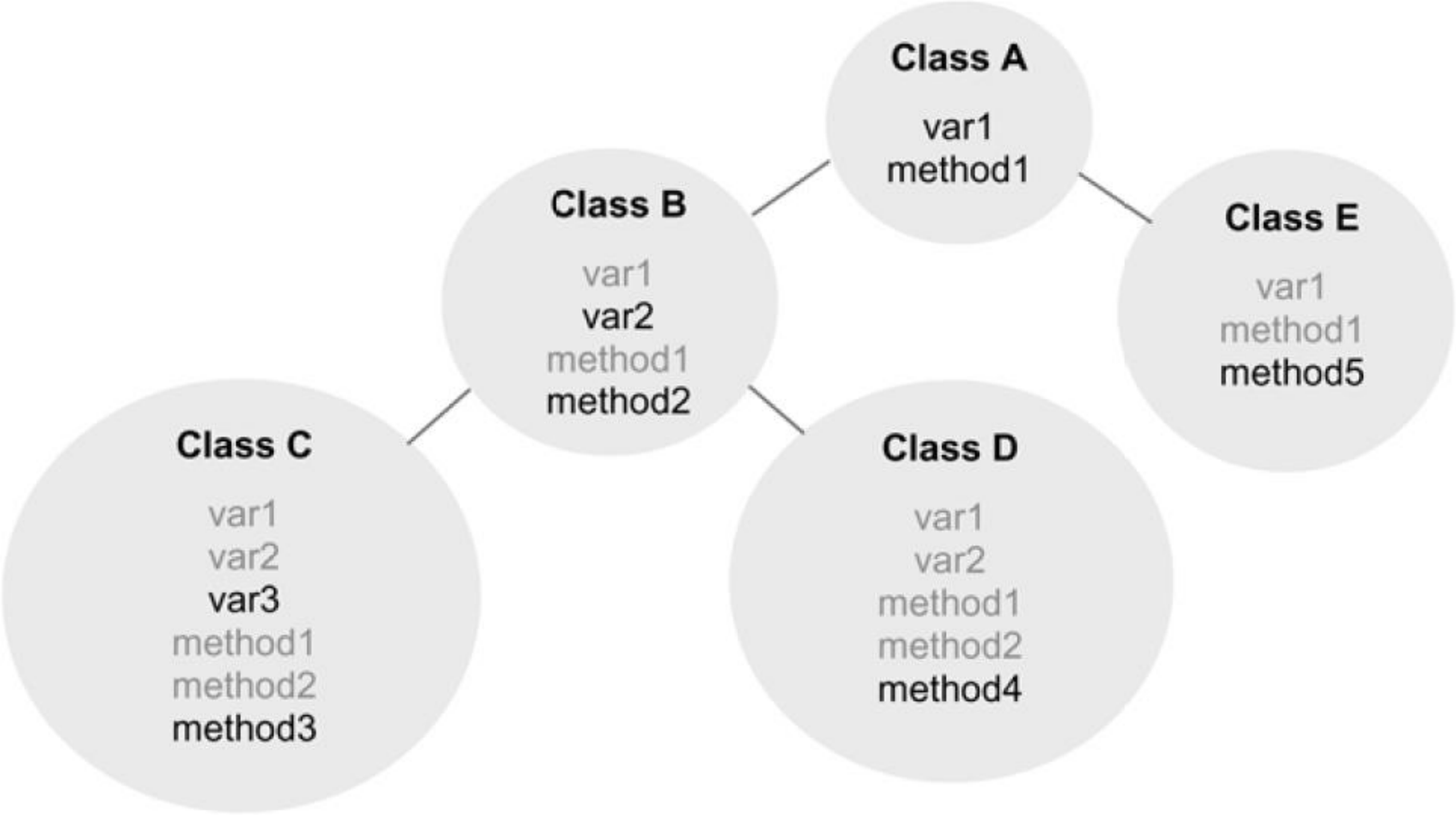
Inheritance



```
class parent:
    def sum(self,a,b):
        c = a+b
        print "addition is ", c

class child(parent):

    def mul(self,a,b):
        c =a*b
        print "multiplication ", c

c = child()
c.sum(10,2)
c.mul(5,6)
```

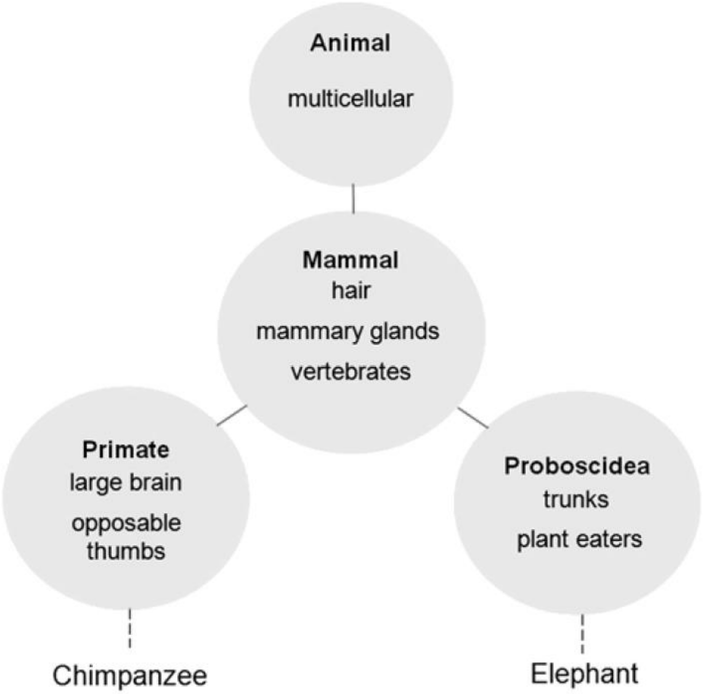
```
G:\Python\class>python class7.py
addition is  12
multiplication  30

G:\Python\class>
```

Inheritance



Subtype

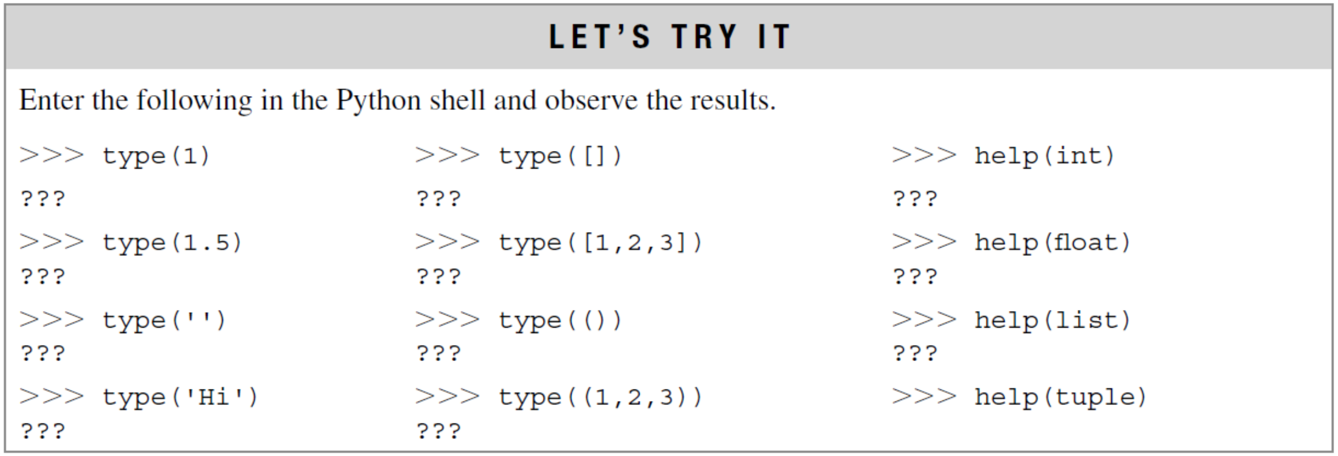
- A **subtype** is something that can be substituted for and behave as its parent type (and *its* parent type, etc.).



Inheritance

- *Andy was very interested in animals. He had many books about them, and went to see animals whenever he had the chance.*
- *Andy was very interested in chimpanzees. He had many books about them, and went to see chimpanzees whenever he had the chance.*
- *Andy was very interested in chimpanzees. He had many books about them, and loved to watch the chimpanzees swing from tree to tree.*
- *Andy was very interested in elephants. He had many books about them, and loved to watch the elephants swing from tree to tree.*

- Built-in function type can be used to determine the type (class name) of any value in Python. Built-in function help can be used to get the class description of a built-in type.

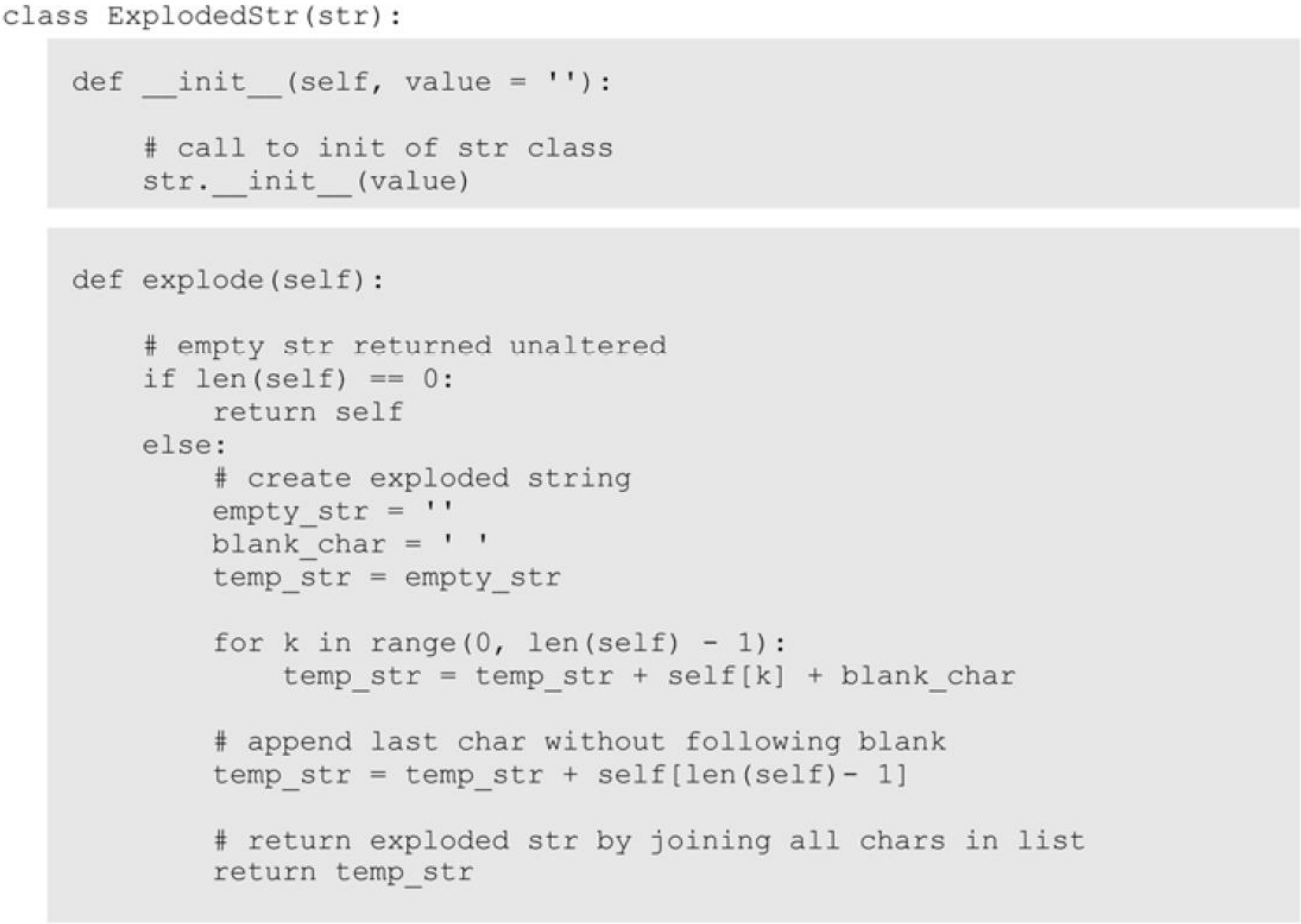


LET'S TRY IT

Enter the following in the Python shell and observe the results.

```
>>> type(1)        >>> type([])          >>> help(int)
???                ???                   ???
>>> type(1.5)      >>> type([1,2,3])     >>> help(float)
???                ???                   ???
>>> type('')       >>> type(())          >>> help(list)
???                ???                   ???
>>> type('Hi')     >>> type((1,2,3))     >>> help(tuple)
???                ???
```

Subtype



```
class ExplodedStr(str):

    def __init__(self, value = ''):

        # call to init of str class
        str.__init__(value)

    def explode(self):

        # empty str returned unaltered
        if len(self) == 0:
            return self
        else:
            # create exploded string
            empty_str = ''
            blank_char = ' '
            temp_str = empty_str

            for k in range(0, len(self) - 1):
                temp_str = temp_str + self[k] + blank_char

            # append last char without following blank
            temp_str = temp_str + self[len(self)- 1]

            # return exploded str by joining all chars in list
            return temp_str
```

Polymorphism

- In object-oriented programming, **polymorphism** allows objects of different types, each with their own specific behaviors, to be treated as the same general type.

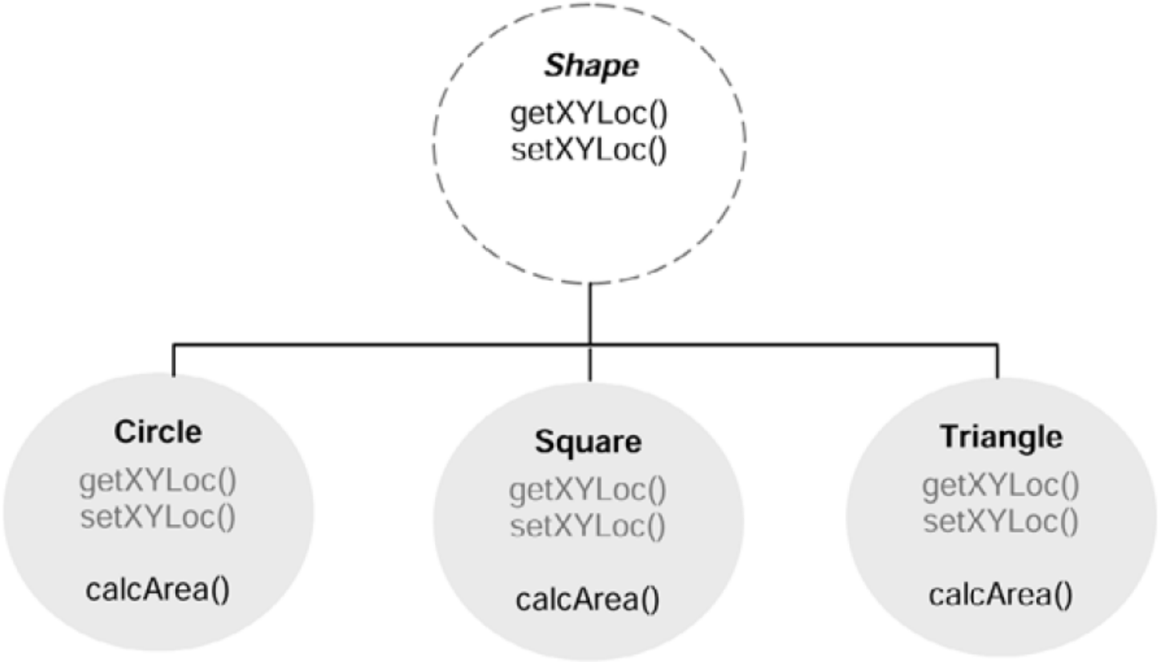


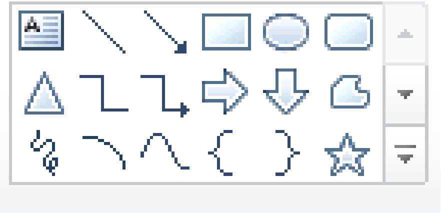
FIGURE 10-19 Polymorphic Shape Class

Polymorphism

"Now Speak!"
"Quack"
"Woof"
"Meow"

```
class Bird(object):
    def __init__(self, w):
        print('__init__ of Bird Class called')
        self.__weight = w

    def getWeight(self):
        return str(self.__weight) + 'ounces'

    def getColor(self):
        raise NotImplementedError( \
              'Method color not implemented')

class BlueJay(Bird):
    def __init__(self, w):
        Bird.__init__(self, w)

    def getColor(self):
        return 'Blue'

class Cardinal(Bird):
    def __init__(self, w):
        Bird.__init__(self, w)

    def getColor(self):
        return 'Red'

class BlackBird(Bird):
    def __init__(self, w):
        Bird.__init__(self, w)

    def getColor(self):
        return 'Black'
```

```
>>> b1 = BlueJay(1)
???

>>> b2 = Cardinal(1.4)
???

>>> b3 = BlackBird(3.5)
???


>>> b1.getWeight()
???

>>> b2.getWeight()
???

>>> b3.getWeight()
???


>>> b1.getColor()
???

>>> b2.getColor()
???

>>> b3.getColor()
???

>>> b4 = Bird(2.0)
>>> b.getColor()
```



Using polymorphism

Circle, Square, and Triangle do not have a common set of methods.

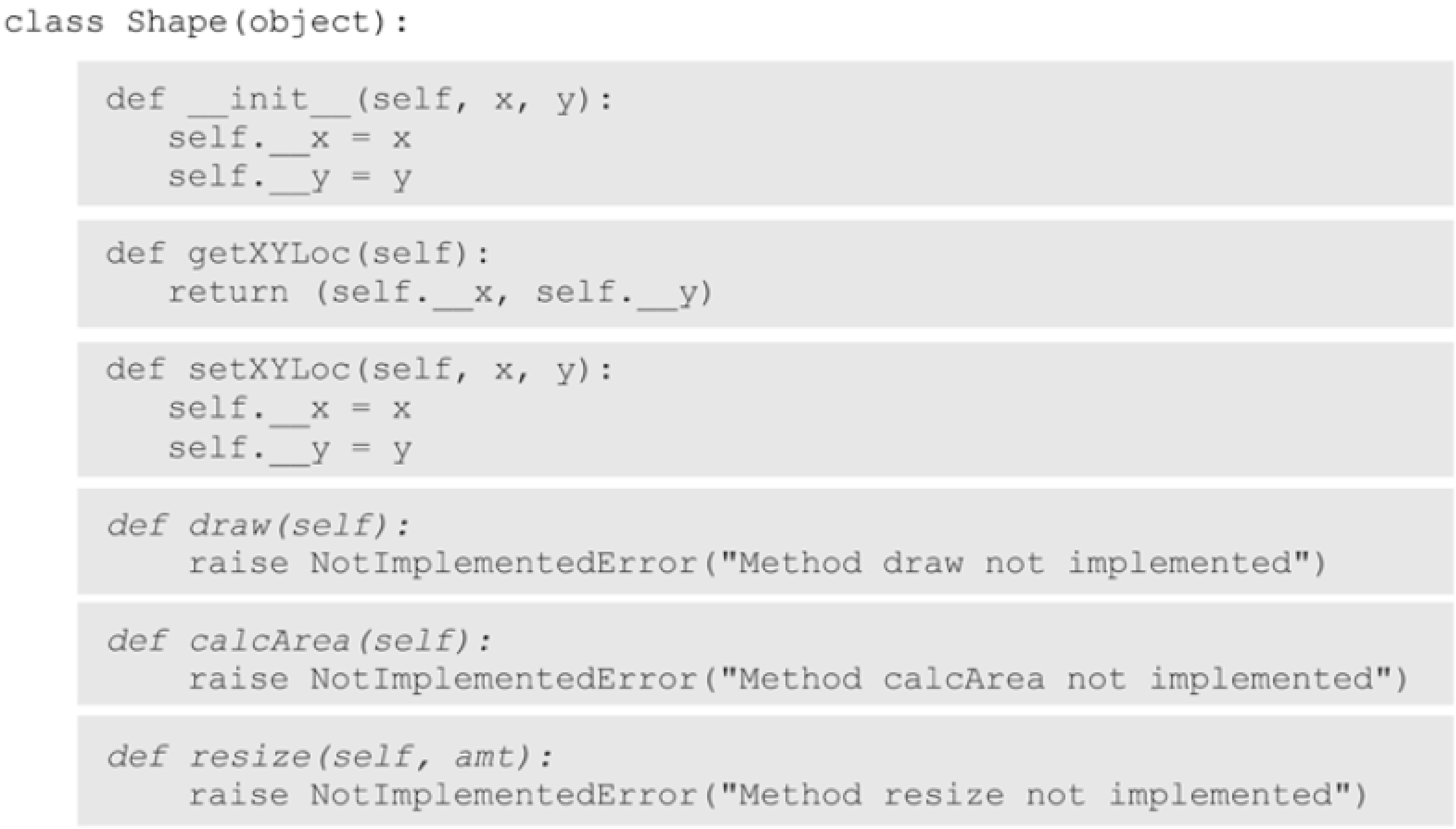
```
if selected_shape == 1:
    cir = Circle(0, 0, 1)
elif selected_shape == 2:
    sqr = Square(0, 0, 1)
elif selected_shape == 3:
    tri = Triangle(0, 0, 1)
```

```
if selected_shape == 1:
    cir.drawCircle()
elif selected_shape == 2:
    sqr.drawSquare()
elif selected_shape == 3:
    tri.drawTriangle()
```

```
if selected_shape == 1:
    area = cir.calcCircleArea()
elif selected_shape == 2:
    area = sqr.calcSquareArea()
elif selected_shape == 3:
    area = tri.calcTriangleArea()
```

```
if selected_shape == 1:
    cir.moveCircle(x, y)
elif selected_shape == 2:
    sqr.moveSquare(x, y)
elif selected_shape == 3:
    tri.moveTriangle(x, y)
```

Using polymorphism



```
class Shape(object):
    def __init__(self, x, y):
       self.__x = x
       self.__y = y

    def getXYLoc(self):
       return (self.__x, self.__y)

    def setXYLoc(self, x, y):
       self.__x = x
       self.__y = y

    def draw(self):
        raise NotImplementedError("Method draw not implemented")

    def calcArea(self):
        raise NotImplementedError("Method calcArea not implemented")

    def resize(self, amt):
        raise NotImplementedError("Method resize not implemented")
```

Non-Polymorphic Code

```
# Create Appropriate Object

if selected_shape == 1:
    cir = Circle(0, 0, 1)
elif selected_shape == 2:
    sqr = Square(0, 0, 1)
elif selected_shape == 3:
    tri = Triangle(0, 0, 1)


# draw
if selected_shape == 1:
    cir.drawCircle()
elif selected_shape == 2:
    sqr.drawSquare()
elif selected_shape == 3:
    tri.drawTriangle()

# calc area
if selected_shape == 1:
    area = cir.calcCircleArea()
elif selected_shape == 2:
    area = sqr.calcSquareArea()
elif selected_shape == 3:
    area = tri.calcTriangleArea()

# resize
if selected_shape == 1:
    cir.resizeCircle(percentage)
elif selected_shape == 2:
    sqr.resizeSquare(percentage)
elif selected_shape == 3:
    tri.resizeTriangle(percentage)

# reposition
if selected_shape == 1:
    cir.setXY(x, y)
elif selected_shape == 2:
    sqr.setPosition(x, y)
elif selected_shape == 3:
    tri.moveTo(x, y)
```

Polymorphic Code

```
# Create Appropriate Object

if selected_shape == 1:
    fig = Circle(0, 0, 1)
elif selected_shape == 2:
    fig = Square(0, 0, 1)
elif selection_shape == 3:
    fig = Triangle(0, 0, 1)


# draw
fig.draw()

# calc area
area = fig.calcArea()

# resize
fig.resize(selected_percentage)

# reposition
fig.setXYLoc(x, y)
```

For the call to method `draw()`, the method defined in the specific subclass is the actual method called.

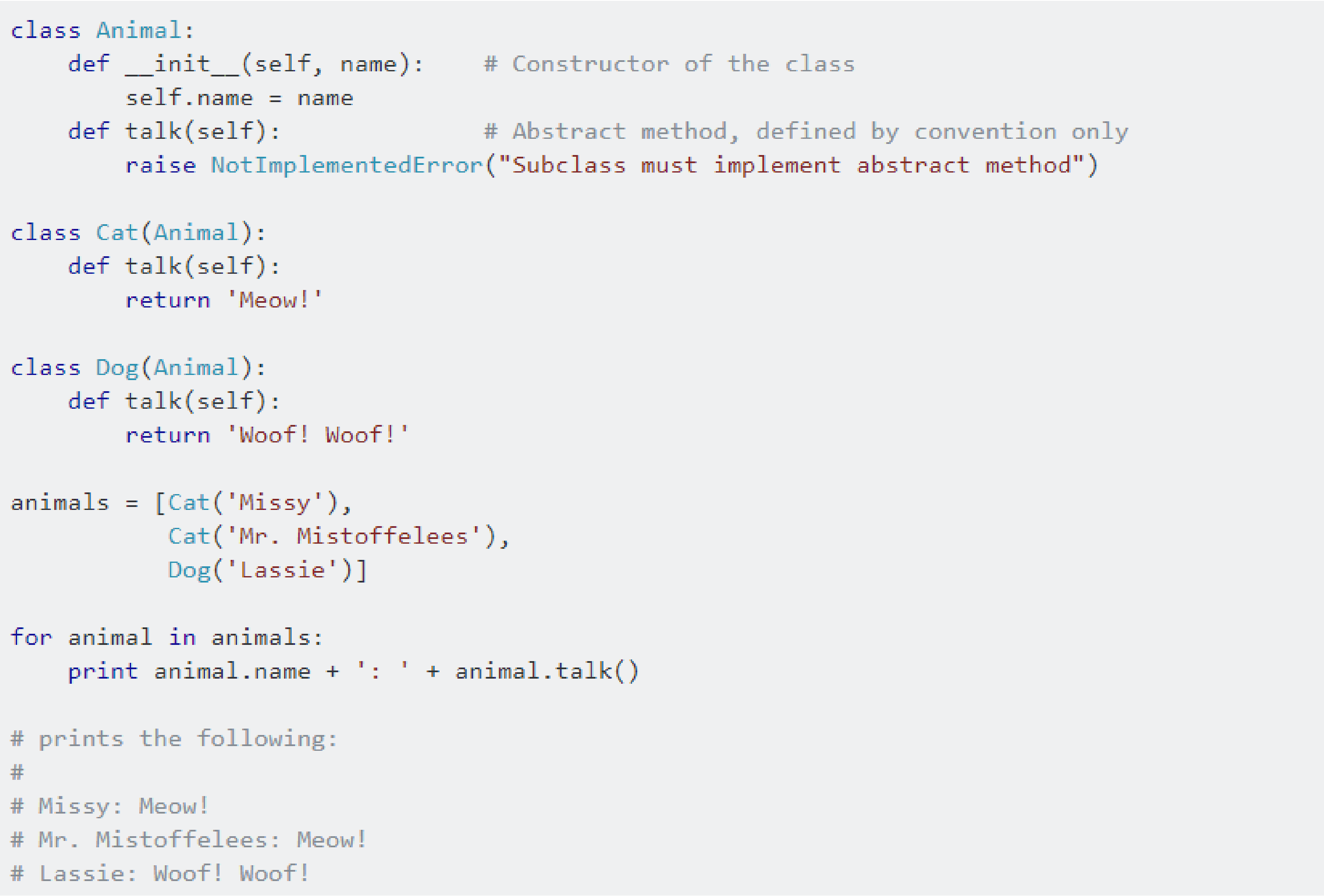
For the call to method `calcArea()`, the method defined in the specific subclass is the actual method called.

For the call to method `resize()`, the method defined in the specific subclass is the actual method called.

For the call to method `setXYLoc()`, there is no method defined in any of the subclasses. Therefore, for each particular shape, the method of the `Shape` class is the method called.

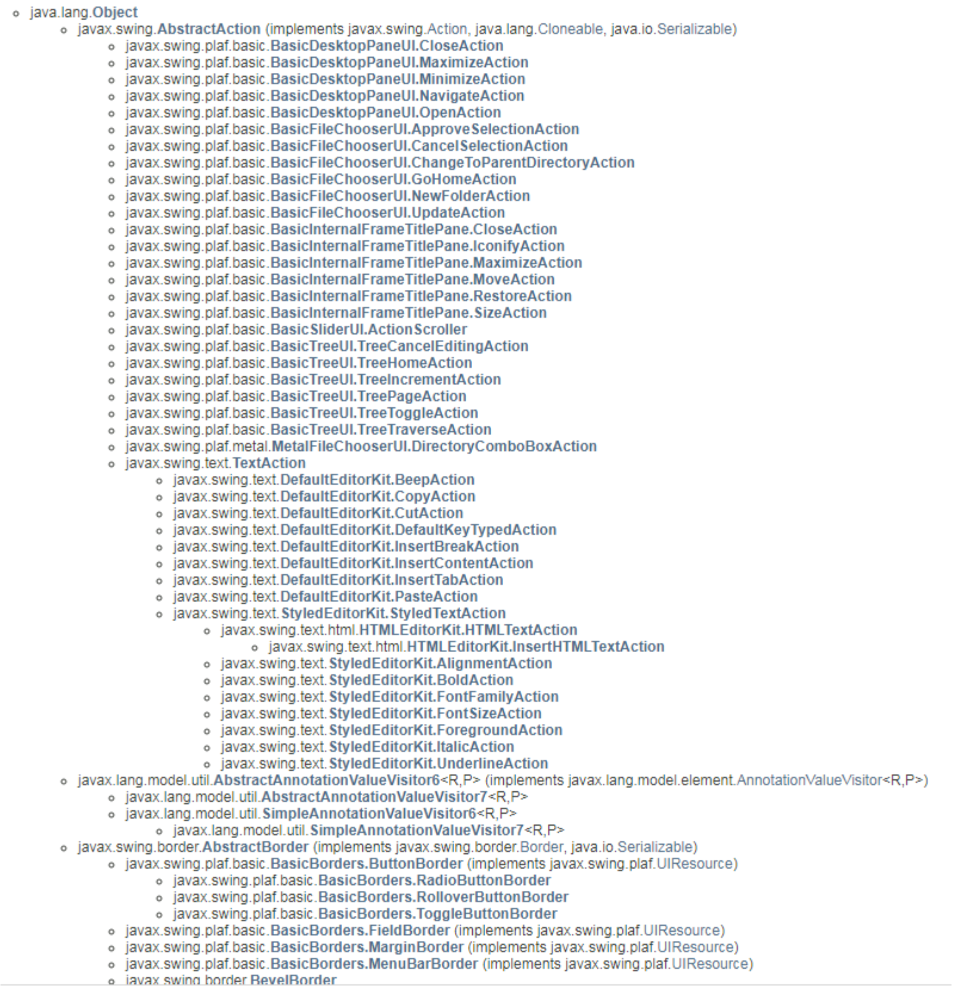
FIGURE 10-23 Nonpolymorphic vs. Polymorphic Code

Using polymorphism



```
class Animal:
    def __init__(self, name):    # Constructor of the class
        self.name = name
    def talk(self):              # Abstract method, defined by convention only
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'),
           Cat('Mr. Mistoffelees'),
           Dog('Lassie')]

for animal in animals:
    print animal.name + ': ' + animal.talk()

# prints the following:
#
# Missy: Meow!
# Mr. Mistoffelees: Meow!
# Lassie: Woof! Woof!
```

- https://docs.oracle.com/javase/7/docs/api/overview-tree.html



- java.lang.**Object**
  - javax.swing.**AbstractAction** (implements javax.swing.Action, java.lang.Cloneable, java.io.Serializable)
    - javax.swing.plaf.basic.**BasicDesktopPaneUI.CloseAction**
    - javax.swing.plaf.basic.**BasicDesktopPaneUI.MaximizeAction**
    - javax.swing.plaf.basic.**BasicDesktopPaneUI.MinimizeAction**
    - javax.swing.plaf.basic.**BasicDesktopPaneUI.NavigateAction**
    - javax.swing.plaf.basic.**BasicDesktopPaneUI.OpenAction**
    - javax.swing.plaf.basic.**BasicFileChooserUI.ApproveSelectionAction**
    - javax.swing.plaf.basic.**BasicFileChooserUI.CancelSelectionAction**
    - javax.swing.plaf.basic.**BasicFileChooserUI.ChangeToParentDirectoryAction**
    - javax.swing.plaf.basic.**BasicFileChooserUI.GoHomeAction**
    - javax.swing.plaf.basic.**BasicFileChooserUI.NewFolderAction**
    - javax.swing.plaf.basic.**BasicFileChooserUI.UpdateAction**
    - javax.swing.plaf.basic.**BasicInternalFrameTitlePane.CloseAction**
    - javax.swing.plaf.basic.**BasicInternalFrameTitlePane.IconifyAction**
    - javax.swing.plaf.basic.**BasicInternalFrameTitlePane.MaximizeAction**
    - javax.swing.plaf.basic.**BasicInternalFrameTitlePane.MoveAction**
    - javax.swing.plaf.basic.**BasicInternalFrameTitlePane.RestoreAction**
    - javax.swing.plaf.basic.**BasicInternalFrameTitlePane.SizeAction**
    - javax.swing.plaf.basic.**BasicSliderUI.ActionScroller**
    - javax.swing.plaf.basic.**BasicTreeUI.TreeCancelEditingAction**
    - javax.swing.plaf.basic.**BasicTreeUI.TreeHomeAction**
    - javax.swing.plaf.basic.**BasicTreeUI.TreeIncrementAction**
    - javax.swing.plaf.basic.**BasicTreeUI.TreePageAction**
    - javax.swing.plaf.basic.**BasicTreeUI.TreeToggleAction**
    - javax.swing.plaf.basic.**BasicTreeUI.TreeTraverseAction**
    - javax.swing.plaf.metal.**MetalFileChooserUI.DirectoryComboBoxAction**
    - javax.swing.text.**TextAction**
      - javax.swing.text.**DefaultEditorKit.BeepAction**
      - javax.swing.text.**DefaultEditorKit.CopyAction**
      - javax.swing.text.**DefaultEditorKit.CutAction**
      - javax.swing.text.**DefaultEditorKit.DefaultKeyTypedAction**
      - javax.swing.text.**DefaultEditorKit.InsertBreakAction**
      - javax.swing.text.**DefaultEditorKit.InsertContentAction**
      - javax.swing.text.**DefaultEditorKit.InsertTabAction**
      - javax.swing.text.**DefaultEditorKit.PasteAction**
      - javax.swing.text.**StyledEditorKit.StyledTextAction**
        - javax.swing.text.html.**HTMLEditorKit.HTMLTextAction**
          - javax.swing.text.html.**HTMLEditorKit.InsertHTMLTextAction**
        - javax.swing.text.**StyledEditorKit.AlignmentAction**
        - javax.swing.text.**StyledEditorKit.BoldAction**
        - javax.swing.text.**StyledEditorKit.FontFamilyAction**
        - javax.swing.text.**StyledEditorKit.FontSizeAction**
        - javax.swing.text.**StyledEditorKit.ForegroundAction**
        - javax.swing.text.**StyledEditorKit.ItalicAction**
        - javax.swing.text.**StyledEditorKit.UnderlineAction**
  - javax.lang.model.util.**AbstractAnnotationValueVisitor6**<R,P> (implements javax.lang.model.element.AnnotationValueVisitor<R,P>)
    - javax.lang.model.util.**AbstractAnnotationValueVisitor7**<R,P>
    - javax.lang.model.util.**SimpleAnnotationValueVisitor6**<R,P>
      - javax.lang.model.util.**SimpleAnnotationValueVisitor7**<R,P>
  - javax.swing.border.**AbstractBorder** (implements javax.swing.border.Border, java.io.Serializable)
    - javax.swing.plaf.basic.**BasicBorders.ButtonBorder** (implements javax.swing.plaf.UIResource)
      - javax.swing.plaf.basic.**BasicBorders.RadioButtonBorder**
      - javax.swing.plaf.basic.**BasicBorders.RolloverButtonBorder**
      - javax.swing.plaf.basic.**BasicBorders.ToggleButtonBorder**
    - javax.swing.plaf.basic.**BasicBorders.FieldBorder** (implements javax.swing.plaf.UIResource)
    - javax.swing.plaf.basic.**BasicBorders.MarginBorder** (implements javax.swing.plaf.UIResource)
    - javax.swing.plaf.basic.**BasicBorders.MenuBarBorder** (implements javax.swing.plaf.UIResource)
    - javax.swing.border.**BevelBorder**