Much of the power of object-oriented programming derives from the notion of *inheritance*. A name that immediately follows a ‘dot’ is called a qualified name (it is qualified by the value of the expression to the left of the dot). When you use a qualified name — e.g., the name `attr` in the expression `obj.attr` — Python searches a tree of namespaces to find the value associate with the name. This name-space tree is defined using the *class hierarchy*. This exercise will deepen your understanding of inheritance and the *scope rules* that Python uses with qualified names.

**Part (a): [Class hierarchy]** The header of a class definition contains a parenthesized type name after the class name. This type is called the *parent* of the new class, and the new class is called a *child* of the parent type.

Answer the following questions about the program drinks.py which :

**Q1:** For each type, what is its parent?

Parent of *Drink*: ___________________________  Parent of *CocaCola*: ___________________________

Parent of *Milk*: ___________________________  Parent of *ChocolateMilk*: ______________________

**Q2:** For each type, what are its children (if any)?

Children of *Drink*: ___________________________  Children of *CocaCola*: ___________________________

Children of *Milk*: ___________________________  Children of *ChocolateMilk*: ______________________

The parent/child relation on the classes in a Python program defines the class hierarchy. Computer scientists show a class hierarchy as a tree and use similar terminology as for a family tree. As in a family tree, a child is drawn below its parent and a line connects them. *Ancestors* and *descendants* are defined as in a family tree. Draw a picture of the class hierarchy defined by the drinks program below.
**Part (b): [Inheritance, Scope rules for qualified names, Class hierarchy]** The idea behind inheritance is that a descendant inherits the attributes of its ancestors, but also defines its own unique attributes and can redefine inherited ones. Python defines special scope rules for qualified names to implement inheritance: To find the value of an expression of the form `obj.attr`, the shell searches namespaces for one that associates `attr` with an object. In order, it searches:

1. First, the instance’s *local namespace*—i.e., the name space referenced by `obj`
2. Then, the *class namespace*—i.e., the namespace defined by `obj`’s type (i.e., its class)
3. Then, the namespaces of this class’ ancestors, in turn, going up the class hierarchy

If Python does not find a definition for `attr` after reaching `object`, it raises an `AttributeError`.

Because of these rules, an *adorned class diagram* can be useful in showing the data attributes and methods associated with instances of a class, as well as where they are defined. Beside each class name, the adorned diagram lists any (new) data attributes defined in the class constructor and lists all methods that are defined in the class definition. Finish adorning the class diagram below and then discuss with a partner how it helps you answer the questions below the diagram.

Assume the assignment:  

```python
A, C = CocaCola(), ChocolateMilk()
```

- What names are in the object (namespace) that `A` refers to?
- What names are in the object (namespace) that `C` references?
- Which of the following are legal?

  ```python
  A.get_price()  
  A.get_fat()  
  print( A )  
  ```

  ```python
  C.get_price()  
  C.get_fat()  
  print( C )  
  ```
For each block of code that follows, assume the class definitions in `drinks.py`.

1. Hand trace execution of the code, writing the numbers of the statements in `drinks.py` that will be executed in the order that they will be executed in the space provided.
2. Below the list of statements executed, show what is printed.
3. Draw a picture of the object that `d` references.

To illustrate, we’ve filled in and will walk through the first one with you. After finishing each of the next two code segments, raise your hand and we’ll check it before you move on to the next.

---

**Code Segment:**

```python
import drinks
d = drinks.Drink()
print( d )
print()
d.print_price_sticker( )
```

**Statements executed:**

- #1, #2, #3,
- #7, #6, #8, #5

**Prints:**

```
Drink @ $ 0.00
Drink
Price: 0.00
```

**Picture:**

```
<table>
<thead>
<tr>
<th>Price</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kind</td>
<td>“Drink”</td>
</tr>
</tbody>
</table>
```

**Code Segment:**

```python
import drinks
d = drinks.CocaCola()
print( d )
print()
d.print_price_sticker( )
```

**Statements executed:**

**Prints:**
For this one, don’t worry about logging the statements that are executed. But predict what the drink object will ‘look like’ and what will be printed.

Code Segment:

```python
import drinks
d = drinks.Milk( 1 )
print( d )
print()
d.print_price_sticker( )
```

Prints:

---

```
For this one, don’t worry about logging the statements that are executed. But predict what the drink object will ‘look like’ and what will be printed.
```

Code Segment:

```python
import drinks
d = drinks.ChocolateMilk( 2 )
print( d )
print()
d.print_price_sticker( )
```

Prints:
Part (c): [Inheritance and typing] A descendant class is called a \textit{subclass} or a \textit{subtype} of all of its ancestor classes. Similarly, an ancestor class is also called a \textit{superclass} or a \textit{supertype} of all of its descendant classes. An instance is considered to be an instance of its own type and also an instance of all of its supertypes (superclasses). Given the class definitions in \texttt{drinks.py}, predict what is printed by the code below.

```python
from drinks import Drink, CocaCola, Milk, ChocolateMilk

generic_drink = Drink()
coke = CocaCola()
milk = Milk( 2 )
choc_milk = ChocolateMilk()

print( type( generic_drink ) == Drink )   # Prints:__________________
print( type( coke ) == Drink )            # Prints:__________________
print( type( milk ) == Milk )             # Prints:__________________
print( type( choc_milk ) == Milk )        # Prints:__________________

print( isinstance(generic_drink, Drink) )          # Prints:_______
print( isinstance( coke, Drink ) )                 # Prints:_______
print( isinstance( generic_drink, CocaCola ) )     # Prints:_______
print( isinstance( choc_milk, Milk ) )             # Prints:_______
print( isinstance( milk, ChocolateMilk ) )         # Prints:_______
print( isinstance( choc_milk, CocaCola ) )         # Prints:_______
print( isinstance( choc_milk, WhiteMilk ) )        # Prints:_______
print( isinstance( choc_milk, object ) )           # Prints:_______
print( isinstance( Milk, Drink ) )                 # Prints:_______
print( isinstance( Milk, type ) )                  # Prints:_______
```

To check your answers, download both \texttt{drinks.py} and \texttt{subtypeDemo.py} (to the same folder); then load and run \texttt{subtypeDemo.py}.
Part (d): Write a program that behaves as follows.

- It repeatedly prompts the user to select from a menu to add a drink to an order or to end the order.
- If the user selects a Coca Cola, the program prompts for a size, either ‘L’ (large, costing $1.75) or ‘S’ (small, costing $1.25). Accept either upper case or lower-case response and use ‘S’ as the default (if the user enters anything other than ‘L’ or ‘l’).
- If the user selects a white milk or a chocolate milk, the program prompts for 0, 1 or 2 indicating the percent fat. Use 2% as the default (if the user enters anything other than 0, 1 or 2). All milk costs $0.75.
- If the user selects to end the order, the program prints out the order and the total cost.

For example:

To add a drink to your order or to quit, enter:
'0' to quit
'1' for a Coca Cola
'2' for a white milk
'3' for a chocolate milk

Enter your selection here: 1
Enter 'L' (large, $1.75) or 'S' (small, the default, $1.25): 1

Enter your selection here: 2
Enter the % fat (0, 1, or 2(the default)):

Enter your selection here: 3
Enter the % fat (0, 1, or 2(the default)): 0

Enter your selection here: ?
Illegal selection: ? Try again.

Enter your selection here: 1
Enter 'L' (large, $1.75) or 'S' (small, the default, $1.25): S

Enter your selection here: 3
Enter the % fat (0, 1, or 2(the default)):

Enter your selection here: 0

Your order:
-----------
Coca-Cola Classic @ $ 1.75
White Milk @ $ 0.75 (2 % fat)
Chocolate Milk @ $ 0.75 (0 % fat)
Coca-Cola Classic @ $ 1.25
Chocolate Milk @ $ 0.75 (2 % fat)

Total is: $5.25