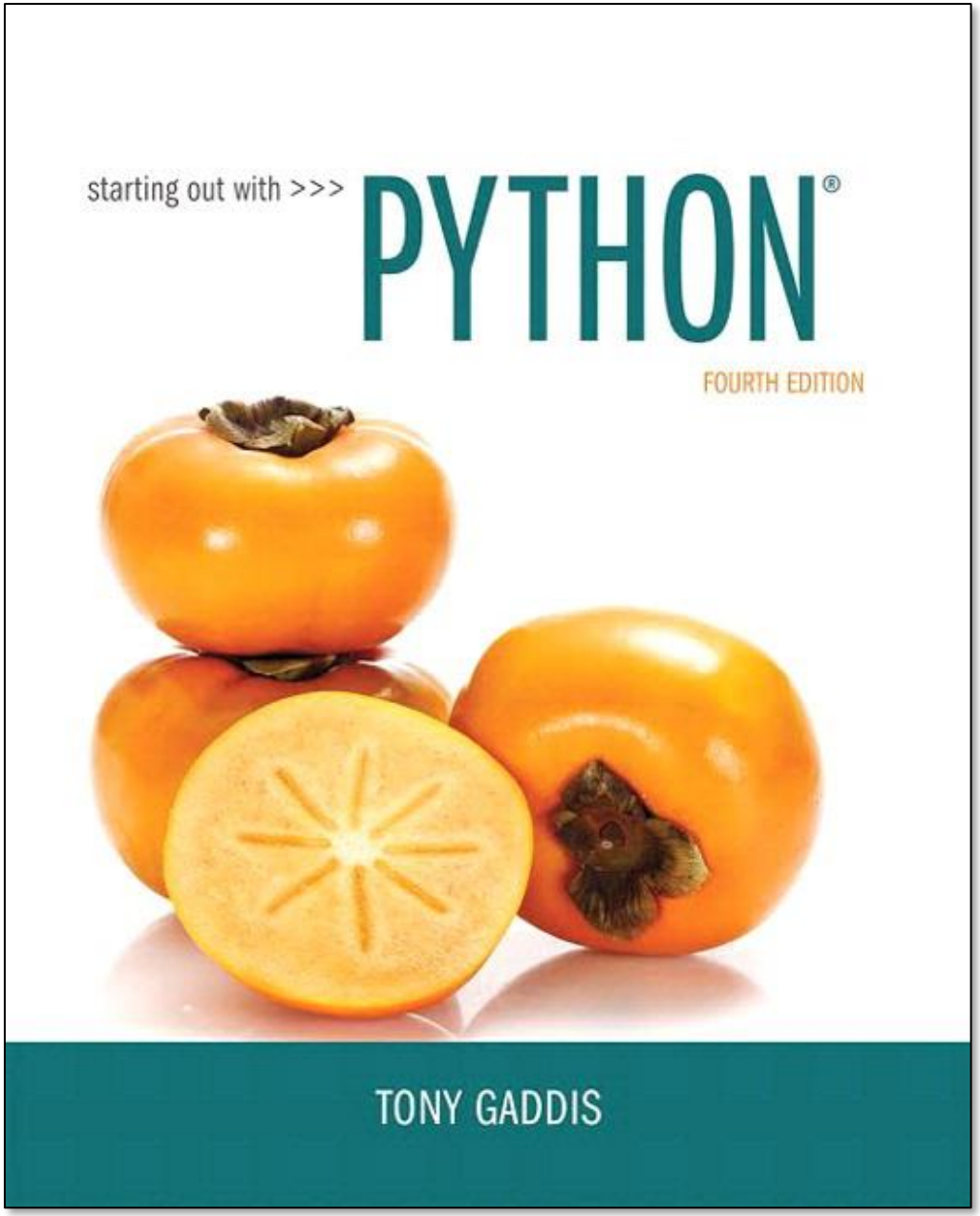
CHAPTER 10

Classes and Object-Oriented Programming



The `__str__` method

- **Object's state: the values of the object's attribute at a given moment**
- **`__str__` method: displays the object's state**
  - Automatically called when the object is passed as an argument to the `print` function
  - Automatically called when the object is passed as an argument to the `str` function

The `__str__` method

```
def __str__(self):
        return 'The balance is $' + format(self.__balance,
        ',.2f')
```

```
# Create a BankAccount object
savings = BankAccountClass.BankAccount(start_bal)

# Display the balance
print(savings)

#Alternate way to display calling the str method
message = str(savings)
print(message)
```

Accessor and Mutator Methods

- **Typically, all of a class's data attributes are private and provide methods to access and change them**
- **Accessor methods: return a value from a class's attribute without changing it**
  - Safe way for code outside the class to retrieve the value of attributes
- **Mutator methods: store or change the value of a data attribute**

Working With Instances

- **Instance attribute: belongs to a specific instance of a class**
  - Created when a method uses the `self` parameter to create an attribute
- **If many instances of a class are created, each would have its own set of attributes**

Working With Instances

```
def main():
# Create three objects from the Coin class.
coin1 = coin.Coin()
coin2 = coin.Coin()
coin3 = coin.Coin()
```

Figure 10-8 The `coin1`, `coin2`, and `coin3` variables reference three `Coin` objects

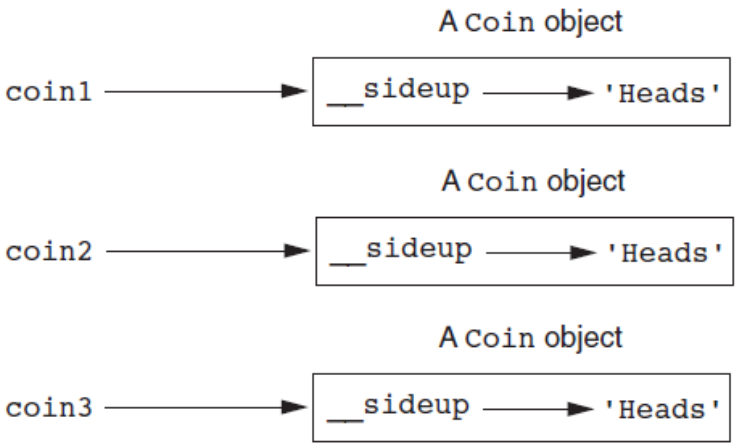
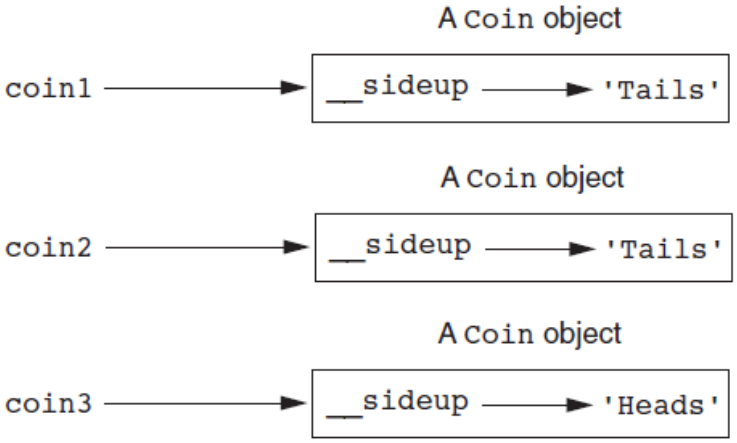


Figure 10-9 The objects after the `toss` method



Passing Objects as Arguments

- **Methods and functions often need to accept objects as arguments**
- **When you pass an object as an argument, you are actually passing a reference to the object**
  - The receiving method or function has access to the actual object
    - Methods of the object can be called within the receiving function or method, and data attributes may be changed using mutator methods

Passing Objects as Arguments

```
# define a function that calls the get_sideup() method of the
# coin object
def show_coin_status(coin_obj):
    print('This side of the coin is up:', coin_obj.get_sideup())


# define a function that calls the toss() method of the
# coin object
def flip(coin_obj):
    coin_obj.toss()

my_coin = coin.Coin()          # create an instance of the coin object
show_coin_status(my_coin)      # call the show_coin_status function
flip(my_coin)                  # call the flip function
```

Techniques for Designing Classes

- **UML diagram: standard diagrams for graphically depicting object-oriented systems**
  - Stands for Unified Modeling Language
- **General layout: box divided into three sections:**
  - Top section: name of the class
  - Middle section: list of data attributes
  - Bottom section: list of class methods

Figure 10-10 General layout of a UML diagram for a class

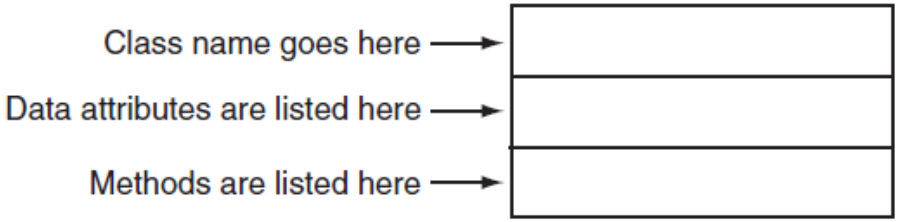
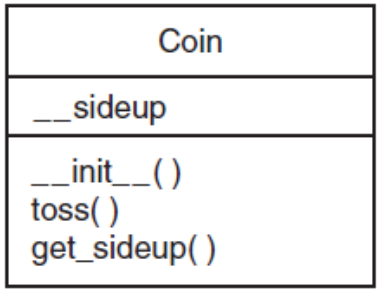
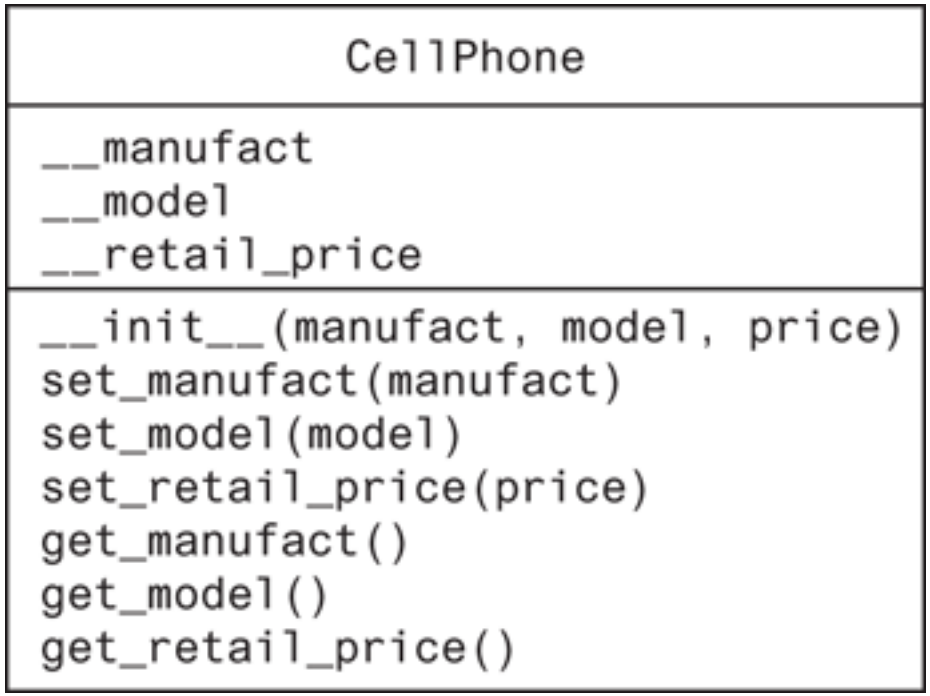


Figure 10-11 UML diagram for the `coin` class



UML of CellPhone Class



Joe's Automotive Shop

Joe's Automotive Shop services foreign cars and specializes in servicing cars made by Mercedes, Porsche, and BMW. When a customer brings a car to the shop, the manager gets the customer's name, address, and telephone number. The manager then determines the make, model, and year of the car and gives the customer a service quote. The service quote shows the estimated parts charges, estimated labor charges, sales tax, and total estimated charges.

Create a class from a UML Diagram

Design a Python program for Joe's Automotive Shop by using classes based on the UML diagrams below and the write-up from previous slide.