Neshat Beheshti
University of Texas Arlington

The Material for this Notebook is Adopted from
Starting Out with Python (4th Edition)- by Tony Gaddis Pearson

# Classes and Object- Oriented Programming

There are primarily two methods of programming:

1. procedural
2. object-oriented

## 1.Procedural Programming

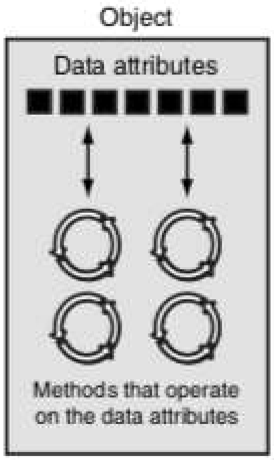
**Procedural programming: writing programs made of functions that perform specific tasks**

- Procedures typically operate on data items that are separate from the procedures
- Data items commonly passed from one procedure to another
- Focus: to create procedures that operate on the program's data

## 2.Object-Oriented Programming

**Object-oriented programming: focused on creating objects**
**Object: entity that contains data and procedures**

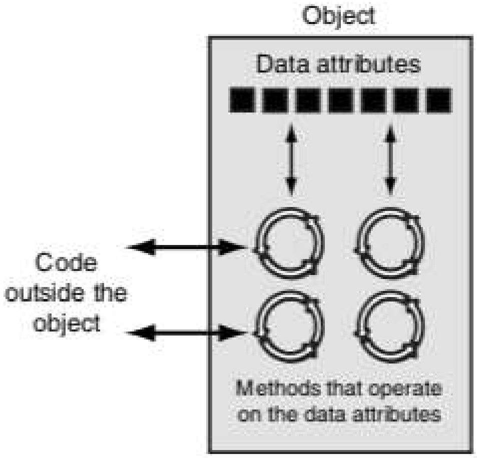
- Data is known as data attributes and procedures are known as methods
- Methods perform operations on the data attributes



*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*

**Encapsulation: combining data and code into a single object**
**Data hiding: object’s data attributes are hidden from code outside the object**

- Access restricted to the object’s methods
  - Protects from accidental corruption
  - Outside code does not need to know internal structure of the object



*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*

## 2.1. Object Reusability

Object reusability: the same object can be used in different programs

- Example: 3D image object can be used for architecture and game programming

## 2.2. An Everyday Example of an Object

Data attributes: define the state of an object

- Example: clock object would have *second*, *minute*, and *hour* data attributes

Public methods: allow external code to manipulate the object

- Example: *set_time, set_alarm_time*

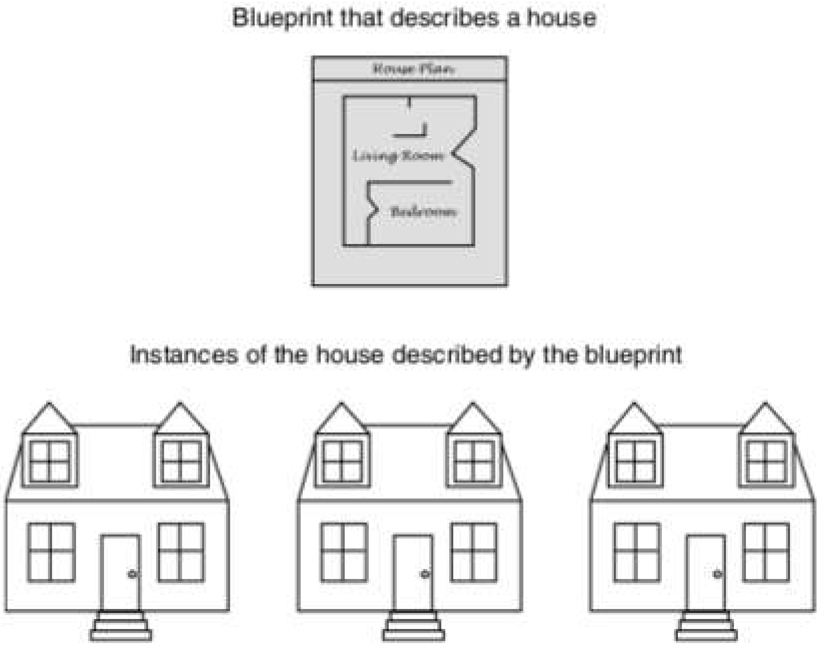
Private methods: used for object's inner workings

- Example: *increment_current_second, increment_current_hour*

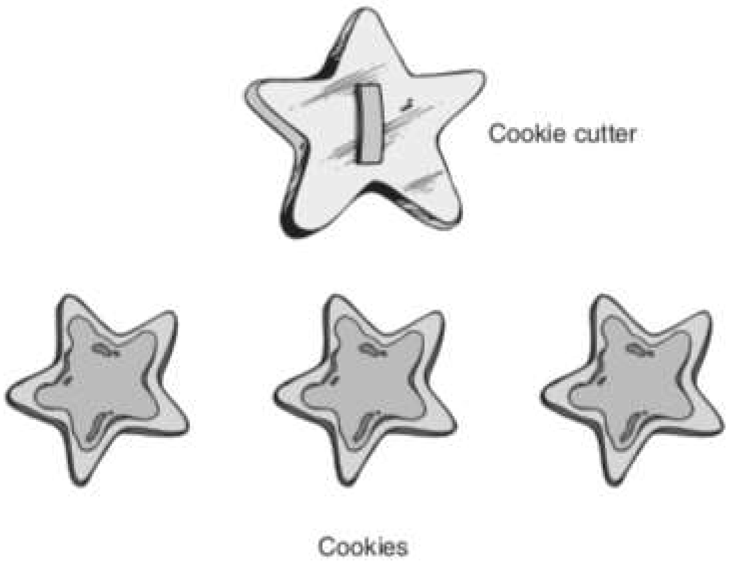
## 3. Classes

**Class: code that specifies the data attributes and methods of a particular type of object**

- Similar to a blueprint of a house or a cookie cutter


Blueprint that describes a house
House Plan
Living Room
Bedroom
Instances of the house described by the blueprint

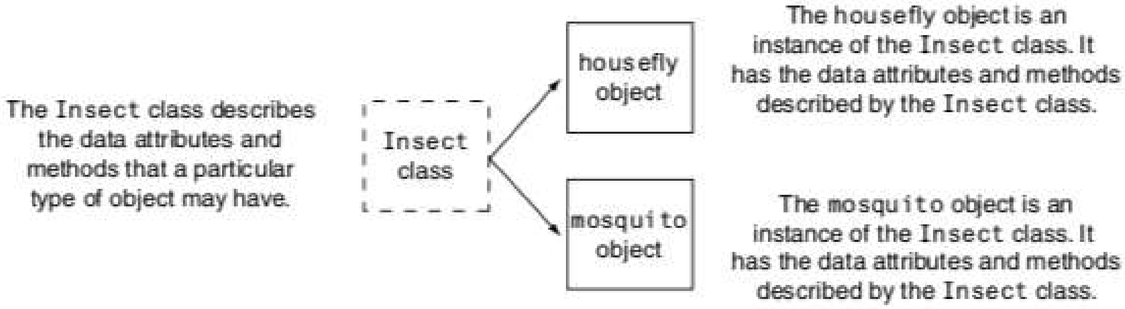
*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*



*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*

**Instance: an object created from a class**

- Similar to a specific house built according to the blueprint or a specific cookie
- There can be many instances of one class


The Insect class describes the data attributes and methods that a particular type of object may have.
Insect class
housefly object
mosquito object
The housefly object is an instance of the Insect class. It has the data attributes and methods described by the Insect class.
The mosquito object is an instance of the Insect class. It has the data attributes and methods described by the Insect class.

*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*

## 3.1. Class Definitions

**Class definition: set of statements that define a class's methods and data attributes**

- Format: begin with *class Class_name*:
    - Class names often start with uppercase letter
- Method definition like any other python function definition
    - self parameter: required in every method in the class – references the specific object that the method is working on

**Initializer method: automatically executed when an instance of the class is created**

- Initializes object’s data attributes and assigns self parameter to the object that was just created
- Format: *def __init__ (self):*
- Usually the first method in a class definition

- To create a new instance of a class call the initializer method
  - Format: *My_instance* = *Class_Name()*
- To call any of the class methods using the created instance, use dot notation
  - Format: *My_instance.method()*
  - Because the *self* parameter references the specific instance of the object, the method will affect this instance
    - Reference to *self* is passed automatically

**Example:**

In [ ]:

```
import random
# The Coin class simulates a coin that can
# be flipped.

class Coin:
    # The __init__ method initializes the
    # sideup data attribute with 'Heads'.

    def __init__(self):
        self.sideup = 'Heads'
        # The toss method generates a random number
        # in the range of 0 through 1. If the number
        # is 0, then sideup is set to 'Heads'.
        # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'
        # The get_sideup method returns the value
        # referenced by sideup.

    def get_sideup(self):
        return self.sideup
```

In [ ]:

```
coin1 = Coin()
coin2 = Coin()
coin3 = Coin()
```

In [ ]:

```
coin2.get_sideup()
```

In [ ]:

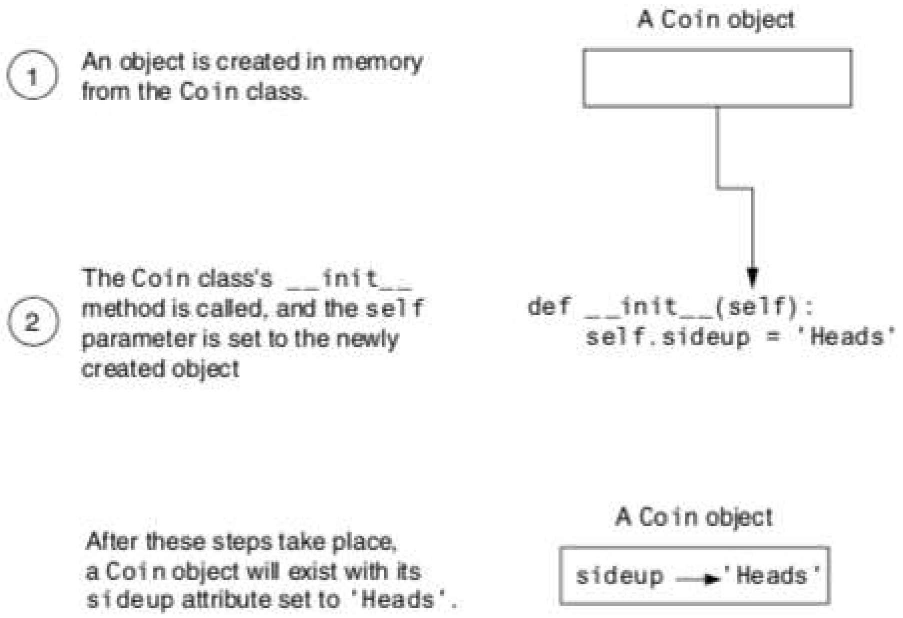
```
coin1.toss()
```

In [ ]:

```
coin1.get_sideup()
```

In [ ]:

```
import random
# The Coin class simulates a coin that can
# be flipped.

class Coin:
    # The __init__ method initializes the
    # sideup data attribute with 'Heads'.

    def __init__(self):
        self.sideup = 'Heads'
        # The toss method generates a random number
        # in the range of 0 through 1. If the number
        # is 0, then sideup is set to 'Heads'.
        # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'
        # The get_sideup method returns the value
        # referenced by sideup.

    def get_sideup(self):
        return self.sideup
# The main function.
def main():
    # Create an object from the Coin class.
    my_coin = Coin()
    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())
    # Toss the coin.
    print('I am tossing the coin ...')
    my_coin.toss()
    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

main()
```



An object is created in memory from the Coin class.

The Coin class's `__init__` method is called, and the self parameter is set to the newly created object

```
def __init__(self):
    self.sideup = 'Heads'
```

After these steps take place, a Coin object will exist with its sideup attribute set to 'Heads'.

*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*

In [ ]:

```
import random
# The Coin class simulates a coin that can be flipped.

class Coin:
    # The __init__ method initializes the
    # sideup data attribute with 'Heads'.

    def __init__(self, init_sideup = 'Heads'):
        self.sideup = init_sideup
        # The toss method generates a random number
        # in the range of 0 through 1. If the number
        # is 0, then sideup is set to 'Heads'.
        # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'
        # The get_sideup method returns the value
        # referenced by sideup.

    def get_sideup(self):
        return self.sideup
```

In [ ]:

```
coin1 = Coin('Heads')
coin2 = Coin('Tails')
coin3 = Coin()
```

In [ ]:

```
coin2.get_sideup()
```

### 3.2. Hiding Attributes and Storing Classes in Modules

An object’s data attributes should be private

- To make sure of this, place two underscores (__) in front of attribute name
  - Example: __current_minute

In [ ]:

```python
import random
# The Coin class simulates a coin that can
# be flipped.

class Coin:
    # The __init__ method initializes the
    # sideup data attribute with 'Heads'.

    def __init__(self):
        self.sideup = 'Heads'
        # The toss method generates a random number
        # in the range of 0 through 1. If the number
        # is 0, then sideup is set to 'Heads'.
        # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'

    # The get_sideup method returns the value referenced by sideup.
    def get_sideup(self):
        return self.sideup

def main():

    # Create an object from the Coin class.
    my_coin = Coin()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am tossing the coin ...')
    my_coin.toss()

    # But now I'm going to cheat! I'm going to
    # directly change the value of the object's
    # sideup attribute to 'Heads'.
    my_coin.sideup = 'Heads'

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

main()
```

In [ ]:

```
my_coin = Coin()
```

In [ ]:

```
my_coin.sideup
```

**Example:**

In [ ]:

```
import random

# The Coin class simulates a coin that can be flipped.
class Coin:

    # The __init__ method initializes the
    # __sideup data attribute with ‘Heads’.
    def __init__(self):
        self.__sideup = 'Heads'

    # The toss method generates a random number
    # in the range of 0 through 1. If the number
    # is 0, then sideup is set to 'Heads'.
    # Otherwise, sideup is set to 'Tails'.
    def toss(self):
        if random.randint(0, 1) == 0:
            self.__sideup = 'Heads'
        else:
            self.__sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup.
    def get_sideup(self):
        return self.__sideup

def main():
    # Create an object from the Coin class.
    my_coin = Coin()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am going to toss the coin ten times:')
    for count in range(10):
        my_coin.toss()
        my_coin.__sideup = 'Tails'
        print(my_coin.get_sideup())

main()
```

## Storing Classes in Modules

Classes can be stored in modules

- Filename for module must end in .py
- Module can be imported to programs that use the class

**Example:**

In [ ]:

```
%%file coin.py
import random

# The Coin class simulates a coin that can be flipped.
class Coin:

    # The __init__ method initializes the __sideup data attribute with 'Heads
    def __init__(self):
        self.__sideup = 'Heads'

    # The toss method generates a random number in the range of 0 through 1.
    #If the number is 0, then sideup is set to 'Heads'.
    # Otherwise, sideup is set to 'Tails'.
    def toss(self):
        if random.randint(0, 1) == 0:
            self.__sideup = 'Heads'
        else:
            self.__sideup = 'Tails'

    # The get_sideup method returns the value referenced by sideup.
    def get_sideup(self):
        return self.__sideup

# This program imports the coin module and
# creates an instance of the Coin class.
```

In [ ]:

```
import coin

def main():
    # Create an object from the Coin class.
    my_coin = coin.Coin()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am going to toss the coin ten times:')
    for count in range(10):
        my_coin.toss()
        print(my_coin.get_sideup())
main()
```

## 3.3.The BankAccount Class – More About Classes

Class methods can have multiple parameters in addition to self

- For __*init*__, parameters needed to create an instance of the class
  - Example: a *BankAccount* object is created with a balance
    - When called, the initializer method receives a value to be assigned to a __*balance* attribute
- For other methods, parameters needed to perform required task
  - Example: *deposit* method amount to be deposited

In [ ]:

```
%%file bankaccount.py

class BankAccount:
# The __init__ method accepts an argument for # the account's balance. It is
# the __balance attribute.
    def __init__(self, bal):
        self.__balance = bal

# The deposit method makes a deposit into the account.
    def deposit(self, amount):
        self.__balance += amount
# The withdraw method withdraws an amount from the account.
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    # The get_balance method returns the account balance.
    def get_balance(self):
        return self.__balance
```

In [ ]:

```python
import bankaccount

def main():
    # Get the starting balance.
    start_bal = float(input('Enter your starting balance: '))

    # Create a BankAccount object.
    savings = bankaccount.BankAccount(start_bal)

    # Deposit the user's paycheck.
    pay = float(input('How much were you paid this week? '))
    print('I will deposit that into your account.')
    savings.deposit(pay)

    # Display the balance.
    print('Your account balance is $', savings.get_balance())

    # Get the amount to withdraw.
    cash = float(input('How much would you like to withdraw? '))
    print('I will withdraw that from your account.')
    savings.withdraw(cash)


    # Display the balance.
    print('Your account balance is $',savings.get_balance())

main()
```

## 3.4. 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

In [ ]:

```
%%file bankaccount2.py
class BankAccount:

    # The __init__ method accepts an argument for the account's balance. It i
    def __init__(self, bal):
        self.__balance = bal

    # The deposit method makes a deposit into the account.
    def deposit(self, amount):
        self.__balance += amount

    # The withdraw method withdraws an amount from the account.
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    # The get_balance method returns the account balance.
    def get_balance(self):
        return self.__balance

    # The __str__ method returns a string indicating the object's state.
    def __str__(self):
        return 'The balance is $' + format(self.__balance, ',.2f')
```

In [ ]:

```
import bankaccount2

def main():

    # Get the starting balance.
    start_bal = float(input('Enter your starting balance: '))

    # Create a BankAccount object.
    savings = bankaccount2.BankAccount(start_bal)

    # Deposit the user's paycheck.
    pay = float(input('How much were you paid this week? '))
    print('I will deposit that into your account.')
    savings.deposit(pay)

    # Display the balance.
    print(savings)

    # Get the amount to withdraw.
    cash = float(input('How much would you like to withdraw? '))
    print('I will withdraw that from your account.')
    savings.withdraw(cash)

    # Display the balance.
    print(savings)

main()
```

In [ ]:

```
account = bankaccount2.BankAccount(1500.0)
message = str(account)
print(message)
```

# 4. Working With Instances

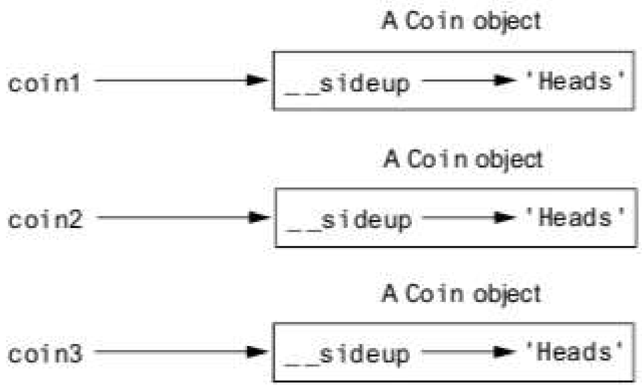
Instance attribute: belongs to a specific instance of a class

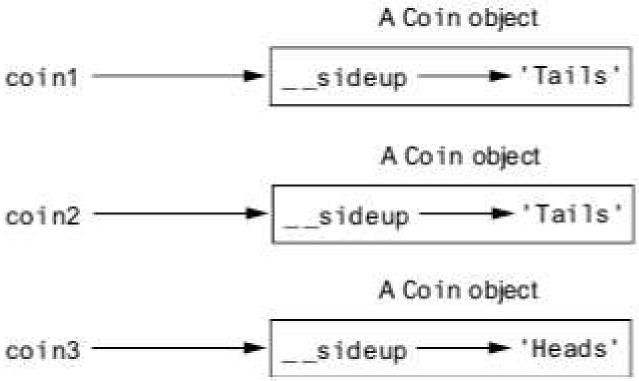
- Created when a method uses the self parameter to create an attribute

If many instances of a class are created, each would have its own set of attributes

In [ ]:

```
import coin

def main():

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

    # Display the side of each coin that is facing up.
    print('I have three coins with these sides up:')
    print(coin1.get_sideup())
    print(coin2.get_sideup())
    print(coin3.get_sideup())
    print()

    # Toss the coin.
    print('I am tossing all three coins ...')
    print()
    coin1.toss()
    coin2.toss()
    coin3.toss()

    # Display the side of each coin that is facing up.
    print('Now here are the sides that are up:')
    print(coin1.get_sideup())
    print(coin2.get_sideup())
    print(coin3.get_sideup())
    print()

main()
```





*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*

In [ ]:

```
%%file cellphone.py
class CellPhone:

    # The __init__ method initializes the attributes.
    def __init__(self, manufact, model, price):
        self.__manufact = manufact
        self.__model = model
        self.__retail_price = price

    # The set_manufact method accepts an argument for # the phone's manufactu
    def set_manufact(self, manufact):
        self.__manufact = manufact

    # The set_model method accepts an argument for the phone's model number.
    def set_model(self, model):
        self.__model = model

    # The set_retail_price method accepts an argument for the phone's retail

    def set_retail_price(self, price):
        self.__retail_price = price

    # The get_manufact method returns the phone's manufacturer.

    def get_manufact(self):
        return self.__manufact

    # The get_model method returns the phone's model number.

    def get_model(self):
        return self.__model

    # The get_retail_price method returns the phone's retail price.

    def get_retail_price(self):
        return self.__retail_price
```

In [ ]:

```
import cellphone

def main():
    # Get the phone data.
    man = input('Enter the manufacturer: ')
    mod = input('Enter the model number: ')
    retail = float(input('Enter the retail price: '))

    # Create an instance of the CellPhone class.
    phone = cellphone.CellPhone(man, mod, retail)

    # Display the data that was entered.
    print('Here is the data that you entered:')
    print('Manufacturer:', phone.get_manufact())
    print('Model Number:', phone.get_model())
    print('Retail Price: $', format(phone.get_retail_price(), ',.2f'), sep=''


main()
```

## 4.1.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

In [ ]:

```
import coin

def main():
    my_coin = coin.Coin()

    # This will display 'Heads'.
    print(my_coin.get_sideup())

    # Pass the object to the flip function.
    flip(my_coin)

    # This might display 'Heads', or it might display 'Tails'.
    print(my_coin.get_sideup())

# The flip function flips a coin.
def flip(coin_obj):
    coin_obj.toss()

main()
```

In [ ]:

```
```

In [ ]:

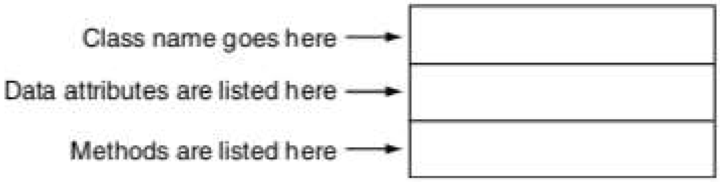
```
import pickle
import cellphone
# Constant for the filename.
FILENAME = 'cellphones.dat'
def main():
# Initialize a variable to control the loop.
    again = 'y'
# Open a file.
    output_file = open(FILENAME, 'wb')
# Get data from the user.
    while again.lower() == 'y':
        # Get cell phone data.
        man = input('Enter the manufacturer: ')
        mod = input('Enter the model number: ')
        retail = float(input('Enter the retail price: '))
# Create a CellPhone object.
        phone = cellphone.CellPhone(man, mod, retail)
# Pickle the object and write it to the file.
        pickle.dump(phone, output_file)
# Get more cell phone data?
        again = input('Enter more phone data? (y/n): ')
# Close the file.
    output_file.close()
    print('The data was written to', FILENAME)
main()
```

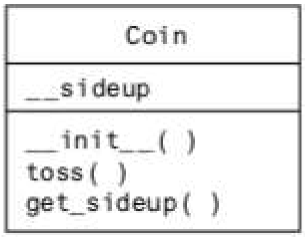
In [ ]:

```
import pickle
import cellphone
# Constant for the filename.
FILENAME = 'cellphones.dat'
def main():
    end_of_file = False
# To indicate end of file
# Open the file.
    input_file = open(FILENAME, 'rb')
# Read to the end of the file.
    while not end_of_file:
        try:
# Unpickle the next object.
            phone = pickle.load(input_file)
# Display the cell phone data.
            display_data(phone)
        except EOFError:
# Set the flag to indicate the end of the file has been reached.
            end_of_file = True
# Close the file.
    input_file.close()
# The display_data function displays the data from the CellPhone object passe
def display_data(phone):
    print('Manufacturer:', phone.get_manufact())
    print('Model Number:', phone.get_model())
    print('Retail Price: $',phone.get_retail_price())
    print()

main()
```

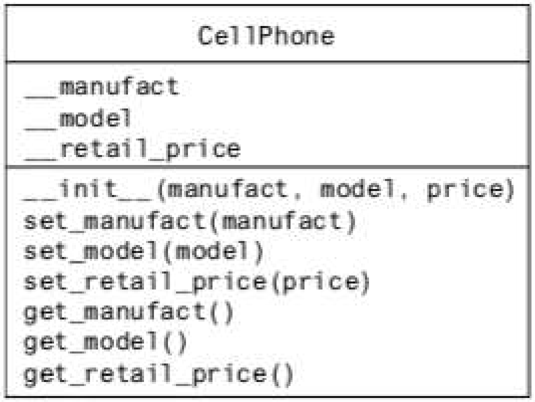
## 5. Techniques for Designing Classes

- 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





```
Coin
__sideup
__init__( )
toss( )
get_sideup( )
```



```
CellPhone
__manufact
__model
__retail_price
__init__(manufact, model, price)
set_manufact(manufact)
set_model(model)
set_retail_price(price)
get_manufact()
get_model()
get_retail_price()
```

*Source: Starting Out with Python (4th Edition)- by Tony Gaddis Pearson*

## 5.1.Finding the Classes in a Problem

- When developing object oriented program, first goal is to identify classes
  - Typically involves identifying the real-world objects that are in the problem
  - Technique for identifying classes:
    1. Get written description of the problem domain
    2. Identify all nouns in the description, each of which is a potential class
    3. Refine the list to include only classes that are relevant to the problem

1.Get written description of the problem domain

- May be written by you or by an expert
- Should include any or all of the following:
  - Physical objects simulated by the program
  - The role played by a person
  - The result of a business event
  - Recordkeeping items

2.Identify all nouns in the description, each of which is a potential class

- Should include noun phrases and pronouns
- Some nouns may appear twice

3.Refine the list to include only classes that are relevant to the problem

- Remove nouns that mean the same thing
- Remove nouns that represent items that the program does not need to be concerned with
- Remove nouns that represent objects, not classes
- Remove nouns that represent simple values that can be assigned to a variable

## 5.2.Identifying a Class’s Responsibilities

A classes responsibilities are:

- The things the class is responsible for knowing
  - Identifying these helps identify the class’s data attributes
- The actions the class is responsible for doing
  - Identifying these helps identify the class’s methods

To find out a class’s responsibilities look at the problem domain

- Deduce required information and actions