



香港中文大學(深圳)  
The Chinese University of Hong Kong, Shenzhen

# **Introduction to Computer Engineering: Programming Applications**

## **Lecture 6 Object Oriented Programming**

**Prof. Junhua Zhao**  
**School of Science and Engineering**

# Object

- In Python, **everything** is an object (number, string, etc)
- You can use the **id()** function and **type()** function to get information about an object

```
>>> n = 3 # n is an integer
>>> id(n)
505408904
>>> type(n)
<class 'int'>
>>> s = "Welcome" # s is a string
>>> id(s)
36201472
>>> type(s)
<class 'str'>
```

# ID and type

- The **id** of an object is automatically assigned a unique integer by Python when the program is executed
- The id for the object **will not be changed** during the execution of the program
- The **type** for the object is determined by Python according to the value of the object

# Variable is actually only a reference

- A **variable** in Python is actually a **reference** to an object.

`n = 3`

`n` → id: 505408904  
The object for int 3

`f = 3.0`

`f` → id: 26647120  
The object for float 3.0

`s = "Welcome"`

`s` → id: 36201472  
The object for str "Welcome"

# Methods

- You can perform **operations** on an object
- The operations are defined using **functions**
- The functions for the objects are called **methods** in Python
- Methods can only be invoked from a specific object

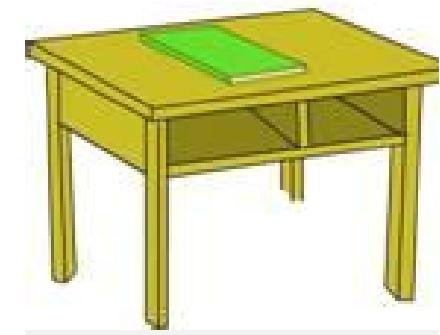
```
>>> s = "Welcome"
>>> s1 = s.lower() # Invoke the lower method
>>> s1
'welcome'
>>> s2 = s.upper() # Invoke the upper method
>>> s2
'WELCOME'
>>>
```

# Why we need object oriented programming?

- Writing a real software is a complicated process
- A sub-field in computer science called **software engineering** is invented to help with the development of large-scale software systems
- People are always trying to invent new ways of writing programs so that software development can be more efficient – structural programming, OO programming, service oriented architecture, etc
- **Object oriented programing** allows us to write program in a way that naturally match the problem that we are trying to solve

# Object

- An **object** represents an entity in the real world that can be distinctly identified.
- **Examples:** a student, a desk, a circle, a button, and even a loan
- An object has a unique **identity**, **state**, and **behaviours**



# Key elements of an object

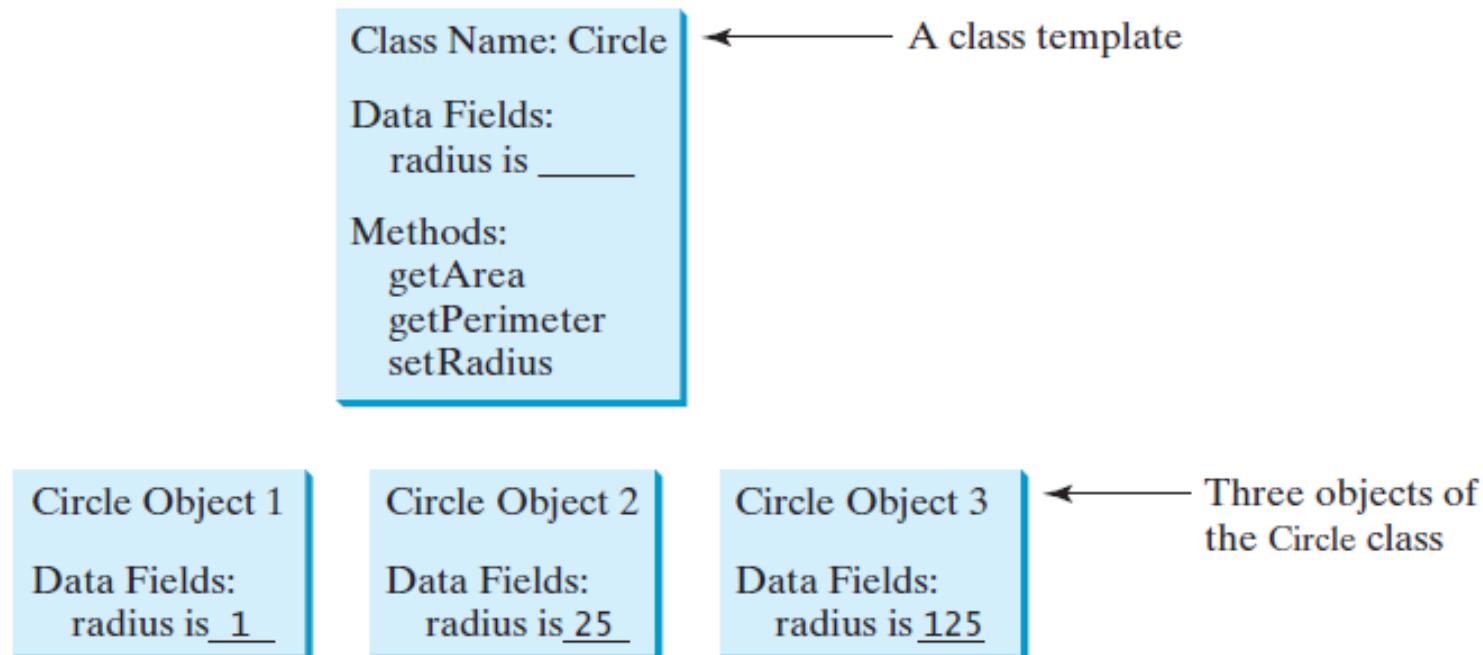
- An object's **identity** is like a person's ID. Python automatically assigns each object **a unique id** for identifying the object at runtime.
- An object's **state** (also known as its **properties** or **attributes**) is represented by variables, called data fields.
- Python uses **methods** to define an object's behavior (also known as its actions). Recall that methods are defined as functions. You make an object perform an action by invoking a method on that object.

# Class

- Objects of the same kind are defined by using a common **class**
- The relationship between Classes and objects is analogous to that between an apple-pie recipe and apple pies
- A Python **class** uses **variables** to store data fields and defines **methods** to perform actions
- A class is a **contract**—also sometimes called a template or blueprint

# Object v.s. class

- An object is an **instance** of a class, and you can create **many instances** of a class
- Creating an instance of a class is referred to as **instantiation**
- The terms object and instance are often used interchangeably



# Class Example

- Class name: Human
- Data fields: Height, body weight, IQ, EQ, education level ...
- Methods:

Eat()

Sleep()

Marry()

Work()

.....

# Object example

Object name: Zhao Junhua

莫哈哈哈哈，  
我要当学霸

## Data Fields:

Weight = 87 kg

Height = 176 cm

IQ = 147

EQ = High

Education level = PhD



## Methods:

Eat()

Sleep()

Marry()

Teach\_programming()

Research()

.....

# Object example

Object name: Trump

**Make America  
Great Again!**

## Data Fields:

Weight = ??

Height = ??

IQ = ??

EQ = ??

Education level = ??



## Methods:

Eat()

Sleep()

Marry()

Run\_election()

Manage\_government()

.....

# Define class

- Python uses the following syntax to define a class
- a class provides a special method, `__init__()`. This method, known as an **initializer**, is invoked to initialize a new object's state when it is created

```
class ClassName:  
    initializer  
    methods
```

# Example

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.radius = radius

    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

    def setRadius(self, radius):
        self.radius = radius
```

```
>>> circle1 = Circle()
>>> circle1.radius
1
>>> circle1.getPerimeter()
6.283185307179586
>>> circle1.getArea()
3.141592653589793
>>> circle1 = Circle(2)
>>> circle1.radius
2
>>> circle1.radius = 10
>>> circle1.getArea()
314.1592653589793
```

# Constructing objects

- Once a **class** is defined, you can **create objects** from the class with a **constructor**. The constructor does two things:
  - ✓ It creates an object in the memory for the class
  - ✓ It invokes the class's `__init__()` method to initialize the object

1. It creates an object in the memory for the class.

object  
Data Fields:

2. It invokes the class's `__init__` method to initialize the object. The `self` parameter in the `__init__` method is automatically set to reference the object that was just created.

`__init__(self, ...)`

# Self

- All methods, including the initializer, have the first parameter `self`
- This parameter refers to the **object that invokes the method.**
- The `self` parameter in the `__init__()` method is automatically set to reference the object that was just created

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.radius = radius

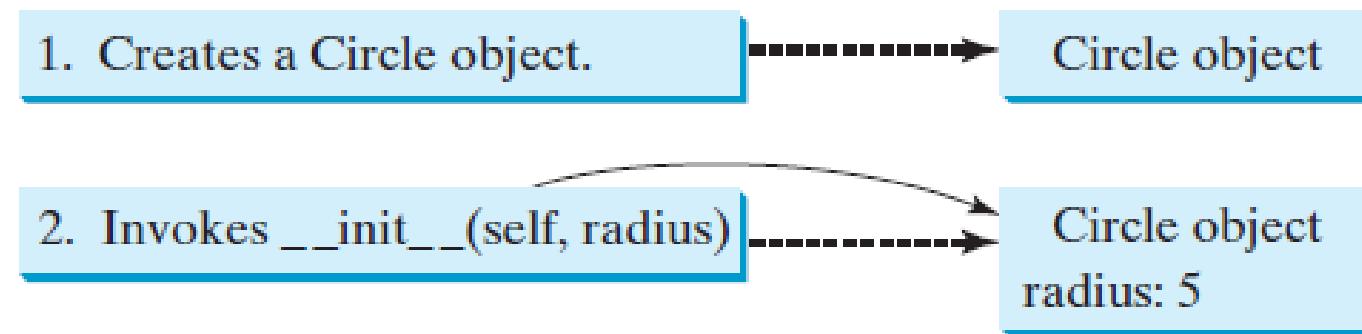
    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

    def setRadius(self, radius):
        self.radius = radius
```

# Constructor arguments

- The arguments of the constructor **match** the parameters in the `__init__()` method **without self**



- The initializer in the Circle class has a **default** radius value, then the constructor **without arguments** will assign the default values to data fields

# Accessing member of objects

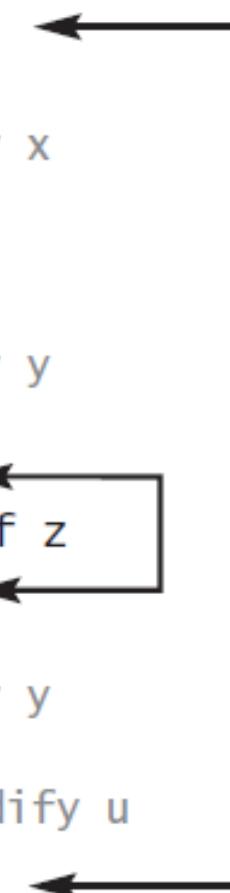
- Data fields are also called **instance variables**, because each object (instance) has a specific value for a data field
- Methods are also called **instance methods**, because a method which is invoked by an object (instance) will perform actions based on the data fields of that object
- You can access the object's data fields and invoke its methods by using the **dot operator (.)**, also known as the **object member access operator**

```
>>> c = Circle(5)
>>> c.radius
5
>>> c.getPerimeter()
31.41592653589793
>>> c.getArea()
78.53981633974483
>>>
```

# Scope of self

- The scope of an instance variable is the entire class once it is created
- You can also create local variables in a method
- The scope of a local variable is within the method

```
def ClassName:  
    def __init__(self, ...):  
        self.x = 1 # Create/modify x  
        ...  
  
    def m1(self, ...):  
        self.y = 2 # Create/modify y  
        ...  
        z = 5 # Create/modify z  
        ...  
        Scope of z  
  
    def m2(self, ...):  
        self.y = 3 # Create/modify y  
        ...  
        u = self.x + 1 # Create/modify u  
        self.m1(...) # Invoke m1
```



Scope of self.x and self.y

Scope of z

# Example

```
def main():
    # Create a circle with radius 1
    circle1 = Circle()
    print("The area of the circle of radius",
          circle1.radius, "is", circle1.getArea())

    # Create a circle with radius 25
    circle2 = Circle(25)
    print("The area of the circle of radius",
          circle2.radius, "is", circle2.getArea())

    # Create a circle with radius 125
    circle3 = Circle(125)
    print("The area of the circle of radius",
          circle3.radius, "is", circle3.getArea())

    # Modify circle radius
    circle2.radius = 100 # or circle2.setRadius(100)
    print("The area of the circle of radius",
          circle2.radius, "is", circle2.getArea())

main() # Call the main function
```

# Result

The area of the circle of radius 1.0 is 3.141592653589793

The area of the circle of radius 25.0 is 1963.4954084936207

The area of the circle of radius 125.0 is 49087.385212340516

The area of the circle of radius 100.0 is 31415.926535897932

# What is wrong with this program?

```
class A:  
    def __init__(self, i):  
        self.i = i  
  
def main():  
    a = A()  
    print(a.i)  
  
main() # Call the main function
```

# What is wrong with these programs?

```
class A:  
    # Construct an object of the class  
    def A(self):  
        radius = 3
```

```
class A:  
    # Construct an object of the class  
    def __init__(self):  
        radius = 3  
  
    def setRadius(radius):  
        self.radius = radius
```

# Example: TV class

TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: bool	Indicates whether this TV is on/off.
TV()	Constructs a default TV object.
turnOn(): None	Turns on this TV.
turnOff(): None	Turns off this TV.
getChannel(): int	Returns the channel for this TV.
setChannel(channel: int): None	Sets a new channel for this TV.
getVolume(): int	Gets the volume level for this TV.
setVolume(volumeLevel: int): None	Sets a new volume level for this TV.
channelUp(): None	Increases the channel number by 1.
channelDown(): None	Decreases the channel number by 1.
volumeUp(): None	Increases the volume level by 1.
volumeDown(): None	Decreases the volume level by 1.

```
class TV:  
    def __init__(self):  
        self.channel = 1 # Default channel is 1  
        self.volumeLevel = 1 # Default volume level is 1  
        self.on = False # Initially, TV is off  
  
    def turnOn(self):  
        self.on = True  
  
    def turnOff(self):  
        self.on = False  
  
    def getChannel(self):  
        return self.channel  
  
    def setChannel(self, channel):  
        if self.on and 1 <= channel <= 120:  
            self.channel = channel  
  
    def getVolumeLevel(self):  
        return self.volumeLevel  
  
    def setVolume(self, volumeLevel):  
        if self.on and \  
            1 <= volumeLevel <= 7:  
            self.volumeLevel = volumeLevel  
  
    def channelUp(self):
```

```
        if self.on and self.channel < 120:  
            self.channel += 1  
  
    def channelDown(self):  
        if self.on and self.channel > 1:  
            self.channel -= 1  
  
    def volumeUp(self):  
        if self.on and self.volumeLevel < 7:  
            self.volumeLevel += 1  
  
    def volumeDown(self):  
        if self.on and self.volumeLevel > 1:  
            self.volumeLevel -= 1
```

# Example: the code to use TV class

```
from TV import TV

def main():
    tv1 = TV()
    tv1.turnOn()
    tv1.setChannel(30)
    tv1.setVolume(3)

    tv2 = TV()
    tv2.turnOn()
    tv2.channelUp()
    tv2.channelUp()
    tv2.volumeUp()

    print("tv1's channel is", tv1.getChannel(),
          "and volume level is", tv1.getVolumeLevel())
    print("tv2's channel is", tv2.getChannel(),
          "and volume level is", tv2.getVolumeLevel())

main() # Call the main function
```

tv1's channel is 30 and volume level is 3  
tv2's channel is 3 and volume level is 2

# Practice

```
class Count:  
    def __init__(self, count = 0):  
        self.count = count  
  
def main():  
    c = Count()  
    n = 1  
    m(c, n)  
  
    print("count is", c.count)  
    print("n is", n)  
  
def m(c, n):  
    c = Count(5)  
    n = 3  
  
main() # Call the main function
```

count is 0  
n is 1

- What would be the output of the above program?

# Hiding data fields

- Direct access of a data field in an object is **not good !!**
- First, data may be **tampered with**
- Second, the class becomes **difficult to maintain** and  
vulnerable to bugs

# Private data fields

- Prevent other programmers from directly accessing the data fields of your class is a common industrial practice
- This is known as **data hiding**
- This can be done by defining **private data fields**

# Private data fields

- In Python, the **private data fields** are defined with two **leading underscores**. You can also define a **private method** named with two leading underscores
- Private data fields and methods can be accessed within a class, but they **cannot be accessed outside the class**
- Define some **methods** to allow access to private data fields

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def getPerimeter(self):
        return 2 * self.__radius * math.pi

    def getArea(self):
        return self.__radius * self.__radius * math.pi
```

# Practice

```
class A:  
    def __init__(self, i):  
        self.__i = i  
  
def main():  
    a = A(5)  
    print(a.__i)  
  
main() # Call the main function
```

- What is the problem with this program?

# Practice

```
def main():
    a = A()
    a.print()

class A:
    def __init__(self, newS = "Welcome"):
        self.__s = newS

    def print(self):
        print(self.__s)

main() # Call the main function
```

- Is the above code correct? If yes, what would be the output?

# Practice

```
class A:  
    def __init__(self, on):  
        self.__on = not on  
  
def main():  
    a = A(False)  
    print(a.on)  
  
main() # Call the main function
```

- Is the above code correct? If not, how do we fix it?

# Abstraction

- Abstraction means separate the **implementation** of a part of code from the **usage** of that code
- In software engineering, there are many levels of abstraction, a commonly used one is called **function abstraction**
- Function abstraction means separating the implementation of a function from its usage
- Abstraction makes your code easy to **maintain, debug** and **reuse**

# Example

```
# Return the gcd of two integers
def gcd(n1, n2):
    gcd = 1 # Initial gcd is 1
    k = 2   # Possible gcd

    while k <= n1 and k <= n2:
        if n1 % k == 0 and n2 % k == 0:
            gcd = k # Update gcd
        k += 1

    return gcd # Return gcd

# Prompt the user to enter two integers
n1 = eval(input("Enter the first integer: "))
n2 = eval(input("Enter the second integer: "))

print("The greatest common divisor for", n1,
      "and", n2, "is", gcd(n1, n2))
```

```
# Check whether number is prime
def isPrime(number):
    divisor = 2
    while divisor <= number / 2:
        if number % divisor == 0:
            # If true, number is not prime
            return False # number is not a prime
        divisor += 1

    return True # number is prime

def printPrimeNumbers(numberOfPrimes):
    NUMBER_OF_PRIMES = 50 # Number of primes to display
    NUMBER_OF_PRIMES_PER_LINE = 10 # Display 10 per line
    count = 0 # Count the number of prime numbers
    number = 2 # A number to be tested for primeness

    # Repeatedly find prime numbers
    while count < numberOfPrimes:
        # Print the prime number and increase the count
        if isPrime(number):
            count += 1 # Increase the count

            print(number, end = " ")
            if count % NUMBER_OF_PRIMES_PER_LINE == 0:
                # Print the number and advance to the new line
                print()

        # Check if the next number is prime
        number += 1

def main():
    print("The first 50 prime numbers are")
```

Write and maintain isPrime()



Programmer 1

Write and maintain printPrimeNumbers()



Programmer 2

If we write everything together...

```
def printPrimeNumbers(numberOfPrimes):
    NUMBER_OF_PRIMES = 50 # Number of primes to display
    NUMBER_OF_PRIMES_PER_LINE = 10 # Display 10 per line
    count = 0 # Count the number of prime numbers
    number = 2 # A number to be tested for primeness

    # Repeatedly find prime numbers
    while count < numberOfPrimes:

        #Determine whether a number is a prime number
        isPrime = True
        divisor = 2
        while (divisor<=number/2):
            if number%divisor ==0:
                isPrime = False
                break
            divisor +=1

        # Print the prime number and increase the count
        if isPrime==True:
            count += 1 # Increase the count

            print(number, end = " ")
            if count % NUMBER_OF_PRIMES_PER_LINE == 0:
                # Print the number and advance to the new line
                print()

        # Check if the next number is prime
        number += 1

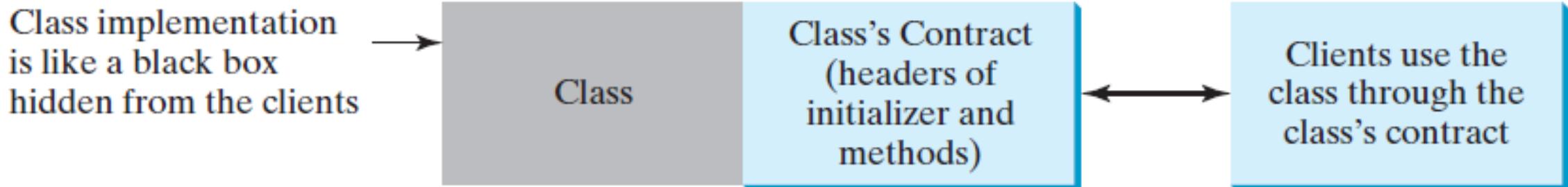
printPrimeNumbers(20)
```

# Class abstraction and encapsulation

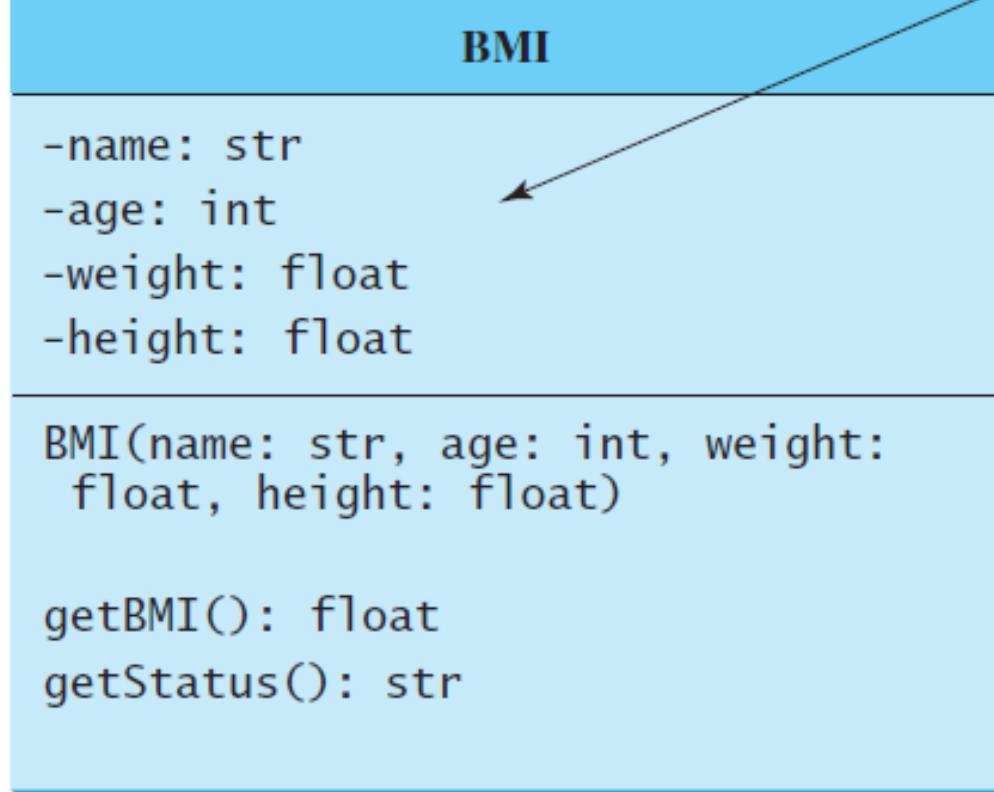
- **Class abstraction** means separating class implementation from the use of a class
- The class implementation details are **invisible** from the user
- The class's collection of methods, together with the description of how these methods are expected to behave, serves as the class's **contract** with the client

# Class abstraction and encapsulation

- The user of the class does not need to know how the class is implemented. The details of implementation are **encapsulated** and **hidden** from the user.
- This is known as **class encapsulation**
- In essence, encapsulation combines data and methods into a single object and hides the data fields and method implementation from the user



# Example – BMI calculation



The get methods for these data fields are provided in the class, but are omitted in the UML diagram for brevity.

The name of the person.  
The age of the person.  
The weight of the person in pounds.  
The height of the person in inches.

Creates a BMI object with the specified name, age (the default is 20), weight, and height.

Returns the BMI.  
Returns the BMI status (e.g., Normal, Overweight, etc.).

# The code to use BMI class

```
from BMI import BMI

def main():
    bmi1 = BMI("John Doe", 18, 145, 70)
    print("The BMI for", bmi1.getName(), "is",
          bmi1.getBMI(), bmi1.getStatus())

    bmi2 = BMI("Peter King", 50, 215, 70)
    print("The BMI for", bmi2.getName(), "is",
          bmi2.getBMI(), bmi2.getStatus())

main() # Call the main function
```

- We can use the BMI class if you have its **contract**
- You **don't need to know** the details about how it is implemented!!

# The BMI class

```
class BMI:  
    def __init__(self, name, age, weight, height):  
        self.__name = name  
        self.__age = age  
        self.__weight = weight  
        self.__height = height  
  
    def getBMI(self):  
        KILOGRAMS_PER_POUND = 0.45359237  
        METERS_PER_INCH = 0.0254  
        bmi = self.__weight * KILOGRAMS_PER_POUND / \  
            ((self.__height * METERS_PER_INCH) * \  
             (self.__height * METERS_PER_INCH))  
        return round(bmi * 100) / 100
```

```
def getStatus(self):  
    bmi = self.getBMI()  
    if bmi < 18.5:  
        return "Underweight"  
    elif bmi < 25:  
        return "Normal"  
    elif bmi < 30:  
        return "Overweight"  
    else:  
        return "Obese"  
  
def getName(self):  
    return self.__name  
  
def getAge(self):  
    return self.__age  
  
def getWeight(self):  
    return self.__weight  
  
def getHeight(self):  
    return self.__height
```

# Practice

(The Rectangle class) Following the example of the Circle class, design a class named Rectangle to represent a rectangle. The class contains:

- Two data fields named width and height.
- A constructor that creates a rectangle with the specified width and height. The default values are 1 and 2 for the width and height, respectively.
- A method named getArea() that returns the area of this rectangle.
- A method named getPerimeter() that returns the perimeter.

# Answer: Rectangle Class

```
class Rectangle:  
    # Construct a rectangle object  
    def __init__(self, width = 1, height = 2):  
        self.width = width  
        self.height = height  
  
    def getArea(self):  
        return self.width * self.height  
  
    def getPerimeter(self):  
        return 2 * (self.width + self.height)
```

# Answer: main() function

```
def main():
    # Create a rectangle with width 4 and height 40
    r1 = Rectangle(4, 40)
    print("The width of the rectangle is", r1.width)
    print("The height of the rectangle is", r1.height)
    print("The area of the rectangle is", r1.getArea())
    print("The perimeter of the rectangle is", r1.getPerimeter())

    # Create a rectangle with width 3.5 and height 35.9
    r1 = Rectangle(3.5, 35.9)
    print("The width of the rectangle is", r1.width)
    print("The height of the rectangle is", r1.height)
    print("The area of the rectangle is", r1.getArea())
    print("The perimeter of the rectangle is", r1.getPerimeter())

main()
```

# Practice

(The Stock class) Design a class named Stock to represent a company's stock that contains:

- A private string data field named symbol for the stock's symbol.
- A private string data field named name for the stock's name.
- A private float data field named previousClosingPrice that stores the stock price for the previous day.
- A private float data field named currentPrice that stores the stock price for the current time.
- A constructor that creates a stock with the specified symbol, name, previous price, and current price.
- A get method for returning the stock name.
- A get method for returning the stock symbol.
- Get and set methods for getting/setting the stock's previous price.
- Get and set methods for getting/setting the stock's current price.
- A method named getChangePercent() that returns the percentage changed from previousClosingPrice to currentPrice.

# Answer: Stock Class

```
class Stock:  
    # Construct a stock object  
    def __init__(self, name, symbol, previousPrice, currentPrice):  
        self.__name = name  
        self.__symbol = symbol  
        self.__previousPrice = previousPrice  
        self.__currentPrice = currentPrice  
  
    def getName(self):  
        return self.__name  
  
    def getSymbol(self):  
        return self.__symbol  
  
    def getPreviousPrice(self):  
        return self.__previousPrice  
  
    def getCurrentPrice(self):  
        return self.__currentPrice  
  
    def setPreviousPrice(self, previousPrice):  
        self.__previousPrice = previousPrice  
  
    def setCurrentPrice(self, currentPrice):  
        self.__currentPrice = currentPrice  
  
    def getChangePercent(self):  
        return format((self.__currentPrice - self.__previousPrice) * 100 / self.__previousPrice, "5.2f") + "%"
```

# Answer: main() function

```
def main():
    # Create a stock
    stock = Stock("Intel", "INTC", 20.5, 20.35)
    print("The price change is", stock.getChangePercent())

main()
```

# Practice

(Algebra: quadratic equations) Design a class named **QuadraticEquation** for a quadratic equation  $ax^2 + bx + c = 0$ . The class contains:

- The private data fields **a**, **b**, and **c** that represent three coefficients.
- A constructor for the arguments for **a**, **b**, and **c**.
- Three **get** methods for **a**, **b**, and **c**.
- A method named **getDiscriminant()** that returns the discriminant, which is  $b^2 - 4ac$ .
- The methods named **getRoot1()** and **getRoot2()** for returning the two roots of the equation using these formulas:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

These methods are useful only if the discriminant is nonnegative. Let these methods return **0** if the discriminant is negative.

```
import math

class QuadraticEquation:
    def __init__(self, a, b, c):
        self.__a = a
        self.__b = b
        self.__c = c

    def getA(self):
        return self.__a

    def getB(self):
        return self.__b

    def getC(self):
        return self.__c

    def getDiscriminant(self):
        return self.__b * self.__b - 4 * self.__a * self.__c

    def getRoot1(self):
        if self.getDiscriminant() < 0:
            return 0
        else:
            return (-self.__b + self.getDiscriminant()) / (2 * self.__a)

    def getRoot2(self):
        if self.getDiscriminant() < 0:
            return 0
        else:
            return (-self.__b - self.getDiscriminant()) / (2 * self.__a)
```

```
def main():
    a, b, c = eval(input("Enter a, b, c: "))
    equation = QuadraticEquation(a, b, c)
    discriminant = equation.getDiscriminant()

    if discriminant < 0:
        print("The equation has no roots")
    elif discriminant == 0:
        print("The root is", equation.getRoot1())
    else: # (discriminant >= 0)
        print("The roots are", equation.getRoot1(), "and", equation.getRoot2())

main()
```