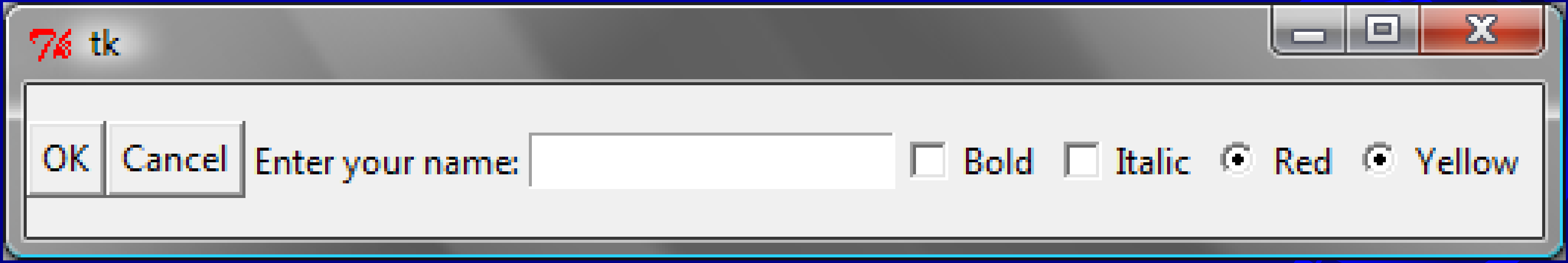
Chapter 7 Object-Oriented Programming

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Motivations

- After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, and functions.
- However, these Python features are not sufficient for developing graphical user interfaces and large scale software systems.
- Suppose you want to develop a graphical user interface as shown below. How do you program it?



© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Objectives

- To describe objects and classes, and use classes to model objects (§7.2).
- To define classes (§7.2.1).
- To construct an object using a constructor that invokes the initializer to create and initialize data fields (§7.2.2).
- To access the members of objects using the dot operator (.) (§7.2.3).
- To reference an object itself with the self parameter (§7.2.4).
- To use UML graphical notation to describe classes and objects (§7.3).
- To distinguish between immutable and mutable (§7.4).
- To hide data fields to prevent data corruption and make classes easy to maintain (§7.5).
- To apply class abstraction and encapsulation to software development (§7.6).
- To explore the differences between the procedural paradigm and the object-oriented paradigm (§7.7).

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Procedural Programming

- Procedural programming: writing programs made of functions that perform specific tasks
  - Procedures (functions) 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

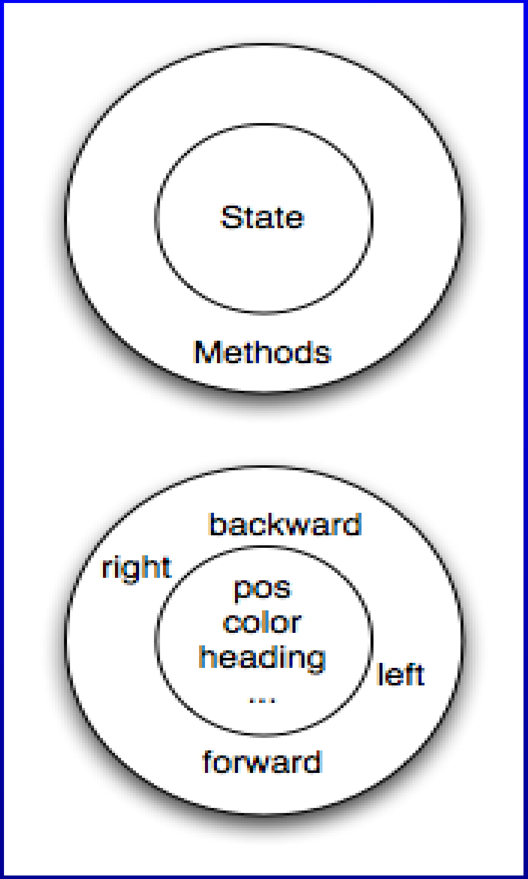
© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

OO Programming Concepts

- Object-oriented programming (OOP) involves programming using objects.
- An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behaviors.
- The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values. The *behavior* of an object is defined by a set of methods.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Object's State and Methods

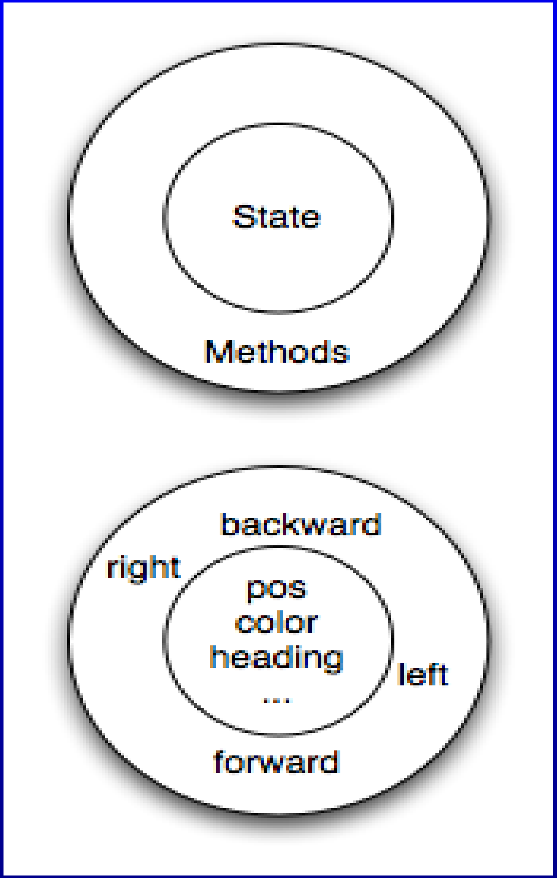


- In Python, every value is actually an object. Whether it be a turtle, a list, or even an integer, they are all objects.
- Programs manipulate those objects either by performing computation with them or by asking them to perform methods.
- To be more specific, we say that an object has a state and a collection of methods that it can perform.

```
>>>turtle.color("blue")
>>> turtle.position()
(0.00,0.00)
>>> turtle.forward(25)
>>> turtle.position()
(25.00,0.00)
>>> turtle.forward(-75)
>>> turtle.position()
(-50.00,0.00)
```

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Object's State and Methods

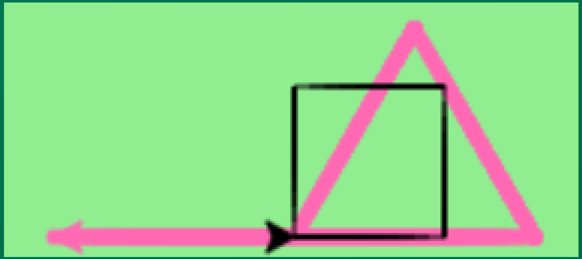


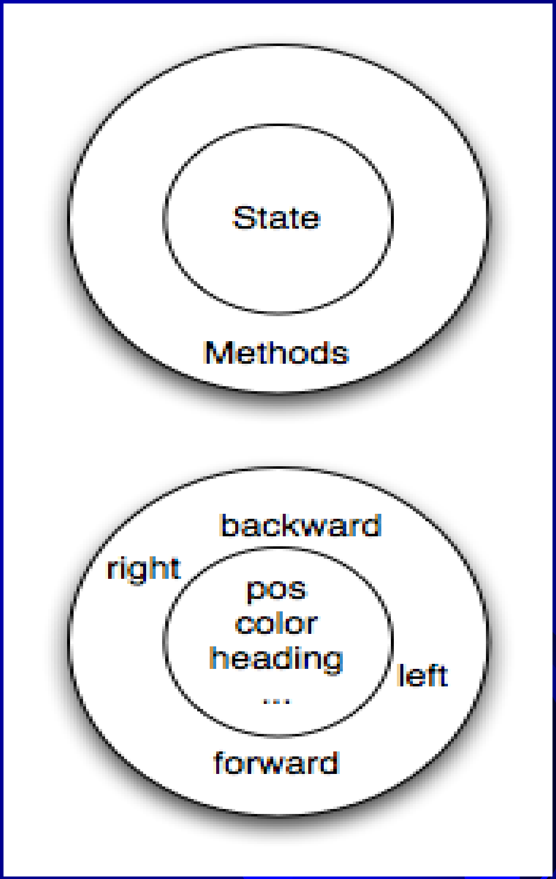
- The state of an object represents those things that the object knows about itself. For example, as we have seen with turtle objects, each turtle has a state consisting of the turtle's position, its color, its heading and so on.
- Each turtle also has the ability to go forward, backward, or turn right or left.
- Individual turtles are different in that even though they are all turtles, they differ in the specific values of the individual state attributes (maybe they are in a different location or have a different heading).

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Object's State and Methods

```
import turtle
wn = turtle.Screen()            # Set up the window and its attributes
wn.bgcolor("lightgreen")

tess = turtle.Turtle()          # create tess and set some attributes
tess.color("hotpink")
tess.pensize(5)

alex = turtle.Turtle()          # create alex

tess.forward(80)                # Let tess draw an equilateral triangle
tess.left(120)
tess.forward(80)
tess.left(120)
tess.forward(80)
tess.left(120)                # complete the triangle

tess.right(180)               # turn tess around
tess.forward(80)                # move her away from the origin

alex.forward(50)                # make alex draw a square
alex.left(90)
alex.forward(50)
alex.left(90)
alex.forward(50)
alex.left(90)
alex.forward(50)
alex.left(90)

wn.exitonclick()
```





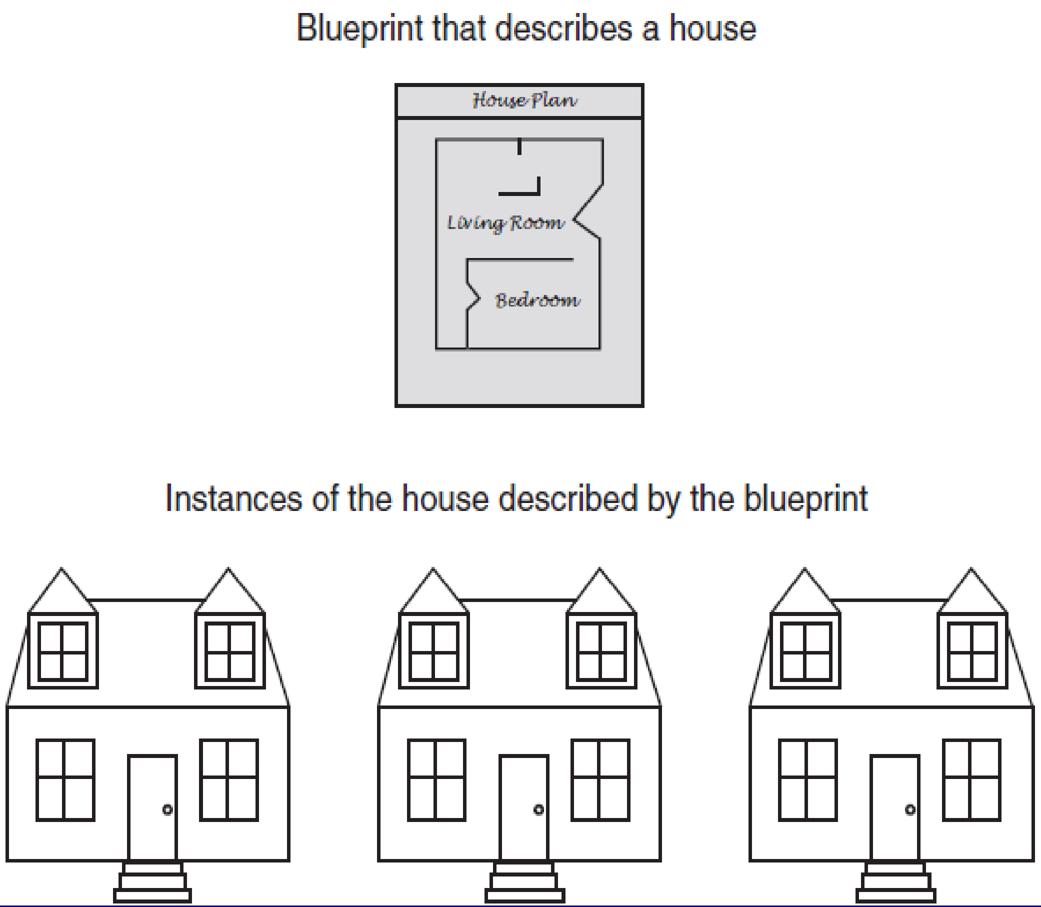
Classes

- 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
  - It is simply a template that we construct objects from it.
- 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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Classes (cont'd.)

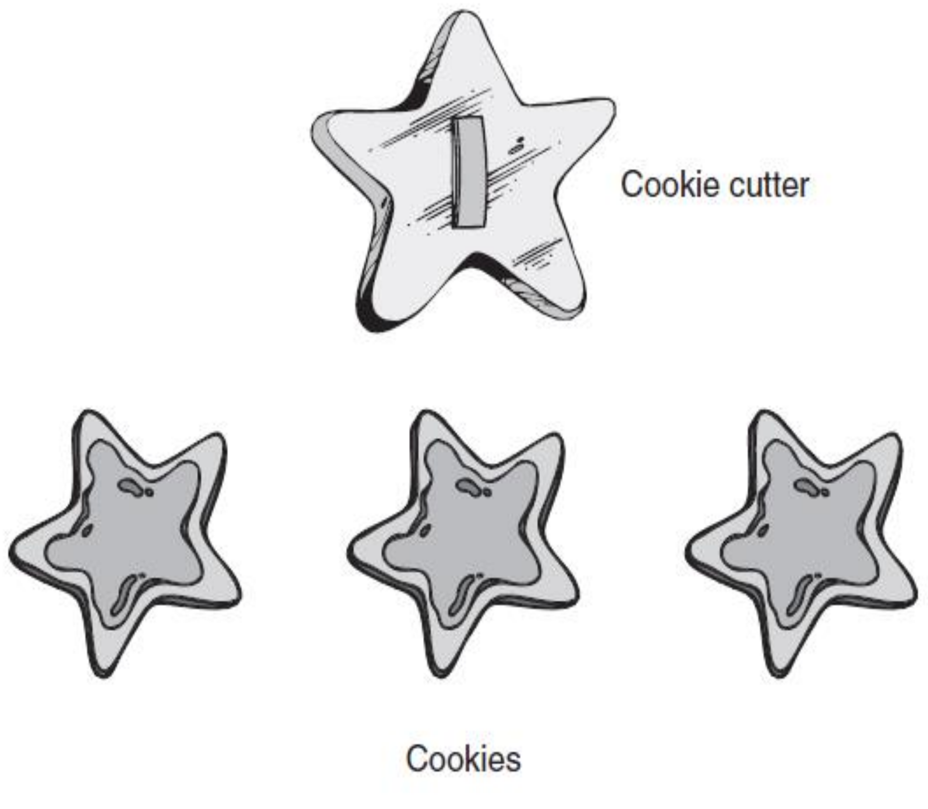
Figure 10-3 A blueprint and houses built from the blueprint


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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

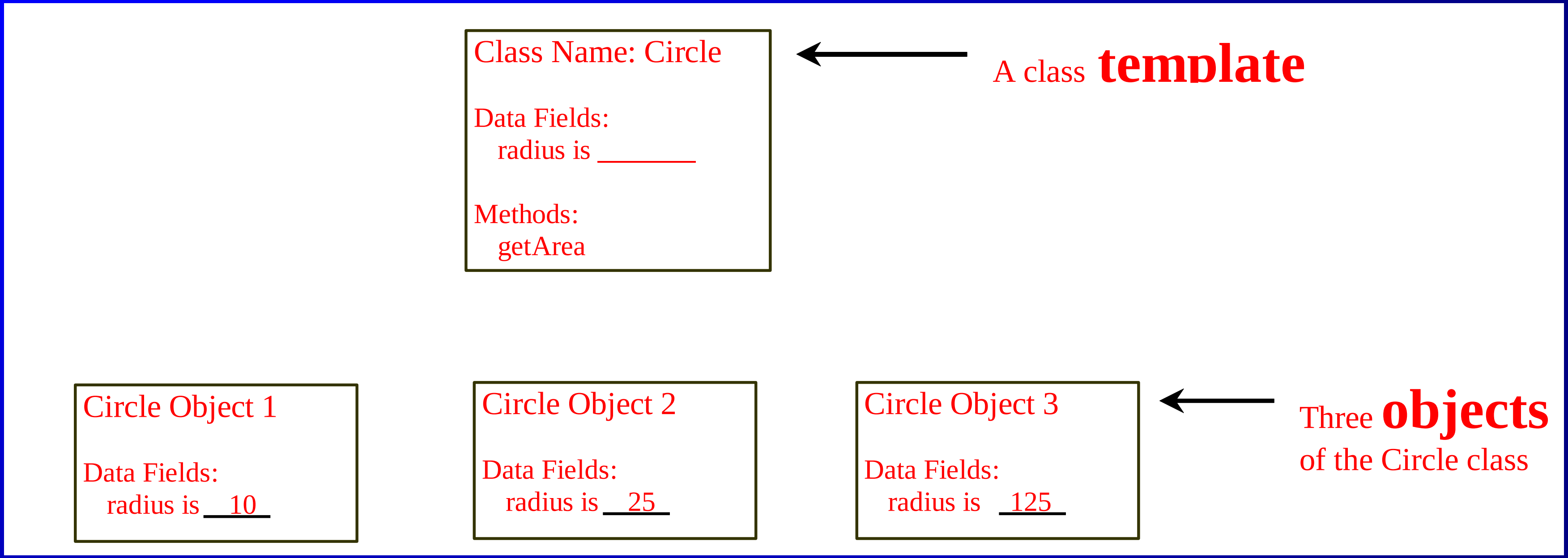
Classes (cont'd.)

Figure 10-4 The cookie cutter metaphor



© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Objects



An object has both a state and behavior. The state defines the object (Data Fields), and the behavior defines what the object does (Methods).

Classes

A Python class uses variables to store data fields and defines methods to perform actions. Additionally, a class provides a special type method, known as *initializer*, which is invoked to create a new object. An initializer can perform any action, but initializer is designed to perform initializing actions, such as creating the data fields of objects.

```
class ClassName:
    initializer
    methods
```

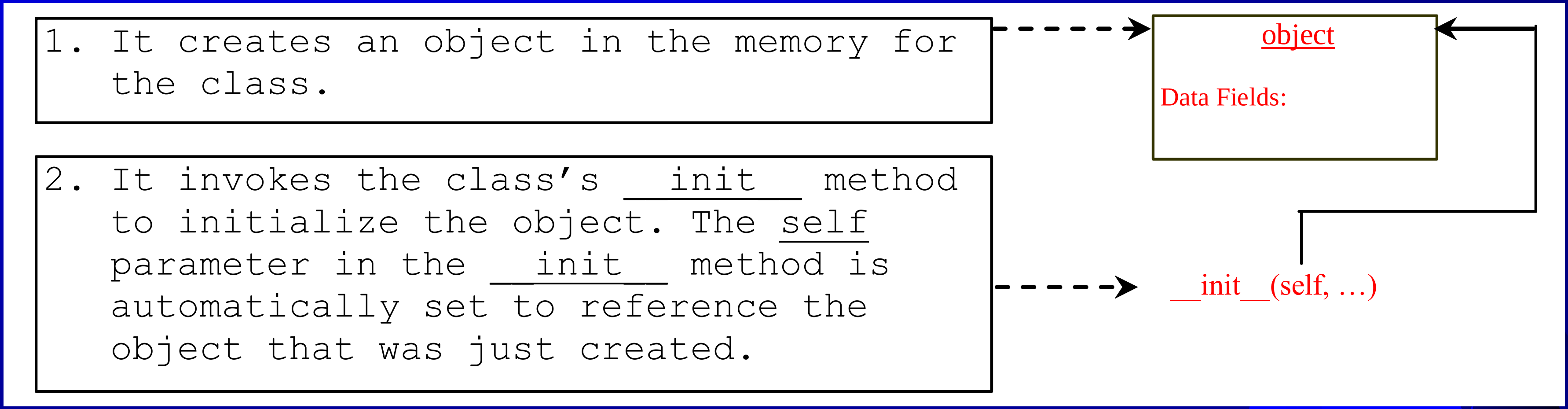
Circle TestCircle Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Constructing Objects

Once a class is defined, you can create objects from the class by using the following syntax, called a *constructor*:

```
className(arguments)
```



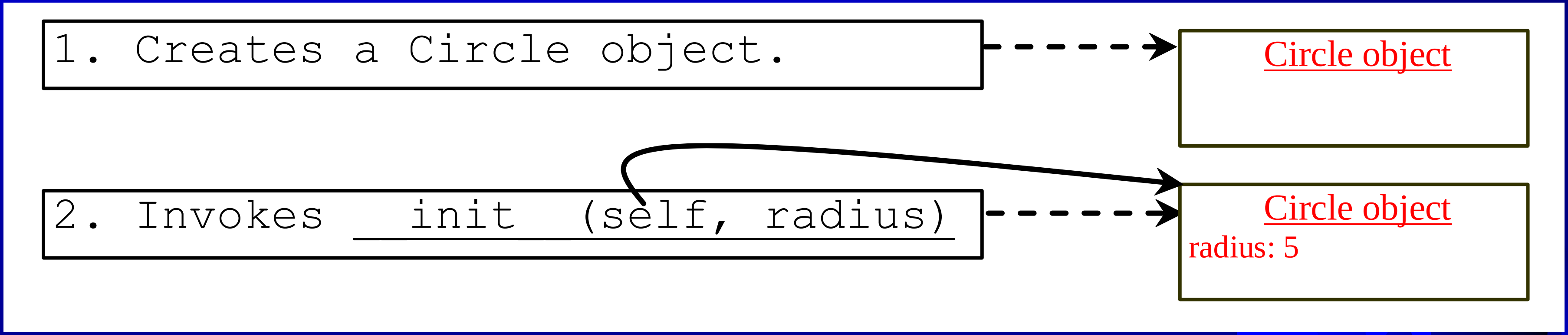
1. It creates an object in the memory for the class.

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.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Constructing Objects

The effect of constructing a Circle object using Circle(5) is shown below:


1. Creates a Circle object.
Circle object
2. Invokes __init__(self, radius)
Circle object
radius: 5

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Instance Methods

- Methods are functions defined inside a class.
- They are invoked by objects to perform actions on the objects.
- For this reason, the methods are also called *instance methods* in Python.
- You probably noticed that all the methods including the constructor have the first parameter **self**, which refers to the object that invokes the method.
- You can use any name for this parameter. But by convention, **self** is used.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Accessing Objects

- After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the *object member access operator*.
- For example, the following code accesses the radius data field and invokes the getPerimeter and getArea methods.

```
>>> from Circle import Circle
>>> c = Circle(5)
>>> c.getPerimeter()
31.41592653589793
>>> c.radius = 10
>>> c.getArea()
314.1592653589793
```

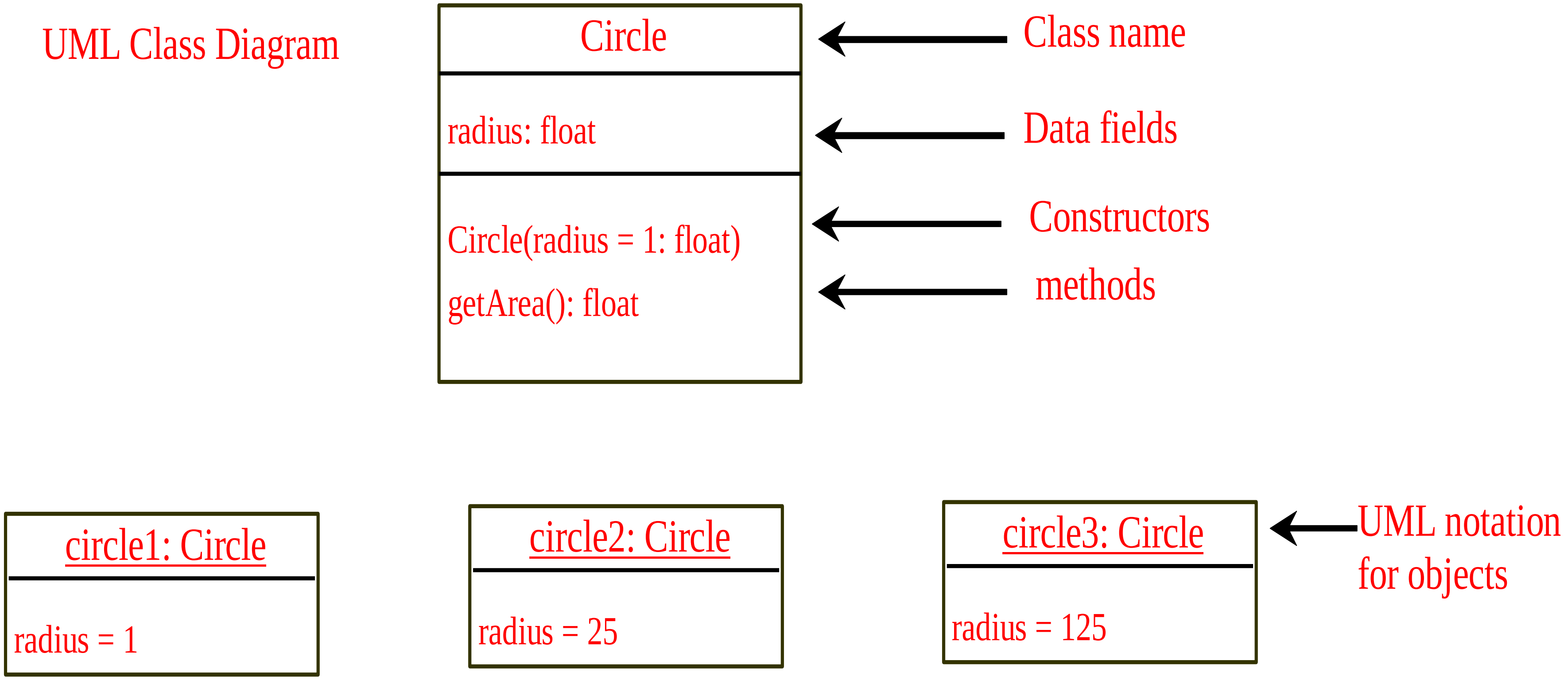
© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Why self?

- Note that the first parameter is special. It is used in the implementation of the method, but not used when the method is called.
- So, what is this parameter self for? Why does Python need it?
- self is a parameter that represents an object. Using self, you can access instance variables in an object. Instance variables are for storing data fields.
- Each object is an instance of a class. Instance variables are tied to specific objects. Each object has its own instance variables.
- You can use the syntax self.x to access the instance variable x for the object self in a method.
- self parameter is required in every method in the class – references the specific object that the method is working on

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

UML Class Diagram



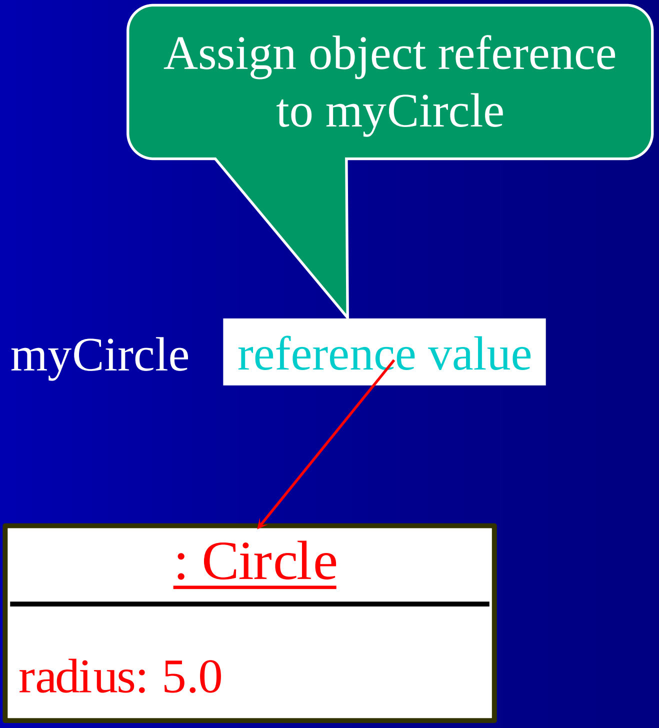
UML : short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specify visualizing, constructing, and documenting the artifacts of software systems, as well as for busi modeling and other non-software systems.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

animation

Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```



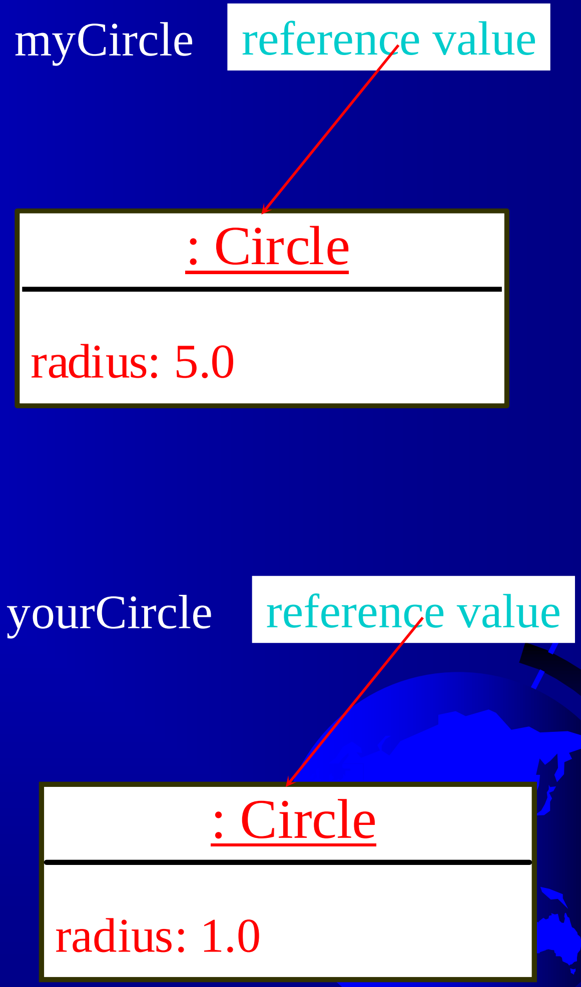
© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

animation

Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```

Assign object reference to yourCircle



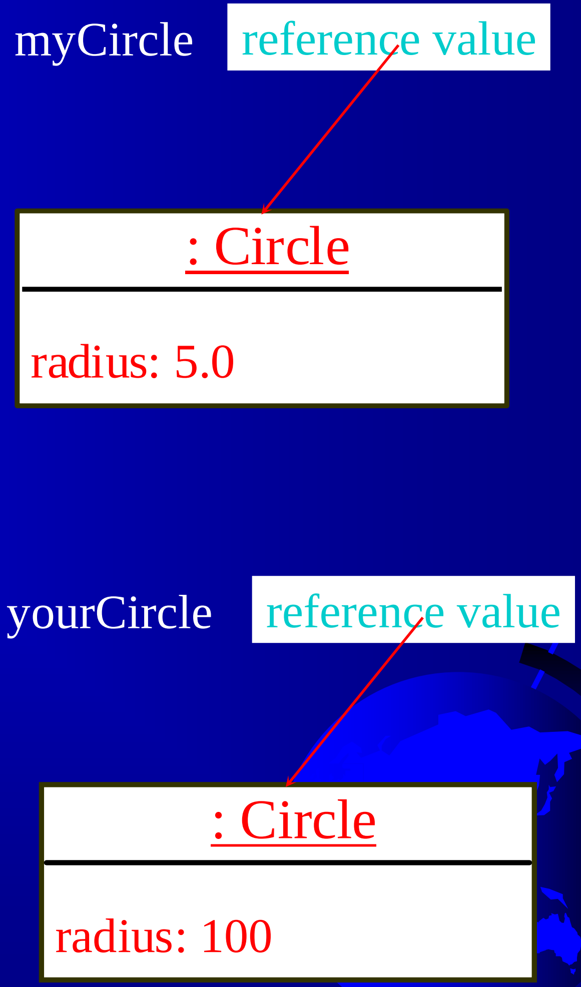
© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

animation

Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```

Modify radius in yourCircle



© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Example: Defining Classes and Creating Objects

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.

TV

TestTV

Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

The built-in datetime Class

```
from datetime import datetime
d = datetime.now()
print("Current year is " + str(d.year))
print("Current month is " + str(d.month))
print("Current day of month is " + str(d.day))
print("Current hour is " + str(d.hour))
print("Current minute is " + str(d.minute))
print("Current second is " + str(d.second))
```

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Encapsulation

- Encapsulation is one of the fundamental concepts in object-oriented programming (OOP).
- It describes the idea of bundling data and methods that work on that data within one unit, e.g., a class.
- This concept is also often used to hide the internal representation, or state, of an object from the outside. This is called information hiding.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Information Hiding

- You can use the encapsulation concept to implement an information-hiding mechanism.
  - To protect data.
  - To make class easy to maintain.
- You implement this information-hiding mechanism by making your class attributes inaccessible from the outside

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Information Hiding

- To prevent direct modifications of data fields, don’t let the client directly access data fields.
- This can be done by defining 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.
- If it is needed, you can provide getter and/or setter methods for attributes that shall be readable or updatable by other classes.

CircleWithPrivateDataRadius

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Data Field Encapsulation

CircleWithPrivateDataRadius

```
>>> from CircleWithPrivateRadius import Circle
>>> c = Circle(5)
>>> c.__radius
AttributeError: 'Circle' object has no attribute
'__radius'
>>> c.getRadius()
 5
```

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Design Guide

- If a class is designed for other programs to use, to prevent data from being tampered with and to make the class easy to maintain, define data fields private (hide).

- If a class is only used internally by your own program, there is no need to encapsulate the data fields.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

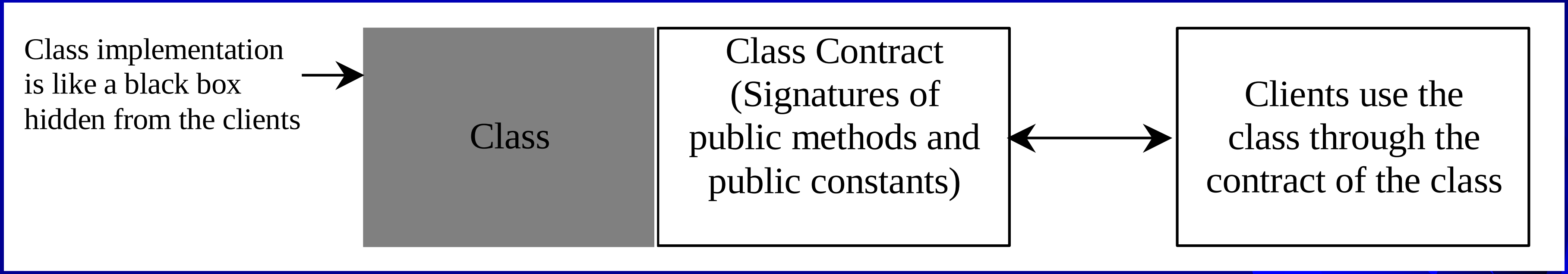
Getter and Setter Methods

- Typically, all of a class's data attributes are private and you have to provide methods to access and change them
- Getter (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
- Setter (Mutator) methods: store or change the value of a data attribute

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Class Abstraction and Encapsulation

- Class abstraction means to separate class implementation from the use of the class.
- The creator of the class provides a description of the class and let the user know how the class can be used.
- The user of the class does not need to know how the class is implemented.
- The detail of implementation is encapsulated and hidden from the user.


Class implementation is like a black box hidden from the clients
Class
Class Contract (Signatures of public methods and public constants)
Clients use the class through the contract of the class

Designing the Loan Class

The – sign denotes a private data field.

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

Loan	
-annualInterestRate: float	The annual interest rate of the loan (default: 2.5).
-numberOfYears: int	The number of years for the loan (default: 1)
-loanAmount: float	The loan amount (default: 1000).
-borrower: str	The borrower of this loan.
Loan(annualInterestRate: float, numberOfYear: int, loanAmount: float, borrower: str)	Constructs a Loan object with the specified annual interest rate, number of years, loan amount, and borrower.

Loan TestLoanClass Run

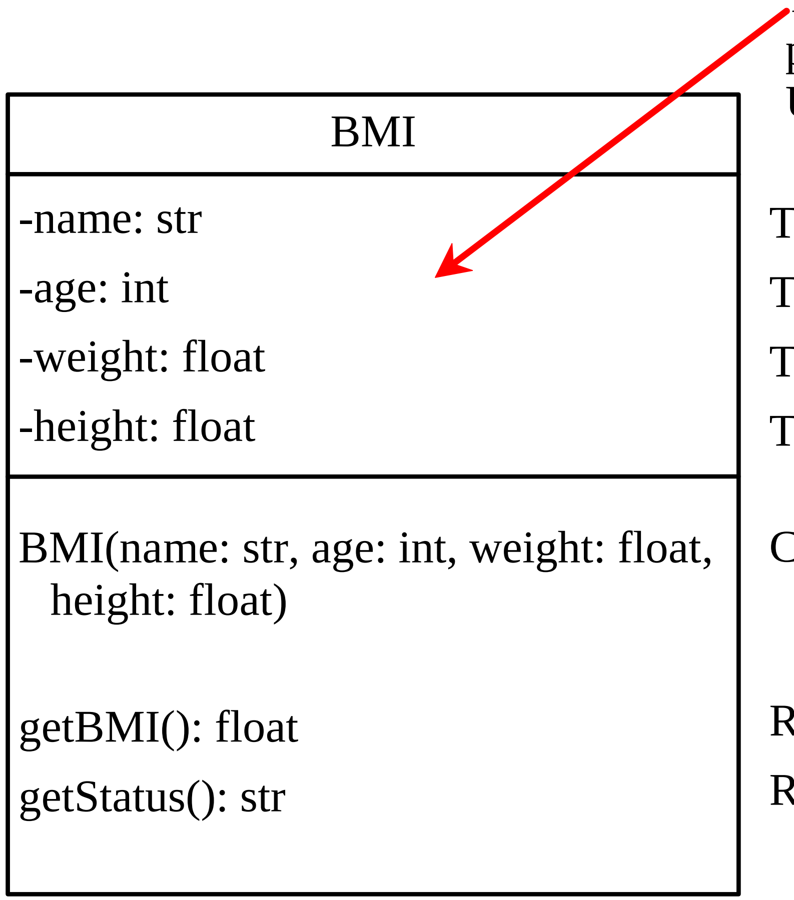
© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Object-Oriented Thinking

- This book's approach is to teach problem solving and fundamental programming techniques before object-oriented programming.
- This section will show how procedural and object-oriented programming differ.
- You will see the benefits of object-oriented programming and learn to use it effectively.
- We will use several examples in the rest of the chapter to illustrate the advantages of the object-oriented approach. The examples involve designing new classes and using them in applications.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

The BMI Class



```
BMI
-name: str
-age: int
-weight: float
-height: float
BMI(name: str, age: int, weight: float,
   height: float)
getBMI(): float
getStatus(): str
```

The get methods for these data fields are provided in the class, but 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, weight, height, and a default age 20.

Returns the BMI

Returns the BMI status (e.g., normal, overweight, etc.)

BMI UseBMIClass Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Object-Oriented Programming

- Object-oriented programming: focused on programming using objects
- Object: entity that contains data and procedures (functions or methods)
  - Data is known as data attributes and procedures are known as methods
    - Methods perform operations on the data attributes
- Encapsulation: combining data and code into a single object

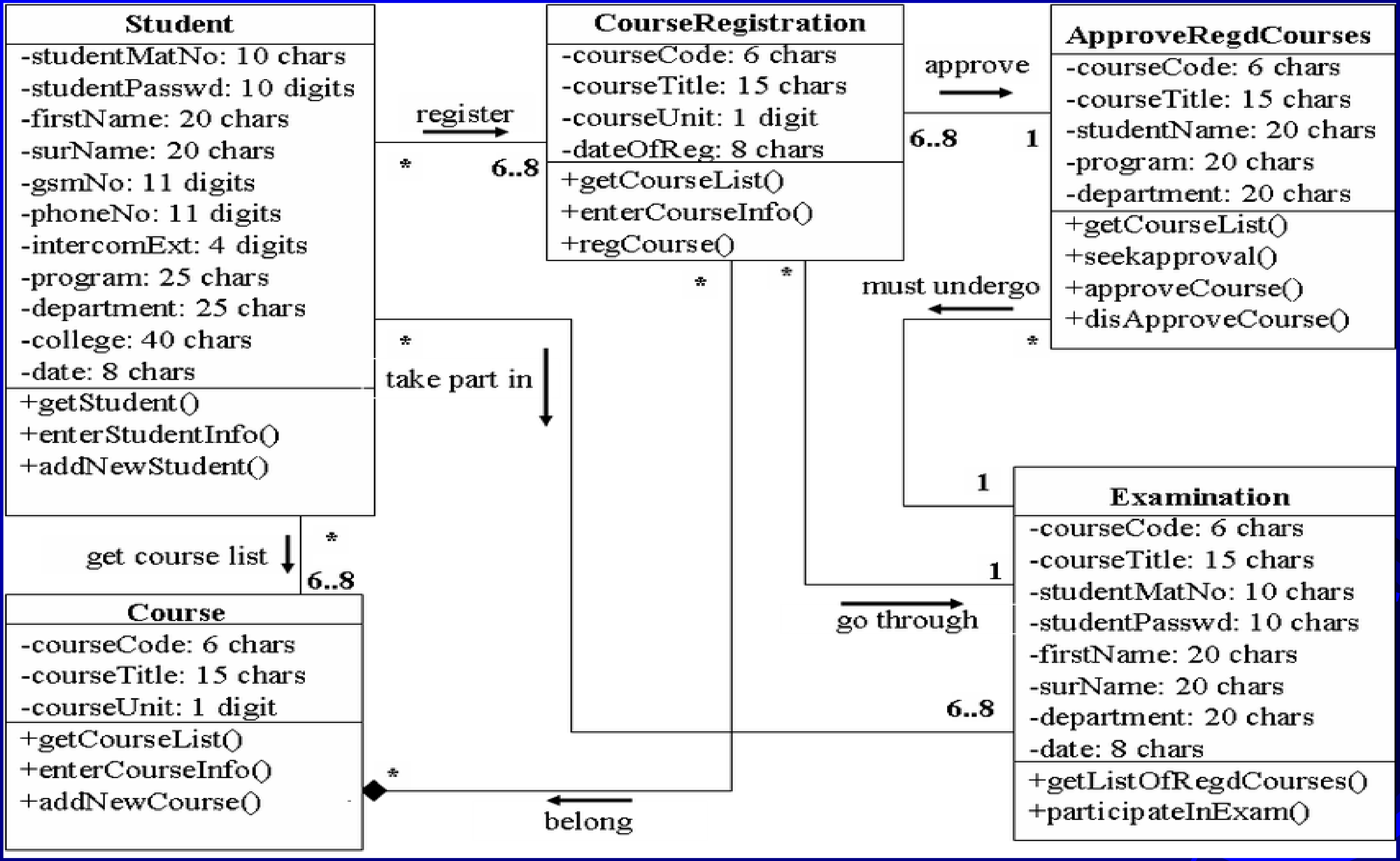
© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Object-Oriented Programming (cont'd.)

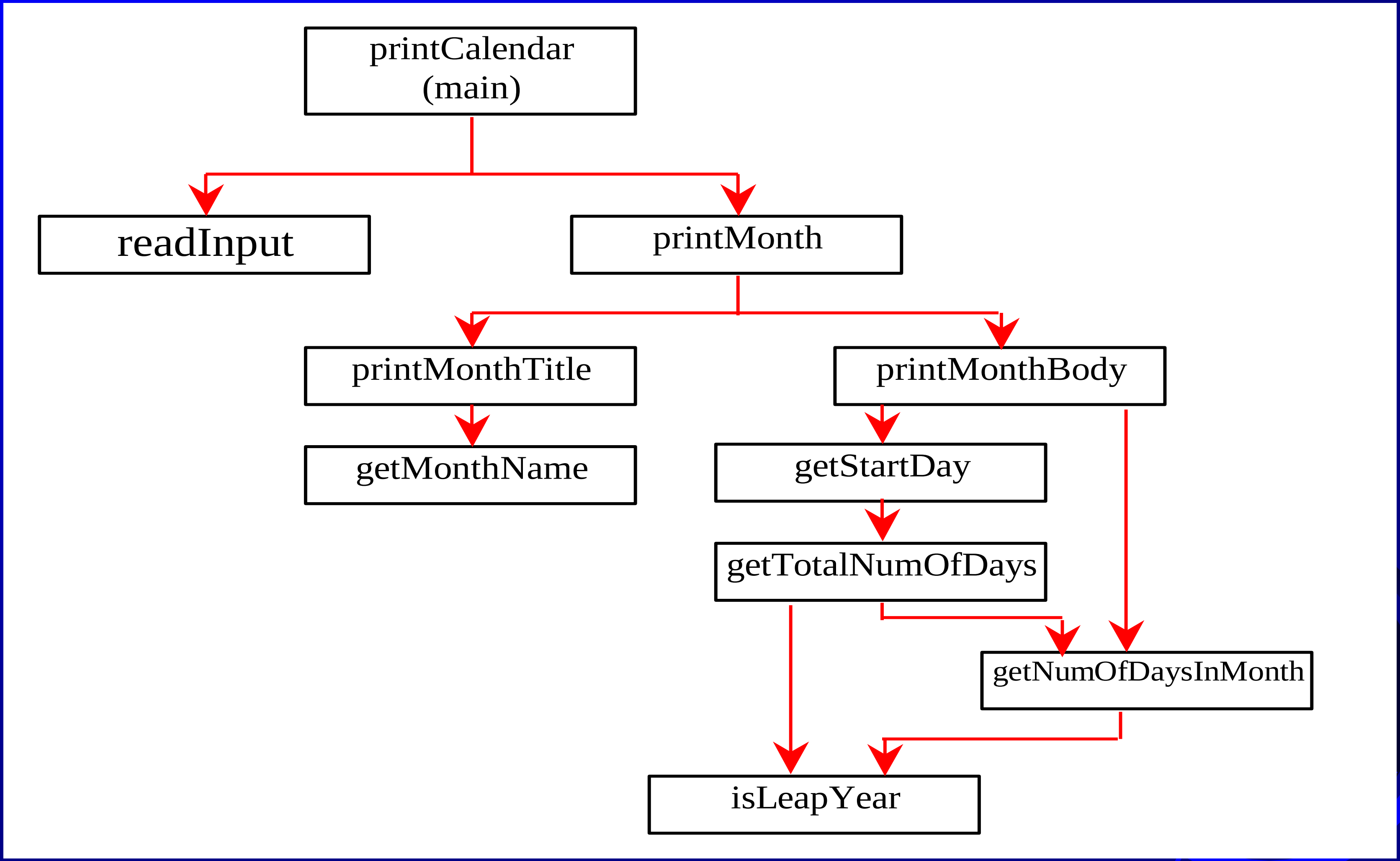
- Data hiding: object's data attributes are hidden from code outside the object
  - Access restricted to the object's methods
    - Protects from accidental corruption
    - Outside code does not need to know internal structure of the object
- Object reusability: the same object can be used in different programs
  - Example: 3D image object can be used for architecture and game programming

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

In object-oriented programming, a program consists of many classes/objects



In procedureal programming, a program consists of many functions



© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Procedural vs. Object-Oriented

- In procedural programming, data and operations on the data are separate, and this methodology requires sending data to methods.
- Object-oriented programming places data and the operations that pertain to them in an object. (Encapsulation)
- This approach solves many of the problems inherent in procedural programming.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

Procedural vs. Object-Oriented

- The object-oriented programming approach organizes programs in a way that mirrors the real world, in which all objects are associated with both attributes and activities.
- Using objects improves software reusability and makes programs easier to develop and easier to maintain.
- Programming in Python involves thinking in terms of objects; a Python program can be viewed as a collection of cooperating objects.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.