Chapter 11: Classes and Object Oriented Programming

Topics

- Procedural and Object-Oriented Programming
- Classes
- Working with Instances
- Techniques for Designing Classes

Addison-Wesley
is an imprint of
PEARSON

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

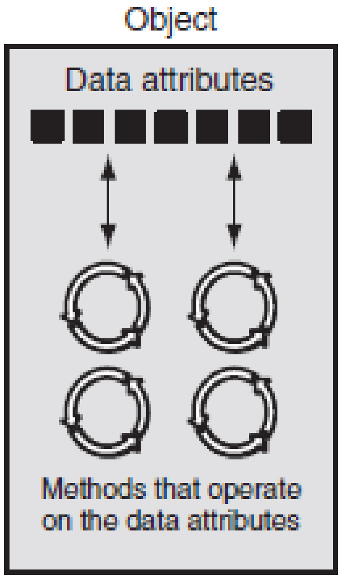
Object-Oriented Programming

- **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
- **Encapsulation: combining data and code into a single object**

Addison-Wesley
is an imprint of
PEARSON

Object-Oriented Programming (cont'd.)

Figure 11-1 An object contains data attributes and methods

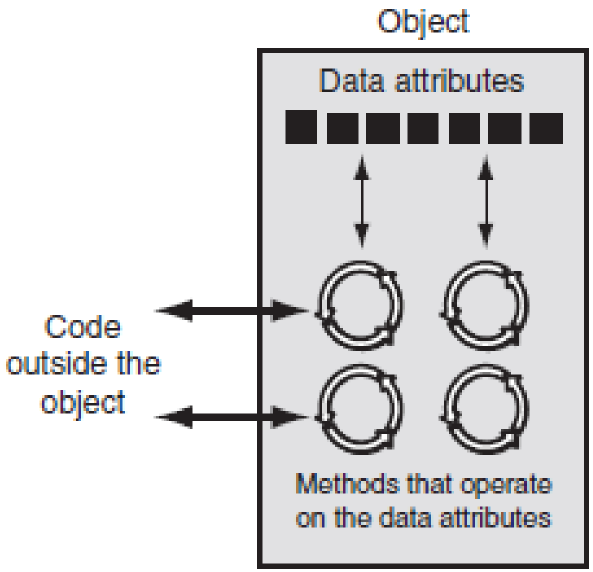


Object-Oriented Programming (cont'd.)

- **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

Object-Oriented Programming (cont’d.)

Figure 11-2 Code outside the object interacts with the object’s methods



Addison-Wesley
is an imprint of
PEARSON

An Everyday Example of an Object

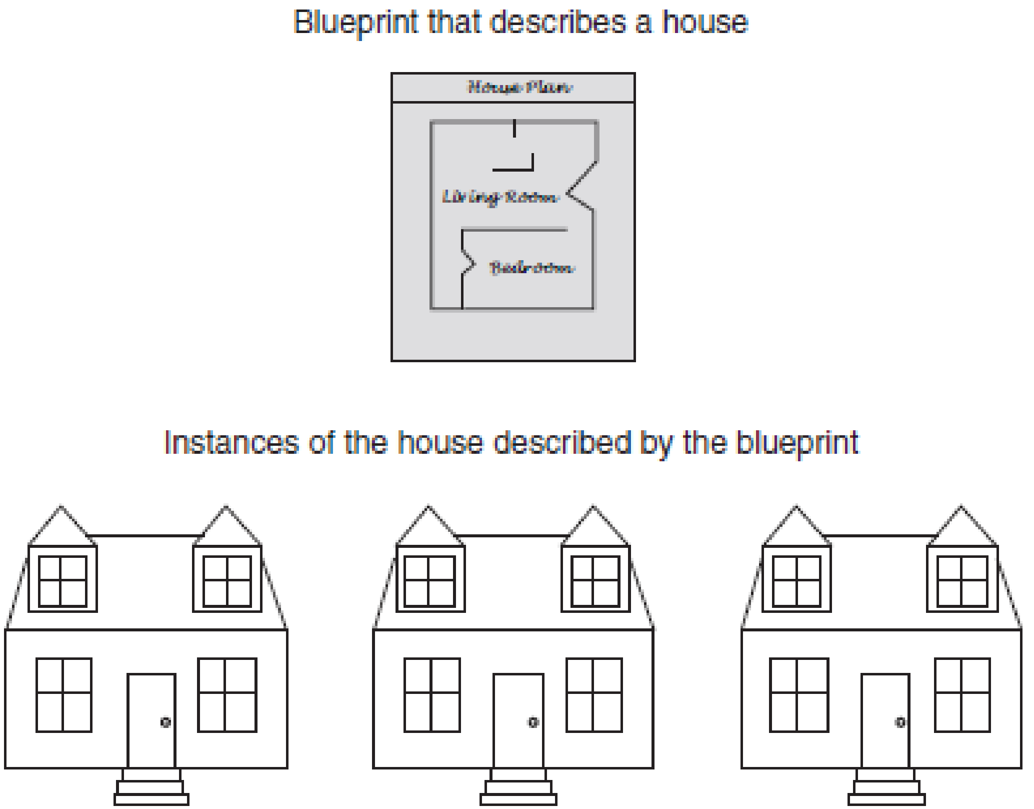
- Data attributes: define the state of an object
  - Example: clock object would have `second`, `minute`, and `hour` data attributes
- Public methods: allow external code to manipulate the object
  - Example: `set_time, set_alarm_time`
- Private methods: used for object's inner workings

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
- **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

Classes (cont'd.)

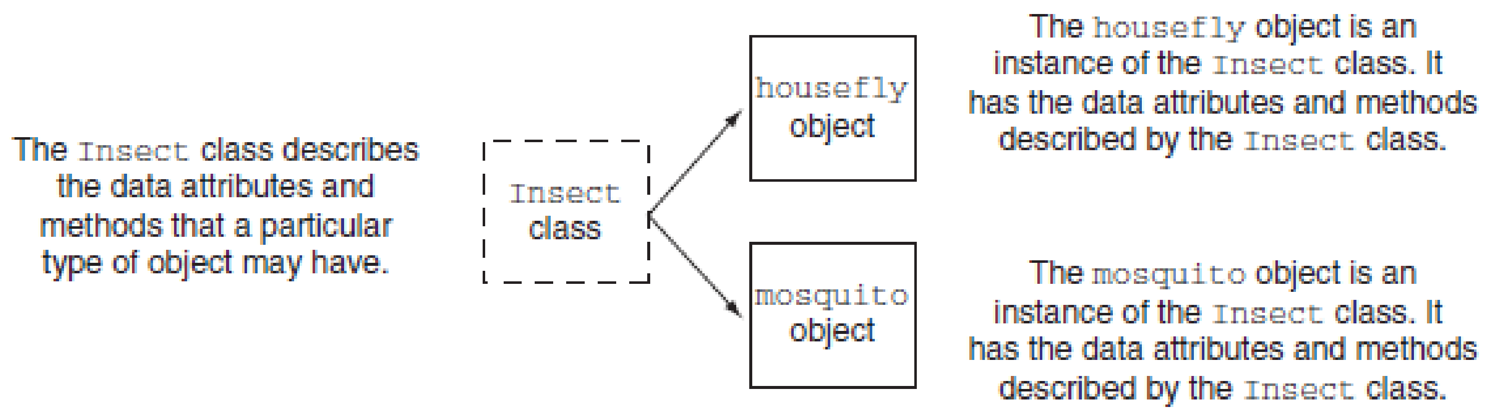
Figure 11-3 A blueprint and houses built from the blueprint


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

Addison-Wesley
is an imprint of
PEARSON

Classes (cont'd.)

Figure 11-5 The housefly and mosquito objects are instances of the Insect 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.

Addison-Wesley is an imprint of

PEARSON

Class Definitions

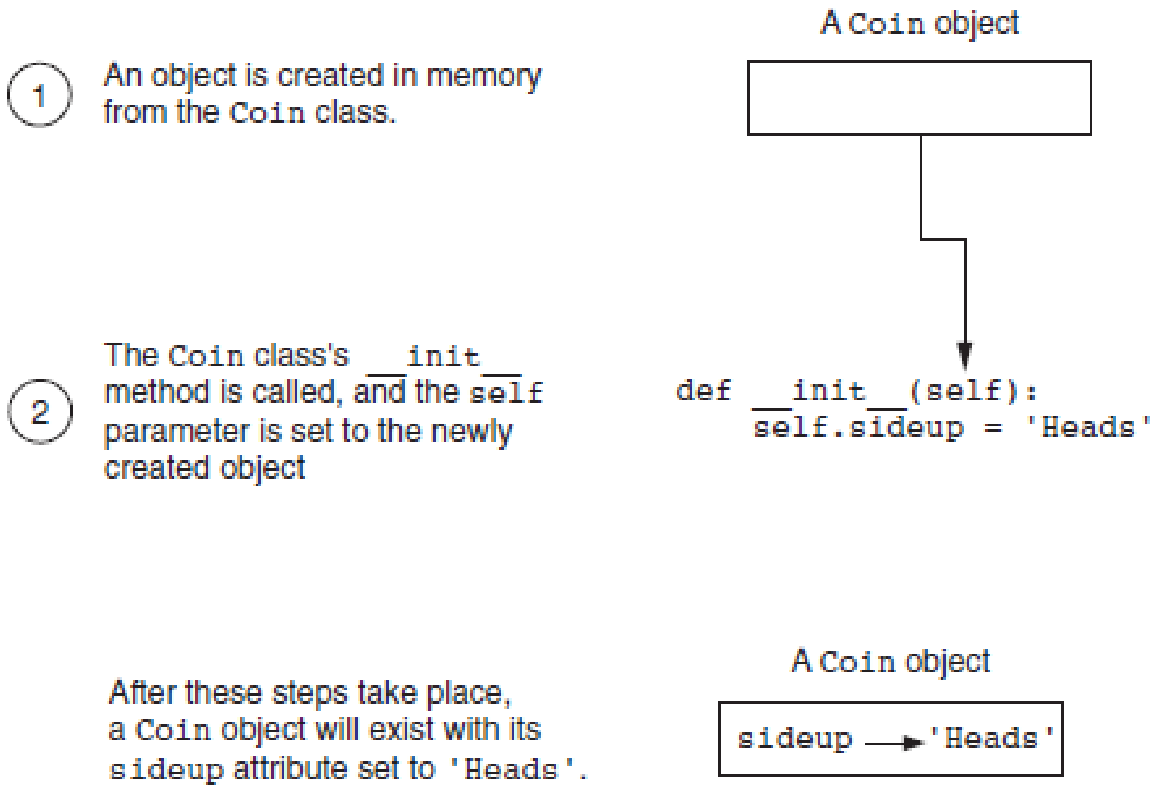
- **Class definition: set of statements that define a class's methods and data attributes**
  - Format: begin with `class` *`Class_name`*`:`
    - Class names often start with uppercase letter
  - Method definition like any other python function definition
    - `self` parameter: required in every method in the class – references the specific object that the method is working on

Class Definitions (cont'd.)

- **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

Class Definitions (cont'd.)

Figure 11-6 Actions caused by the `Coin()` expression



A `Coin` object

1. An object is created in memory from the `Coin` class.

2. 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'`.

A `Coin` object

Class Definitions (cont'd.)

- **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

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`
- **Classes can be stored in modules**
  - Filename for module must end in .py
  - Module can be imported to programs that use the class

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

The `__str__` method

- **<u>Object's state</u>: the values of the object's attribute at a given moment**
- **<u>`__str__ method`</u>: 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

Addison-Wesley
is an imprint of
PEARSON

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**

Addison-Wesley
is an imprint of

PEARSON

Figure 11-8 The coin1, coin2, and coin3 variables reference three Coin objects

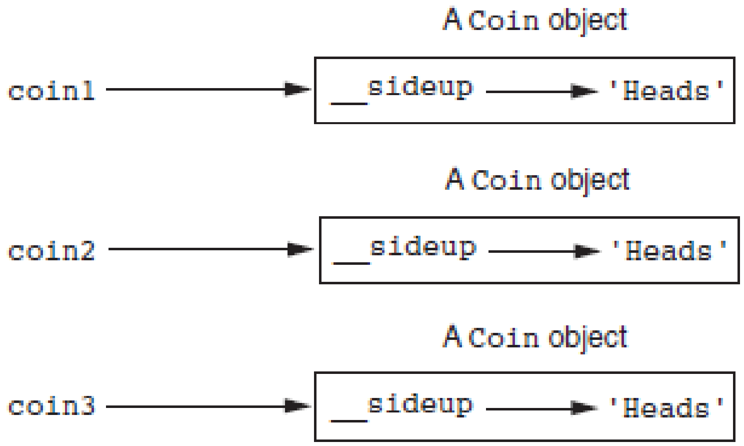
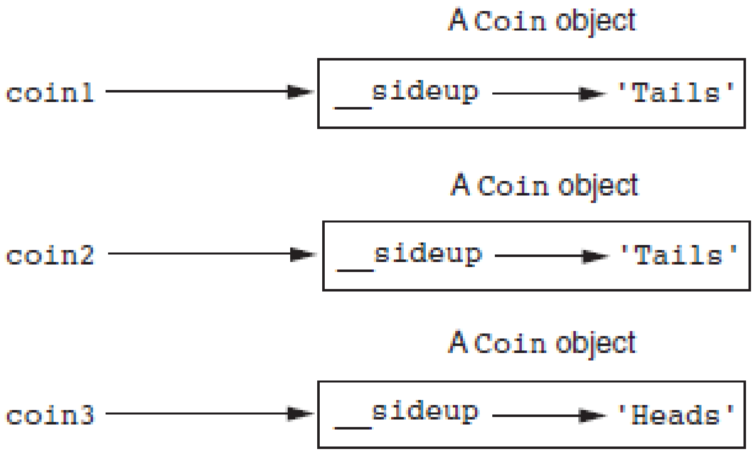


Figure 11-9 The objects after the toss method



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**

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

Addison-Wesley
is an imprint of

PEARSON

Techniques for Designing Classes

- **<u>UML diagram</u>: 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

Addison-Wesley
is an imprint of
PEARSON

Figure 11-10 General layout of a UML diagram for a class

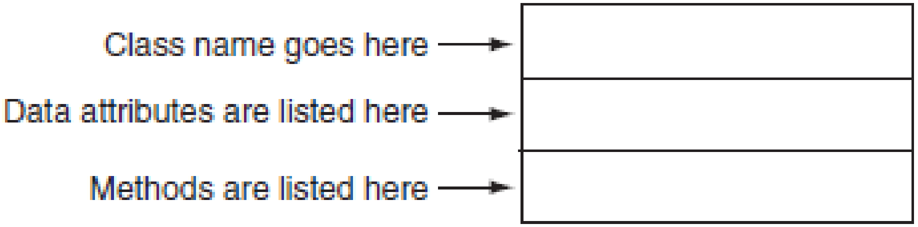
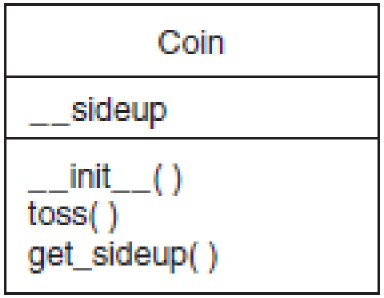


Figure 11-11 UML diagram for the Coin class



Addison-Wesley
is an imprint of

PEARSON

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

Finding the Classes in a Problem (cont'd.)

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

Addison-Wesley
is an imprint of

PEARSON

Finding the Classes in a Problem (cont'd.)

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

Finding the Classes in a Problem (cont'd.)

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

Addison-Wesley is an imprint of PEARSON

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

Summary

- **This chapter covered:**
  - Procedural vs. object-oriented programming
  - Classes and instances
  - Class definitions, including:
    - The `self` parameter
    - Data attributes and methods
    - `__init__` and `__str__` functions
    - Hiding attributes from code outside a class
  - Storing classes in modules
  - Designing classes