Chapter Nine

OBJECTS AND CLASSES

Chapter Goals

- To understand the concepts of classes, objects and encapsulation
- To implement instance variables, methods and constructors
- To be able to design, implement, and test your own classes
- To understand the behavior of object references

In this chapter, you will learn how to discover, specify, and implement your own classes, and how to use them in your programs.

Contents

- Object-Oriented Programming
- Implementing a Simple Class
- Specifying the Public Interface of a Class
- Designing the Data Representation
- Constructors
- Implementing Methods
- Testing a Class
- Problem Solving: Tracing Objects
- Problem Solving: Patterns for Object data
- Object References
- Application: Writing a Fraction Class

Object-Oriented Programming

- You have learned structured programming
  - Breaking tasks into subtasks
  - Writing re-usable methods to handle tasks
- We will now study Objects and Classes
  - To build larger and more complex programs
  - To model objects we use in the world

A class describes objects with the same behavior.
For example, a Car class describes all passenger vehicles that have a certain capacity and shape.

Objects and Programs

- You have learned how to structure your programs by decomposing tasks into functions
  - Experience shows that it does not go far enough
  - It is difficult to understand and update a program that consists of a large collection of functions
- To overcome this problem, computer scientists invented **object-oriented programming**, a programming style in which tasks are solved by collaborating objects
- Each object has its own set of data, together with a set of methods that act upon the data

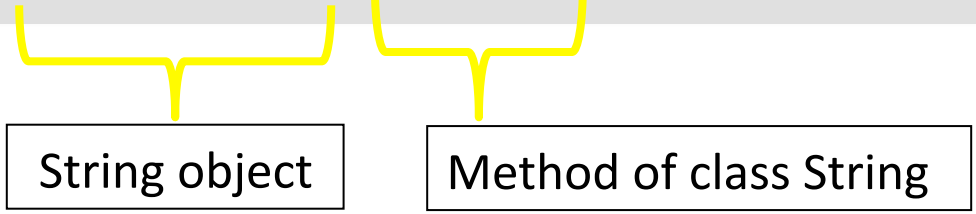
Objects and Programs

- You have already experienced this programming style when you used strings, lists, and file objects. Each of these objects has a set of methods
- For example, you can use the `insert()` or `remove()` methods to operate on list objects

Python Classes

- A class describes a set of objects with the same behavior.
  - For example, the `str` class describes the behavior of all strings
  - This class specifies how a string stores its characters, which methods can be used with strings, and how the methods are implemented.
  - For example, when you have a `str` object, you can invoke the `upper` method:

```
"Hello, World".upper()
```



Python Classes

- In contrast, the `list` class describes the behavior of objects that can be used to store a collection of values
- This class has a different set of methods
- For example, the following call would be illegal—the `list` class has no `upper()` method

```
["Hello", "World"].upper()
```

- However, `list` has a `pop()` method, and the following call is legal

```
["Hello", "World"].pop()
```

Public Interfaces

- The set of all methods provided by a class, together with a description of their behavior, is called the public interface of the class
- When you work with an object of a class, you do not know how the object stores its data, or how the methods are implemented
  - You need not know how a `str` object organizes a character sequence, or how a list stores its elements
- All you need to know is the public interface—which methods you can apply, and what these methods do

Public Interfaces

- The process of providing a public interface, while hiding the implementation details, is called **encapsulation**
- If you work on a program that is being developed over a long period of time, it is common for implementation details to change, usually to make objects more efficient or more capable
  - When the implementation is hidden, the improvements do not affect the programmers who use the objects

Check yourself

1. Is the method call `"Hello, World".print()` legal? Why or why not?

▶Show Answer

---

2. When using a `str` object, you do not know how it stores its characters. How can you access them?

▶Show Answer

---

3. Describe a way in which a `str` object might store its characters.

▶Show Answer

---

4. Suppose the providers of your Python interpreter decide to change the way that a `str` object stores its characters, and they update the `str` method implementations accordingly. Which parts of your code do you need to change when you get the new interpreter?

▶Show Answer

Implementing a Simple Class

- Example:
- Tally Counter: A class that models a mechanical device that is used to count people
  - For example, to find out how many people attend a concert or board a bus
- What should it do?
  - Increment the tally
  - Get the current total

Using the `Counter` Class

- First, we construct an object of the class (object construction will be covered shortly):
- In Python, you don't explicitly declare instance variables
  - Instead, when one first assigns a value to an instance variable, the instance variable is created

```
tally = Counter()   # Creates an instance
```

Using the Counter Class

- Next, we invoke methods on our object

```
tally.reset()
tally.click()
tally.click()
result = tally.getValue()  # Result is 2
tally.click()
result = tally.getValue() # Result is 3
```

Instance Variables

- An object stores its data in **instance variables**
- An *instance* of a class is an object of the class
- In our example, each `Counter` object has a single instance variable named `_value`
  - For example, if `concertCounter` and `boardingCounter` are two objects of the `Counter` class, then each object has its own `_value` variable