HOW TO 9.1

Implementing a Class

A very common task is to implement a class whose objects can carry out a set of specified actions. This How To walks you through the necessary steps.

As an example, consider a class `Menu`. An object of this class can display a menu such as

```
1) Open new account
2) Log into existing account
3) Help
4) Quit
```

Then the menu waits for the user to supply a value. If the user does not supply a valid value, the menu is redisplayed, and the user can try again.

© Mark Evans/iStockphoto.

Step 1 Get an informal list of the responsibilities of your objects.

Be careful that you restrict yourself to features that are actually required in the problem. With real-world items, such as cash registers or bank accounts, there are potentially dozens of features that might be worth implementing. But your job is not to faithfully model the real world. You need to determine only those responsibilities that you need for solving your specific problem.

In the case of the menu, you need to

Display the menu.
Get user input.

Now look for hidden responsibilities that aren't part of the problem description. How do objects get created? Which mundane activities need to happen, such as clearing the cash register at the beginning of each sale?

In the menu example, consider how a menu is produced. The programmer creates an empty menu object and then adds options "Open new account", "Help", and so on. That is another responsibility:

Add an option.

Step 2 Specify the public interface.

Turn the list in Step 1 into a set of methods, with specific parameter variables and return values. Many programmers find this step simpler if they write out method calls that are applied to a sample object, like this:

```
mainMenu = Menu()
mainMenu.addOption("Open new account")
# Add more options
input = mainMenu.getInput()
```

Now we have a specific list of methods.

- `addOption(option)`
- `getInput()`

What about displaying the menu? There is no sense in displaying the menu without also asking the user for input. However, `getInput` may need to display the menu more than once if the user provides a bad input. Thus, `display` is a good candidate for a helper method.

To complete the public interface, you need to specify the constructor. Ask yourself what information you need in order to construct an object of your class. If you need user-supplied values, then the constructor must specify one or more parameter variables.

In the case of the menu example, we can get by with a constructor that requires no arguments.

Here is the public interface:

```
class Menu :
   def __init__(self) :
      . . .
   def addOption(self, option) :
      . . .
   def getInput(self) :
      . . .
```

Step 3 Document the public interface.

Supply a documentation comment for the class, then comment each method.

```
## A menu that is displayed in the terminal window.
#
class Menu :
   ## Constructs a menu with no options.
   #
   def __init__(self) :

   ## Adds an option to the end of this menu.
   #  @param option the option to add
   #
   def addOption(self, option) :

   ## Displays the menu, with options numbered starting with 1, and prompts
   #  the user for input. Repeats until a valid input is supplied.
   #  @return the number that the user supplied
   #
   def getInput(self) :
```

Step 4 Determine instance variables.

Ask yourself what information an object needs to store to do its job. The object needs to be able to process every method using just its instance variables and the method arguments.

Go through each method, perhaps starting with a simple one or an interesting one, and ask yourself what the object needs in order to carry out the method's task. Which data items are required in addition to the method arguments? Make instance variables for those data items.

In our example, let's start with the `addOption` method. We clearly need to store the added menu option so that it can be displayed later as part of the menu. How should we store the options? As a list of strings? As one long string? Both approaches can be made to work. We will use a list here. Exercise P9.3 asks you to implement the other approach.

Now consider the `getInput` method. It shows the stored options and reads an integer. When checking whether the input is valid, we need to know the number of menu items. Because we store them in a list, the number of menu items is simply the size of the list. If you stored the menu items in one long string, you might want to keep another instance variable to store the item count.

Step 5 Implement the constructor.

Implement the constructor of your class, which defines and initializes the instance variables. In this case, `_options` is set to an empty list.

```
def __init__(self) :
   self._options = []
```

Step 6 Implement the methods.

Implement the methods in your class, one at a time, starting with the easiest ones. For example, here is the implementation of the `addOption` method:

```
def addOption(self, option) :
   self._options.append(option)
```

Here is the `getInput` method. This method is a bit more sophisticated. It loops until a valid input has been obtained, displaying the menu options before reading the input:

```
def getInput(self) :
   done = False
   while not done :
      for i in range(len(self._options)) :
         print("%d %s" % (i + 1, self._options[i]))

      userChoice = int(input())
      if userChoice >= 1 and userChoice < len(self._options) :
         done = True

   return userChoice
```

If you find that you have trouble with the implementation of some of your methods, you may need to rethink your choice of instance variables. It is common for a beginner to start out with a set of instance variables that cannot accurately describe the state of an object. Don't hesitate to go back and rethink your implementation strategy.

Step 7 Test your class.

Write a short tester program and execute it. The tester program should call the methods that you found in Step 2.

```
mainMenu = Menu()
mainMenu.addOption("Open new account")
mainMenu.addOption("Log into existing account")
mainMenu.addOption("Help")
mainMenu.addOption("Quit")
choice = mainMenu.getInput()
print("Input:", choice)
```

Program Run

```
1) Open new account
2) Log into existing account
3) Help
4) Quit
5
1) Open new account
2) Log into existing account
3) Help
4) Quit
3
Input: 3
```

The complete `Menu` class and the `menutester` program are provided below.

ch09/how_to_1/menu.py

```
1  ##
2  #  This module defines the Menu class.
3  #
4
5  ## A menu that is displayed in the terminal window.
6  #
7  class Menu :
8     ## Constructs a menu with no options.
9     #
10    def __init__(self) :
11       self._options = []
12
```

```
13    ## Adds an option to the end of this menu.
14    #  @param option the option to add
15    #
16    def addOption(self, option) :
17       self._options.append(option)
18
19    ## Displays the menu, with options numbered starting with 1, and prompts
20    #  the user for input. Repeats until a valid input is supplied.
21    #  @return the number that the user supplied
22    #
23    def getInput(self) :
24       done = False
25       while not done :
26          for i in range(len(self._options)) :
27             print("%d %s" % (i + 1, self._options[i]))
28
29          userChoice = int(input())
30          if userChoice >= 1 and userChoice < len(self._options) :
31             done = True
32
33       return userChoice
```

ch09/how_to_1/menutester.py

```
1  ##
2  #  This program tests the Menu class.
3  #
4
5  from menu import Menu
6
7  mainMenu = Menu()
8  mainMenu.addOption("Open new account")
9  mainMenu.addOption("Log into existing account")
10 mainMenu.addOption("Help")
11 mainMenu.addOption("Quit")
12 choice = mainMenu.getInput()
13 print("Input:", choice)
```

WORKED EXAMPLE 9.1 Implementing a Bank Account Class

Problem Statement Your task is to write a class that simulates a bank account. Customers can deposit and withdraw funds. If sufficient funds are not available for withdrawal, a $10 overdraft penalty is charged. At the end of the month, interest is added to the account. The interest rate can vary every month.

Step 1 Get an informal list of the responsibilities of your objects.

The following responsibilities are mentioned in the problem statement:

Deposit funds.
Withdraw funds.
Add interest.