# Programming exercise set 1: Classes and Objects

## P9.26

Table for the public interface:

|  |  |  |
| --- | --- | --- |
| Task | Method | Data needed |
| Make a purchase, increase accumulated purchase amount | makePurchase(amount) | amount, discountOnNextPurchase |
| Check if customer is eligible for discount on the next purchase | discountReached() | discountOnNextPurchase |

The class is tested within P9\_26.py.

## P9.24

|  |  |  |
| --- | --- | --- |
| Task | Method | Data needed |
| Append a new line to the message body. Call log method. | append (line) | line, messageBody |
| Convert the message into one line of text, with the sender and recipient. | toString() | sender, recipient, messageBody |
| Log a new line of message, updating the class variables. | \_log\_message() | sender, recipient, messageBody, no\_messages, log |

The class is tested within P9\_24.py.

## P9.23

As there is a bit of a discrepancy between the exercise text in “Business P9.23” and the added tasks in the exercise sheet, I have created two classes. Business P9.23 clearly states that the class Country only should contain information about a single country, not multiple countries. It would not make sense to include methods that handle multiple countries in a class meant for a single country, so I made a “CountryCollection” class that can handle multiple countries either as a list of Country objects or as a dictionary with country names as keys and dictionaries as values with “population” and “area” as keys.

The classes are defined in P9\_23\_module.py, where the Country class is also tested. The CountryCollection class is tested in P9\_23\_tester.py.

Doing the steps for the *Country* class:

1. Get an informal list of the responsibilities of your objects.
   1. The Country class needs to have methods that access its:
      1. Name
      2. Area
      3. Population
      4. Population density

This would follow the books programming tip 9.1; “make all instance variables private, most methods public”. Methods to access the instance variables area and population, and a method to calculate and return the population density.

1. Specify the public interface.
   1. To access these values in an elegant way, we can use the in-built @property decorator in Python. Since the method for returning the population density is such a simple method, with the only calculation being dividing one instance variable by another, there is no reason not to use the @property decorator for this method also. If I dropped the use of the @property decorator, I would name the methods differently, specifically with a “get” at the start of the method name. The public interface:

A screen shot of a computer program

Description automatically generated

1. Document the public interface.
   1. The public interface can be documented as such:

A screenshot of a computer program

Description automatically generated

1. Determine the instance variables.
   1. The instance variables should be:
      1. self.\_name
      2. self.\_population
      3. self.\_area

Considering the simplicity of calculating the population density, this could also have been an instance variable. But, in the essence of accessing every instance variable through a method, the population density might as well be calculated by the method.

1. Implement the constructor.
   1. The constructor is, in this case, just initializing the instance variables:

A black background with white text

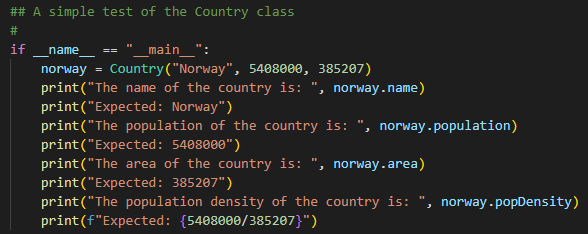
Description automatically generated

1. Implement methods.
   1. The methods for returning the name, area, the population, and the population density:

A screen shot of a computer program

Description automatically generated

1. Test your class.
   1. The Country class is very simple. A very simple test using all the methods is therefore sufficient. This can be done in the module file itself by specifying that it should only be run if the module file itself is executed. This can be achieved by checking if \_\_name\_\_ is equal to “\_\_main\_\_”:



The output in the terminal:

A screenshot of a computer

Description automatically generated

Table for the public interface:

|  |  |  |
| --- | --- | --- |
| Task | Method (as properties) | Data needed |
| Get the country name. | name() | name |
| Get the country area. | area() | area |
| Get the country population. | population() | population |
| Get the country population density. | popDensity() | population, area |

Now for the *CountryCollection* class.

1. Get an informal list of the responsibilities of your objects.
   1. The core responsibilities of the class are:
      1. Find the country with the largest area.
      2. Find the country with the largest population.
      3. Find the country with the largest population density.

We know that we need to have to sets of these methods, one using lists and one using dictionaries. We also need to have methods to add countries to the CountryCollection object. Hence, we have two more methods:

* + 1. Add a country to be used with the list methods.
    2. Add a country to be used with the dictionary methods.

1. Specify the public interface.
   1. Including the constructor, we should now have 9 methods. The public interface is:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. Document the public interface.
   1. Following the documentation style of the book, we can document the public interface like:

A computer screen shot of a country list

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. Determine the instance variables.
   1. For the CountryCollection class we need two instance variables. One list and one dictionary, to be used with their respective methods:
      1. \_country\_list
      2. \_country\_dict
2. Implement the constructor.
   1. The instance variables can simply be initialized as an empty list and dictionary in the constructor:

A computer code with text

Description automatically generated

1. Implement methods.
   1. First off, we have the methods to add countries to the CountryCollection object. For the list method, the best implementation is to use the Country class created earlier such that \_country\_list becomes a list of Country objects. For the dictionary methods, it doesn’t really make sense to use the Country class. Instead, I implement the country names as keys in the \_country\_dict dictionary with the values being dictionaries with “population” and “area” as keys.

A computer screen shot of a code

Description automatically generated

Next, we have the three list methods to answer the three questions in the task, largest area, population, and population density. These all work the same way. First, they make sure the list is not empty, otherwise the methods will print a warning and return False (exit the method). Second, they utilize the in-built max() function in Python to find the largest of their respective values. By specifying a lambda function in the “key” argument of the function, we can maximize over all the areas, populations and population densities while returning the entire Country object corresponding to the largest value. The methods then return the country name.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Next, we have three dictionary methods. They work similarly to the list methods but uses the .items() dictionary method to convert the dictionary to a list of key-value pair tuples. This allows us to use the max() function again. To explain the lambda function, for every tuple “i” in the list, we access first the value (of the key-value pair) by specifying the index 1 (aka item two since Python starts indexing at 0), and then specifying which value from the dictionary we want to maximize over (i.e., the population, area or the population divided by the area – the population density). Then the methods return just the country name, which is the first element of the tuple.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. Test your class.
   1. Since this class is more comprehensive, it can be useful to have a dedicated test file. It can also be useful to use the argparse library to both document the public interface of the class and to test it.

A computer screen shot of a program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

To test the class we can write, e.g. (in a windows terminal):

1. py P9\_23\_tester.py --add\_dict USA 2300000 567899 --add\_dict Canada 450000000 7890000 --add\_dict Norway 5000000 34599000 --dict\_largest\_density
   1. Response:



Table for the public interface:

|  |  |  |
| --- | --- | --- |
| Task | Method | Data needed |
| Add country to the country list. | addCountryToList (name, population, area) | name, population, area, country\_list |
| Add country to the country dictionary. | addCountryToDict (name, population, area) | name, population, area, country\_dict |
| Get the country with the largest area using lists. | list\_largest\_area() | country\_list |
| Get the country with the largest population using lists. | list\_largest\_population() | country\_list |
| Get the country with the largest population density using lists. | list\_largest\_pop\_density() | country\_list |
| Get the country with the largest area using dictionaries. | dict\_largest\_area() | country\_dict |
| Get the country with the largest population using dictionaries. | dict\_largest\_population() | country\_dict |
| Get the country with the largest population density using dictionaries. | dict\_largest\_pop\_density() | country\_dict |