Objects And Classes - Aggregation and Composition

In this notebook we cover another aspect of creating classes. Oftentimes, a class can include an instance variable which is itself the object of another class. This type of relationship between the two classes can be one of two kinds:

Composition: If the inner object is such that it cannot exist without the outer object, we call it `composition`.

Aggregation: If the inner object can exist outside the outer object we call it `aggregation`

In this notebook we will consider a simple example to demonstrate both composition and aggregation.

Consider an Employee class which includes, in addition to other instance variables, an instance variable which is an object of the Department class. Henceforth we will refer to the Department object as an object of the inner class while the Employee object will be referred to an object of the outer class.

We will first write the class definition of the inner class object. Note that this class definition is a simple one and will be written like we did the previous example. We will then write two versions of the Employee class. In one we use composition to include the Department object and in the other we will use aggregation. Finally, we will write a simple function to test out the `Employee` class by creating one `Employee` object and printing it out.

We will begin by writing the class definition for the `Department`.

1. The class has two instance variables, department name and number of employees.
2. Use name mangling for all instance variables.
3. When the object is created, your code should check to make sure that the number of employees is strictly positive. If not, print an appropriate message.
4. In addition to the constructor, your class definition should also have the following methods:
   A. Accessor methods for all instance variables
   B. Mutator methods for all instance variables

In [1]:

```
import sys
class Department:
    def __init__(self, n, num):
        if num <= 0: #This is an example of input validation inside the __init__() method
            print('Invalid value for number of employees')
            sys.exit(0)
        self.__num_emp = num
        self.__dept_name = n

    '''
        Accessor methods
    '''
    def get_dept_name(self):
        return self.__dept_name

    def get_num_emp(self):
        return self.__num_emp

    '''
        Mutator methods
    '''
    def set_dept_name(self, n):
        self.__dept_name = n

    def set_num_emp(self, num):
        self.__num_emp = num

    def __str__(self):
        return  'Department Name: ' + self.__dept_name + ', Number of employees: ' + str(self.__num
_emp)
```

Next we will write two different class definitions for the `Employee` class: one using composition for the `Department` instance variable and the other using Aggregation for the `Department` instance variable. Except for this difference, the two class definitions are identical and described below

The class has four instance variables, employee name, job title, department and salary. Note that the department variable is an object of the `Department` class. Use name mangling for all instance variables. When the object is created, your code should check to make sure that the salary is strictly positive. If not, print an appropriate message.

In addition to the constructor, your class definition should also have the following methods:

1. Accessor methods for all instance variables
2. Mutator methods for all instance variables
3. A redefined __str__() method to print the name, title, department name and salary of each employee object.

In [2]:

```
'''
 In this cell we write the class definition for the Employee class using Composition to define the relationship between
 the Employee and Department classes.
'''
import sys

class EmployeeComposition:

    '''
        Note that since we are using composition, the Department object is created inside the
        __init__ method of the employee object.  We therefore pass the arguments corresponding to the instance variables
        of the Department object.
    '''
    def __init__(self, en, jt, es, dn, dnum):
        if es <= 0:
            print('Invalid value for salary')
            sys.exit(0)
        self.__emp_name = en
        self.__job_title = jt
        self.__emp_sal = es

        '''
            We create the department object inside the __init__ method of the Employee class.  This means that this
            Department object cannot be accessed without the Employee object.
        '''
        self.__dept = Department(dn, dnum)

    '''
        Accessor Methods
    '''
    def get_emp_name(self):
        return self.__emp_name

    def get_job_title(self):
        return self.__job_title

    def get_emp_sal(self):
        return self.__emp_sal

    def get_dept(self):
        return self.__dept

    '''
        Mutator Methods
    '''
    def set_emp_name(self, en):
        self.__emp_name = en

    def set_job_title(self, jt):
        self.__job_title = jt

    def set_emp_sal(self, es):
        self.__emp_sal = es

    def set_dept(self, d):
        self.__dept = d

    def __str__(self):
        emp = 'Emp Name: {}, Job Title: {}, Salary: ${:0.2f}'.format(self.__emp_name, self.__job_title, self.__emp_sal)
```

```
        emp += ', Department Name: {}'.format(self.__dept.get_dept_name())
        emp += ', Number of Employees in Department: {}'.format(self.__dept.get_num_emp())
        return emp
```

In [3]:

```
'''
 In this cell we write the class definition for the Employee class using Aggregation to define the
relationship
 between the Employee and Department classes.
'''
class EmployeeAggregation:

    '''
        Note that since we are using aggregation, the department object was created outside
        the __init__ method of the employee object.  We then pass the department object as an argu
ment
        to the __init__method.
    '''
    def __init__(self,en, jt, es, d):
        if es <= 0:
            print('Invalid value for salary')
            sys.exit(0)
        self.__emp_name = en
        self.__job_title = jt
        self.__emp_sal = es

        self.__dept = d # We directly asssign the department argument to the department instance va
riable.


    '''
        Accessor Methods
    '''
    def get_emp_name(self):
        return self.__emp_name

    def get_job_title(self):
        return self.__job_title

    def get_emp_sal(self):
        return self.__emp_sal

    def get_dept(self):
        return self.__dept

    '''
        Mutator Methods
    '''
    def set_emp_name(self, en):
        self.__emp_name = en

    def set_job_title(self, jt):
        self.__job_title = jt

    def set_emp_sal(self, es):
        self.__emp_sal = es

    def set_dept(self, d):
        self.__dept = d

    def __str__(self):
        emp = 'Emp Name: {}, Job Title: {}, Salary: ${:0.2f}'.format(self.__emp_name, self.__job_ti
tle, self.__emp_sal)
        emp += ', ' + self.__dept.__str__()
        return emp
```

We will then define a driver program to test out our two different `Employee` classes. The only difference in how we create objects of the two different classes is the parameters that are passed each time we create the object.

In [4]:

```
def main():
```

```
    '''
        Since the department object will be created inside the employee object, we just call the e
mployee constructor
        with the instance variables required for the department object
    '''
    emp1 = EmployeeComposition('Jane Doe', 'Senior Product Manager', 98520, 'Marketing', 54)
    print(emp1)

    '''
        The department object is first created before being passed as an argument to the employee
constructor
    '''
    dept2 = Department('Marketing', 54)
    emp2 = EmployeeAggregation('Jane Doe', 'Senior Product Manager', 98520, dept2)
    print(emp2)
main() # call the main function
```

```
Emp Name: Jane Doe, Job Title: Senior Product Manager, Salary: $98520.00, Department Name:
Marketing, Number of Employees in Department: 54
Emp Name: Jane Doe, Job Title: Senior Product Manager, Salary: $98520.00, Department Name:
Marketing, Number of employees: 54
```