Problem Definition:

Develop a program that defines a class named ***Employee*** with 4 attributes to archive 4 pieces of information about an employee. The pieces of information will be: ***name***, ***IDnumber***, ***department***, and ***jobTitle***. The class should be defined with 8 methods: one method for each attribute to assign it a value through a parameter of that method, and one method for each attribute to return the value assigned to the attribute to the statement calling the method.

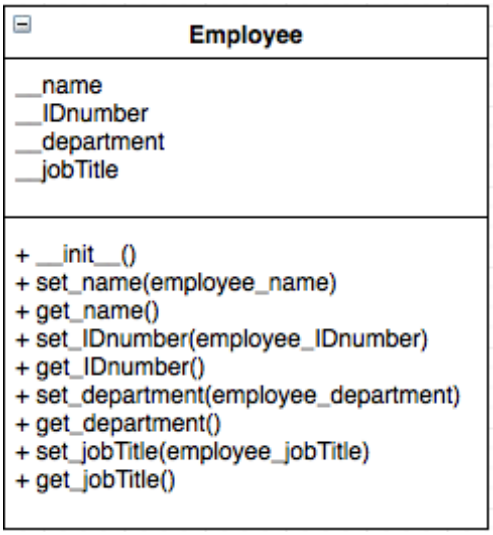
The program should have a ***main*** function that creates three objects with the ***Employee*** class. The main function should collect the name of the business, then prompt the user to enter the information of an employee and assign that input data to its respective attribute within the first object created with the ***Employee*** class. This input and assigning process should be repeated for the other two remaining objects. After the data is collected and stored, the ***main*** function should then call a function to display the output information formatted into the fashion of a table. The output should be well spaced into columns with a header for each piece of information displayed, and the information for each employee will be in it's own row.

EG:

```
                NAME OF BUSINESS EMPLOYEE REPORT
EMPLOYEE NAME         ID NUMBER        DEPARTMENT       TITLE
_____________         _________-       __________       ______
John Smith            123              Sales            Manager
Jane Doe              456              Billing          Clerk
Joe John              789              Sales            Associate
```

Analysis:

To develop a program to meed the requirements specified in the problem definition I plan to define a class titled ***Employee*** within it's own module titled '***employee.py***'. The ***Employee*** class will contain 4 attributes named: ***__name***, ***__IDnumber***, ***__department***, and ***__jobTitle***. Each attribute will have a '**__**' prefixed to its name to preserve the integrity of the value it will contain. This will ensure that only a method defined within this class will be allowed to manipulate any attribute's held value. The class will have 8 methods, one method for each attribute that will assign a value to its associated attribute passed through a parameter, and one method for each attribute that will return the current value of its assigned attribute to the statement calling it. The names of the described attributes and methods are listed in the UML chart displayed.



Since input is being collected and no calculations are to be performed all variables used within this program and class module will be set to the string data type. All variables will be named to appropriately describe the value that will be contained within them. The first variable that appears is ***employee1***. This variable holds, and in a sense, becomes the an object with the properties defined in the ***Employee*** class. The variables that follow are ***employee2*** and ***employee3***. They are also objects of the ***Employee*** class, all of which contain the information collected from the user. ***employee1***, ***employee2***, and ***employee3*** are all output variables as well. The ***displayEmployees*** function will call their ***get_xxxx***

methods to display the contained information back to the user. The first of the input variables to be defined and assigned is the ***business_name*** variable. This variable will hold a value to represent the name of the user's business and is to be collected from the user. The ***business_name*** variable is the only input variable that will be used to display output information back to the user. All of the remaining input variables will be assigned data three separate times in total during the ***main*** function. They will be recycled to collect information from the user to be stored within each of the objects derived from the ***Employee*** class. They will each be passed through a parameter to the corresponding attribute in each of the objects during the ***main*** function. The variable ***employee_name*** will be assigned the value of the employees' names. The variable named ***employee_IDnumber*** will represent the value containing the employees' ID numbers to be assigned and passed. ***employee_department*** will hold the value of all of the employees' departments. Finally, ***employee_jobTitle*** will hold the value of the employee's title at the company. There will be no defined constants in this program.

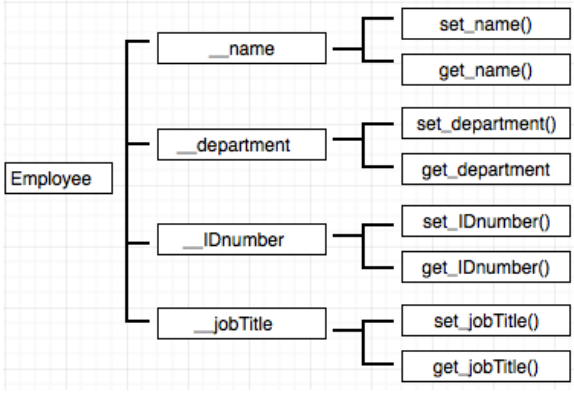
The methods of the Employee class objects will use variables in their parameters to receive the values for their attributes. In this program each of those parameters will have the same name as their input variable counterparts; thusly named: ***employee_name***, ***employee_IDnumber***, ***employee_department***, and ***employee_jobTitle***.

The displayEmployees function will receive 4 arguments to process the output statements with. Each of the Employee class objects will be passed through arguments with the titles: ***employee_ob1***, ***employee_obj2***, and ***employee_obj3***. The numbers of each object will of course correspond with the numbers of the arguments. The ***displayEmployees*** function is to also take an argument for the ***business_name*** input and output variable, and will aptly be named ***business***.

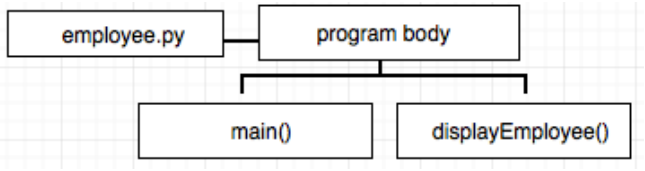
There will be a handful of string literals in the displayEmployees function. These literal values will make up the heading of the table displayed to the user. The literal values will be as follows: "EMPLOYEE NAME", "ID NUMBER", "DEPARTMENT", and "TITLE". There will also be a row of underscores printed below the heading's column titles. This will simply separate the heading from the body of the table to increase readability of the output information. The underscores will be printed as literal strings, the same as the heading titles.

Design:

The structure of my program will be fairly straight forward. I will define the ***Employee*** class in a separate module of it's own titled ***employee.py***. Due to the simplicity of this practice and the benefits that it adds to recycling and longevity to programming in general, I feel it will be a good practice going forward in development. As for the design of the class itself, there will be several methods in its hierarchy. The way that plan to arrange the source code will group the 'set_x_attribute' and 'get_x_attribute' methods together in a pair as they relate to each attribute, rather than arranging the 'set_x_attribute' methods together and the 'get_x_attributes' together. They will be coupled in the order that the attributes themselves are to be initialized. I am choosing this layout for the statements of the class because I view the methods of the class as the vessels of interaction between the program itself and the attributes of the object. With this layout, I suppose the hierarchy of the class will form with each attribute forming a branch of hierarchy where its dedicated methods will branch out after it. The diagram to the right displays that logic.



As for the structure of the program itself, after the module containing the ***Employee*** class is imported two functions will be used. The ***main*** function will be called within the program's body itself, and the ***displayEmployee*** function will be called within the ***main*** function. Due to the simplicity of this structure, the hierarchy of this program will be rather standard, and its structure linear. There will not be any loops or decisions to be made, simple input and output. The hierarchy of the functions and the class module in relation to the program's body is diagramed below.



The logic of the program as a whole is detailed below in a for of pseudocode.

employee.py

```
Start
        define class Employee
                initialization of attributes
                        string __name
                        string __department
                        string __IDnumber
                        string __jobTitle

                define method set_name(employee_name)
                        set __name = employee_name
                define method get_name()
                        return value of __name to calling statement

                define method set_department(employee_department)
                        set __department = employee_department
                define method get_department()
                        return value of _department to calling statement

                define method set_IDnumber(employee_IDnumber)
                        set __IDnumber = employee_IDnumber
                define method get_IDnumber()
                        return value of __IDnumber to calling statement

                define method set_jobTitle(employee_jobTitle)
                        set __jobTitle = employee_jobTitle
                define method get_jobTitle()
                        return value of __jobTitle to calling statement
End
```

CS115_Project2.py

Start

Declarations

str business_name
str employee_name
str employee_IDnumber
str employtt_department
str employee_jobTitle
import module employee.py

define main function

Declarations

employee1 = Employee class
employee2 = Employee class
employee3 = Employee class

input business_name "What's the name of the business? "
input employee_name "What's the 1st employee's name? "
input employee_IDnumber "What's the 1st employee's ID number? "
input employee_department "What's the 1st employee's department? "
input employee_jobTitle "What's the 1st employee's job title? "

call employee1 method set_name(employee_name)
call employee1 method set_IDnumber(employee_IDnumber)
call employee1 method set_department(employee_department)
call employee1 method setjobTitle(employee_jobTitle)

input employee_name "What's the 2nd employee's name? "
input employee_IDnumber "What's the 2nd employee's ID number? "
input employee_department "What's the 2nd employee's department? "
input employee_jobTitle "What's the 2nd employee's job title? "

call employee2 method set_name(employee_name)
call employee2 method set_IDnumber(employee_IDnumber)
call employee2 method set_department(employee_department)
call employee2 method setjobTitle(employee_jobTitle)

input employee_name "What's the 3rd employee's name? "
input employee_IDnumber "What's the 3rd employee's ID number? "
input employee_department "What's the 3rd employee's department? "
input employee_jobTitle "What's the 3rd employee's job title? "

call employee3 method set_name(employee_name)
call employee3 method set_IDnumber(employee_IDnumber)
call employee3 method set_department(employee_department)
call employee3 method setjobTitle(employee_jobTitle)

call displayEmployees(employee1, employee2, employee3, business)

```
        define displayEmployees function take parameters (employee_obj1, employee_obj2,
…       employee_obj3, business)
                output " "
                output business, "EMPLOYEE REPORT" -center-align format: table header
                output "EMPLOYEE NAME" tab "IDENTIFIER" tab "DEPARTMENT" tab "TITLE"
                output "_____" tab "_____" tab "_____" tab "_____"

                output employee_obj1 get_name tab \
                        employee_obj1 get_IDnumber tab \
                        employee_obj1 get_department tab \
                        employee_obj1 get_jobTitle

                output employee_obj2 get_name tab \
                        employee_obj2 get_IDnumber tab \
                        employee_obj2 get_department tab \
                        employee_obj2 get_jobTitle

                output employee_obj3  get_name tab \
                        employee_obj3 get_IDnumber tab \
                        employee_obj3 get_department tab \
                        employee_obj3 get_jobTitle

        call main function
End
```

Implementation:

The computer I used to develop this program uses macOS Sierra, version 10.12.1. To write the source code I used the programming environment Pycharm CE 2016.3.1 as well as it's integrated compiler.
I tested this program extensively with differing values of input. Since all variables were strings the only errors I encountered were if the characters I entered exceeded the space allotment used in the last argument of the format function. This is similar to the issues I encountered during the testing of the previous programming project. After entering several values with what seemed to be reasonable or average character length submissions, I settled on the character length values appearing in the source code of my submission.

The first set of input data entered is as follows:
business name: BILL & TED's EXCELLENT SANDWICH SHOP
employee 1:
Bill Preston, 123, Administration, Co-manager
employee 2:

Ted Logan, 456, Administration, Co-manager
employee 3:
Rufus, 789, R & D, Head advisor
This input returned no errors in syntax or formatting

The second set of input data I entered is as follows:
business name:
EVERETT’s TREASURE & EXPEDITION CO.
employee 1:
Ulysses Everett McGill, 12345, Creativity Department, Ring-leader
employee 2:
Delmar O’Donnell, 67890, Human Resources, Moral Compass/ Tenor Harmony
employee 3:
Pete Hogwallop, 1122334455, Yodeling , Horney Toad/Wildcard
Due to the length of some of the strings of this input, the formatting of the output was misaligned.