# **Programming with Python**

# Chapter9:

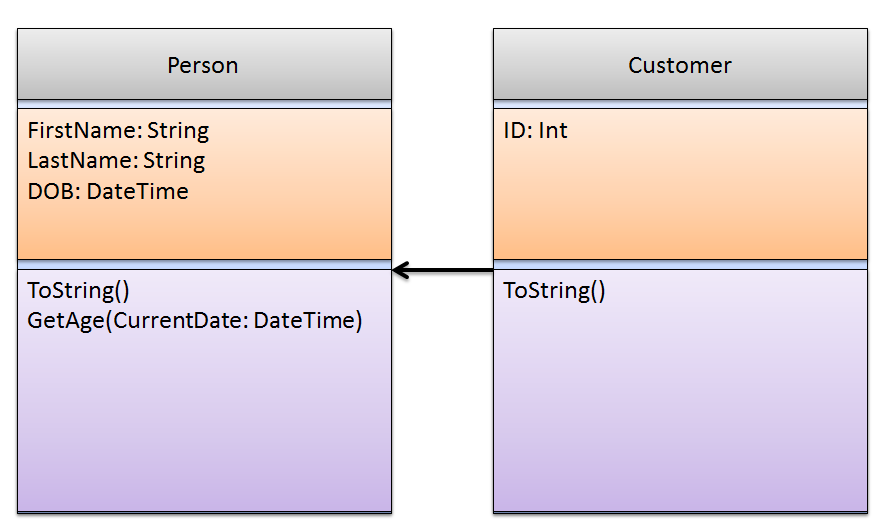


* UML
* Modules
* Applications

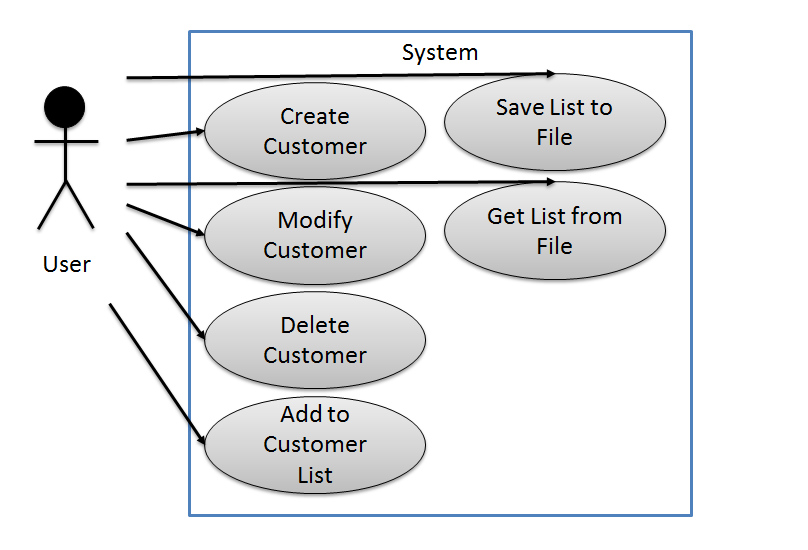
## UML

Unified Modeling Language (UML) is a standard of modeling object-oriented software. UML consists of many type of modeling diagrams, but here are three of the most common ones.

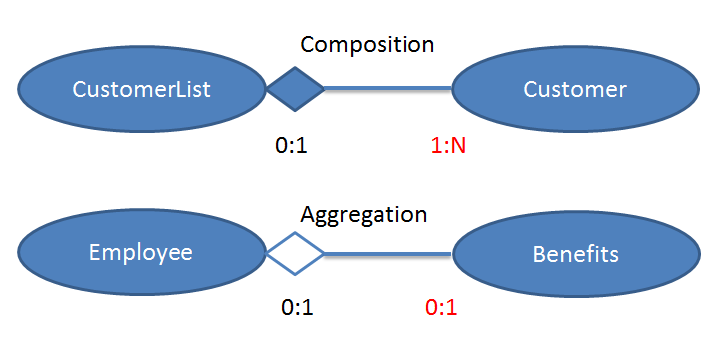
### Class Diagrams



### Use Case Diagrams



### Composition Diagrams



## Modules

"When a module named spam is imported, the interpreter searches for a file named spam.py in the directory containing the input script and then in the list of directories specified by the environment variable PYTHONPATH." -- Python help files

**#------------DataProcessor.py ---------------#**

#Desc: Classes that read and write data

#Dev: RRoot

#Date: 12/12/2020

#ChangeLog:(When,Who,What)

#----------------------------------------#

**class File(object):**

""" Process data using files """

#-------------------------------------#

#Desc: Process data using a file

#Dev: RRoot

#Date: 12/12/2015

#ChangeLog:(When,Who,What)

#-------------------------------------#

**#--Fields--**

#FileName = name of file

#TextData = data read from and written to file

**#--Constructor--**

def \_\_init\_\_(self, FileName = "SavedData.txt", TextData = ""):

#Attributes

self.FileName = FileName

self.TextData = TextData

**#--Properties--**

#FileName

@property

def FileName(self):

return self.\_\_FileName

@FileName.setter

def FileName(self, Value):

self.\_\_FileName = Value

#TextData

@property

def TextData(self):

return self.\_\_TextData

@TextData.setter

def TextData(self, Value):

self.\_\_TextData = Value

**#--Methods--**

def SaveData(self):

"""Writes data"""

try:

objFile = open(self.FileName, "a")

objFile.write(self.TextData)

objFile.close()

except Exception as e:

print("Python reported the following error: " + str(e))

return "Data Saved"

def GetData(self):

"""Reads data"""

try:

objFile = open(self.FileName, "r")

self.TextData = objFile.read()

objFile.close()

except Exception as e:

print("Python reported the following error: " + str(e))

return self.TextData

def ToString(self):

"""Explictly returns field data"""

return self.FileName + "," + self.TextData

def \_\_str\_\_(self):

"""Implictly returns field data"""

return self.ToString()

**#--End of class--**

Now we can use the module from another python script. Creating a small script like this to test the class is known as using a "Test Harness"

**import DataProcessor**

objP = DataProcessor.File()

objP.FileName = "Test.txt"

objP.TextData = "This is a test"

strMessage = objP.SaveData()

print(strMessage)

Checking to see if the module is being ran directly.

**if \_\_name\_\_ == "\_\_main\_\_":**

**raise Exception("This file is not meant to ran by itself")**

Checking to see that a script is not being used as a module.

**if \_\_name\_\_ == "\_\_main\_\_":**

**import DataProcessor, Employees**

else:

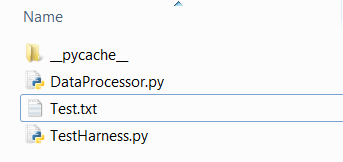
raise Exception("This file was not created to be imported")



## LAB 9-1:

1. Create the DataProcessing Class.
2. Create a test harness and test that it works.
3. Add code to make sure other developers do not run the DataProcessing class as a stand-alone script.

**NOTE:** Python will automatically create a subfolder called *\_\_pycache\_\_* when you first use the module.





## Inheritance with Modules

You can use modules to create parent and child class as well.

**#------------Persons.py Module ---------------#**

#Desc: Classes that hold Personal data

#Dev: RRoot

#Date: 12/12/2020

#ChangeLog:(When,Who,What)

#---------------------------------------------#

**if \_\_name\_\_ == "\_\_main\_\_":**

**raise Exception("This file is not meant to run by itself")**

**#--- Make the class ---**

class Person(object):

""" Base Class for Personal data """

#-------------------------------------#

#Desc: Holds Personal data

#Dev: RRoot

#Date: 12/12/2020

#ChangeLog:(When,Who,What)

#-------------------------------------#

**#--Fields--**

\_\_Counter = 0 #Hey Devs, please consider this a private class field. Thx!

**#--Constructor--**

def \_\_init\_\_(self, FirstName = ""):

#Attributes

self.\_\_FirstName = FirstName # Private Attribute

Person.\_\_SetObjectCount() # Private Method

**#--Properties--**

#FirstName

@property #getter(accessor)

def FirstName(self):

return self.\_\_FirstName

@FirstName.setter #(mutator)

def FirstName(self, Value):

self.\_\_FirstName = Value

**#--Methods--**

def ToString(self):

"""Explictly returns field data"""

return self.FirstName

def \_\_str\_\_(self):

"""Implictly returns field data"""

return self.FirstName

@staticmethod

def GetObjectCount(): # You do not need the self keyword

return Person.\_\_Counter

@staticmethod

def \_\_SetObjectCount(): # This is a private and static method

Person.\_\_Counter += 1

**#--End of class Person--**

Now that you have a Person base class you can create a Child class like this one:

#------------Employees.py ---------------#

#Desc: Classes that hold employee data

#Dev: RRoot

#Date: 12/12/2020

#ChangeLog:(When,Who,What)

#---------------------------------------------#

**import Persons**

if \_\_name\_\_ == "\_\_main\_\_":

raise Exception("This file is not meant to run by itself")

**#--- Make child class ---**

class Employee(**Persons**.Person):

""" Class for Employee data """

#-------------------------------------#

#Desc: Holds Employee data

#Dev: RRoot

#Date: 12/12/2020

#ChangeLog:(When,Who,What)

#-------------------------------------#

**#--Fields--**

#Id = Employee Id

**#--Constructor--**

def \_\_init\_\_(self, Id = ""):

#Attributes

self.\_\_Id = Id

**#--Properties--**

# Id

@property #getter(accessor)

def Id(self):

return self.\_\_Id

@Id.setter #(mutator)

def Id(self, Value):

self.\_\_Id = Value

**#--Methods--**

def ToString(self):

"""Explictly returns field data"""

strData = super().ToString()

return str(self.Id) + ',' + strData

def \_\_str\_\_(self):

"""Implictly returns field data"""

return self.ToString()

**#--End of Class Employee --**

One module file can hold many classes, but each should be associated with the same subject matter. Here is an example:

NOTE: This code is added to the Employees.py file

**#--End of Class Employee --**

**class EmployeeList(object):**

""" Static class for holding a list of Employee data """

#-------------------------------------#

#Desc: Manages a list of Employee data

#Dev: RRoot

#Date: 12/12/2020

#ChangeLog:(When,Who,What)

#-------------------------------------#

**#--Fields--**

\_\_lstEmployees = [] # a list with Employee objects

**#--Constructor--**

#@staticmethod in python constructors cannot be static

#def \_\_init\_\_():

#Attributes

**#--Properties--**

#None

**#--Methods--**

**@staticmethod**

def AddEmployee(Employee):

print(Employee.\_\_class\_\_)#for testing

if(str(Employee.\_\_class\_\_) == "<class 'Employees.Employee'>"):

EmployeeList.\_\_lstEmployees.append(Employee)

#print(EmployeeList.\_\_lstEmployees)#for testing

else:

raise Exception("Only Employee objects can be added to this list")

**@staticmethod**

def ToString(): # This overrides the original method (it's polymorphic)

"""Explictly returns field data"""

strData = "Id,FirstName,LastName\n"

for item in EmployeeList.\_\_lstEmployees:

strData += str(item.Id) + "," + item.FirstName + "," + item.LastName + "\n"

return strData

**@staticmethod**

def \_\_str\_\_(): # This overrides the original method as well

"""Implictly returns field data"""

strData = EmployeeList.ToString

return strData

**#--End of Class --**

Here is another example. This one would be added to the DataProcessing.py file:

#--End of class File--

**class Database(object):**

""" Process data using files """

#-------------------------------------#

#Desc: Process data using a Database

#Dev: RRoot

#Date: 12/12/2020

#ChangeLog:(When,Who,What)

#-------------------------------------#

**#--Fields--**

#ConnectionString = Code to connect to DB

#TextData = data read from and written to DB

**#--Constructor--**

def \_\_init\_\_(self, ConnectionString):

#Attributes

self.ConnectionString = ConnectionString

**raise Exception("Our Class is not ready yet!")**

**#--End of class--**

Now you would create a test harness to test your code:

**import DataProcessor, Persons, Employees**

print("Test the DataProcessor.File class")

objDP = **DataProcessor**.File()

objDP.FileName = "Test.txt"

objDP.TextData = "This is a test"

strMessage = objDP.SaveData()

print(strMessage)

print("\n Test the DataProcessor.Database class")

try:

print("Trying to create an object, but the class is not ready")

objDP = **DataProcessor**.Database()

except:

print("This should fail")

print("\n Test the Persons.Person class")

objP = **Persons**.Person()

objP.FirstName = "Bob"

objP.LastName = "Smith"

print(objP.ToString())

print("\n Test the Employees.Employee class")

objE = **Employees**.Employee()

objE.Id = 1

objE.FirstName = "Bob"

objE.LastName = "Smith"

print(objE.ToString())

print("\n Test the Employee.EmployeeList class")

objEL = **Employees**.EmployeeList()

try:

print("Trying the wrong object type")

objEL.AddEmployee(objP)

except:

print("This should fail")

try:

objEL.AddEmployee(objE)

print("Trying the correct object type")

print(objEL.ToString())

except:

print("This should work")



## LAB 9-2:

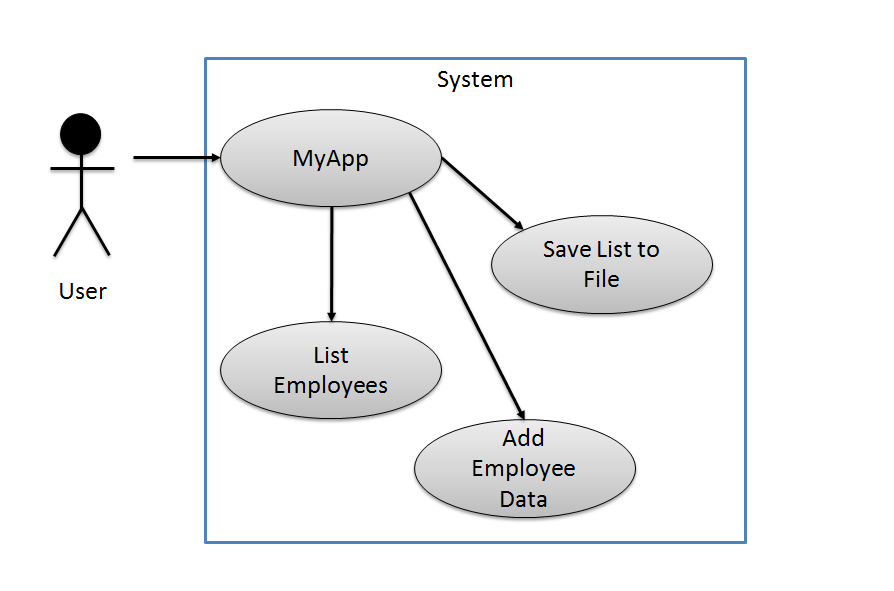
1. Create a Person Base Class in its own module called Person***s***.py.
2. Create an Employee Child Class in its own module called Employee*s*.
3. Create a test harness and test that they work.



## Creating an Application (an example)

To create an application you identify what you are trying to accomplish and plan it out. After that, you implement and test your application.

### Planning



You then identify what data and processing you will need to accomplish your goal:

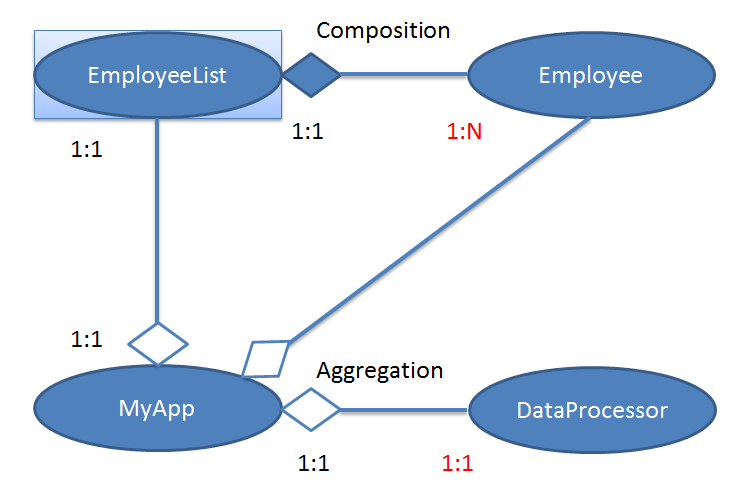
Data:

* Employee Id
* Employee First Name
* Employee Last Name
* File Name

Processing

* Get user Input
* Save data to a file

You create an object diagram like this to outline the objects that you will use:



### Implementing

Once you have your plan you decide if you need to create new scripts and modules or if you can reuse existing ones. For our project we can reuse the DataProcessing, Person, and Employees modules.

### Creating a Main Script file

Now we need to create a script file that will act as our main application.

#-------------------------------------------------#

# Title: EmployeeApp

# Dev: RRoot

# Date: 12/12/2020

# Desc: This application manages employee data

# ChangeLog: (Who, When, What)

#

#-------------------------------------------------#

if \_\_name\_\_ == **"\_\_main\_\_**":

***import DataProcessor, Employees***

else:

raise Exception("This file was not created to be imported")

**#-- Data --#**

**# declare variables and constants**

objE = None #an Employee object

intId = 0 #an EmployeeId

gIntLastId = 0 #Records the last EmployeeId used in the client

strFirstName = "" #an Employee's first name

strLastName = "" #an Employee's last name

strInput = "" #temporary user input

**#-- Processing --#**

**#perform tasks**

def **ProcessNewEmployeeData**(Id, FirstName, LastName):

try:

#Create Employee object

objE = ***Employees***.Employee()

objE.Id = Id

objE.FirstName = FirstName

objE.LastName = LastName

***Employees***.EmployeeList.AddEmployee(objE)

except Exception as e:

print(e)

def **SaveDataToFile**():

try:

objF = ***DataProcessor***.File()

objF.FileName = "EmployeeData.txt"

objF.TextData = ***Employees***.EmployeeList.ToString()

print("Reached here")

objF.SaveData()

except Exception as e:

print(e)

**#-- Presentation (I/O) --#**

#\_\_main\_\_

**#get user input**

strUserInput = ""

while(True):

strUserInput = input("Would you like to add Employee data? (y/n)")

if(strUserInput == "y"):

#Get Employee Id from the User

intId = int(input("Enter an Employee Id (Last id was " + str(gIntLastId) + "): "))

gIntLastId = intId

#Get Employee FirstName from the User

strFirstName = str(input("Enter an Employee First Name: "))

#Get Employee LastName from the User

strLastName = str(input("Enter an Employee Last Name: ") )

#Process input

**ProcessNewEmployeeData**(intId, strFirstName, strLastName)

else:

break

**#send program output**

print("The Current Data is: ")

print("------------------------")

print(**Employees**.EmployeeList.ToString())

**#get user input**

strInput = input("Would you like to save this data to the dat file?(y/n)")

if(strInput == "y"):

**SaveDataToFile**()

#send program output

print("data saved in file")

else:

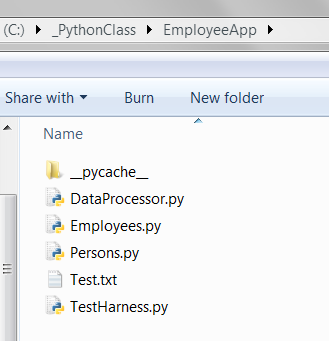
print("data was not saved")

print("This application has ended. Thank you!")



## LAB 9-4:

1. Create folder for you application called C:\\_PythonClass\*EmployeesApp*.
2. Copy the DataProcessing, Persons, and Employees modules to the folder.
3. Use your test harness and test that they still works.



1. Create a new file called MyApp.py in the EmployeesApp folder.
2. Add the main application code to the file.
3. Test that your application works.

