Lab 07: Classes and Objects

Goals for this lab:

- Define a class
- Instantiate objects from programmer defined classes
- Setup appropriate getter and setter functions
- Define a class constructor
- Define class functions
- Create Lists of objects
- Create UML diagrams of classes
- Write, compile, and execute Python programs using the given instructions

General Guidelines:

- Use meaningful variable names
- Use appropriate indentation
- Use comments, especially for the header, variables, and blocks of code.

Example Header:

```
# Author: Your name
# Date: Today's date
# Description: A small description in your own words that describes what
the program does. Any additional information required for the program to
execute should also be provided.
```

Example Function Docstring:

```
"""
General function description
:param p1: DESCRIPTION
:type p1: TYPE
:param p2: DESCRIPTION
:type p2: TYPE
:return: DESCRIPTION.
"""
```

1. Basic Data Storage

One of the basic functionalities provided by classes is that of storing multiple pieces of data which may have different data types.

In this exercise, you will construct a program that prompts a user for information regarding an order for a product. Once the order is complete, your program should then calculate the total cost of the order and report that information back to the user.

To complete this program:

- In a file named **Product.py** , define a class named `Product` to store information related to the product the user wishes to order. The class should contain:
  - A constructor that declares:
    - A variable to store the product's name
    - A variable to store the number of units of product the user wishes to order
    - A variable to store the price of a single unit of the product
    - A variable to store the total cost of the ordered products
  - Appropriate getter and setter functions for each variable
  - Also make sure that each variable is private (remember the naming convention) to provide correct encapsulation
- In a file named **orderSystem.py** , define:
  - A function named `calcTotal` that returns nothing and takes a `Product` object as a parameter. The function should calculate the total cost of the order using the information from the `Product` object (i.e. the number of units multiplied by the price) and then store the total cost in the appropriate variable in the `Product` object.
  - A `main` function, which should:
    - Create an instance of a `Product` object
    - Prompt the user for the name of the product and the number of units they want to order of that product, and store that information in the `Product` object
    - Set the price of the `Product` object to 9.99
    - Using the `calcTotal` function calculate and store the total price of the order in the `Product` object
    - With an appropriate message, output to the user the full statement of their order including the product name, the number of units they wanted, the price of a single unit, and the total cost of their order. Be sure that all monetary values are output with a $ in front of the value and with exactly two decimal places of precision.

Complete the program, making sure to execute it to verify that it produces the correct results. Note that you will submit both the **orderSystem.py** and **Product.py** files to Canvas.

2. Internal Consistency

In addition to simply storing different types of data and basic getter/setter functions, objects can also store more complicated functions to manipulate and update the object's internal data. One of the first functions to execute in an object, is the object's constructor. This function is called whenever you instantiate a new instance of the class. While, for the most part, it is not required that you define a constructor for you class, due to a class always having a default constructor provided by Java, it is sometimes helpful to defined one to initialize certain variables when the object is being created.

For this exercise you will be creating a warehouse inventory tracker. You will first need to create a warehouse class and then use that class, along with a menu implemented via a switch, to allow the user for modify the contents of the warehouse.

- In a file named **Warehouse.py** , define a class named `Warehouse` to store information related to the goods in the warehouse. The class should contain:
  - A constructor function that takes in a single parameter representing the initial amount of goods in the warehouse, declares a private variable to store the total amount of goods in the warehouse, and uses the parameter variable to set the initial total amount of goods
  - A function that prompts the user for how many goods they would like to add to the warehouse and then increments the total number of goods by that amount
  - A function that prompts the user for how many goods they would like to remove from the warehouse and then decrements the total number of goods by that amount. Your function should not remove more goods than you currently have, and should warn this user if they try to make this mistake
- In a file named **manager.py** , define:
  - A main function, which should:
    - Declare a single instance of a `Warehouse` object, with the total number of goods in the warehouse set to 0. Be sure to use your constructor to set the value
    - A loop that outputs a menu of options regarding the warehouse (add goods, remove goods, output total, quit), prompts the user for which option they would like, and then determines which function to call in the `Warehouse` object OR quits the loop if they select quit

Complete the program, making sure to execute it to verify that it produces the correct results. Note that you will submit both the **Warehouse.py** and **manager.py** files to Canvas.

3. `Lists` of Objects

With a combination of data and functions in objects, we now have the ability to maintain far more information in a much more organized system. By creating `Lists` of objects, we can now really start to provide more complex collections of information in a programmer friendly system. Additionally, we can have our objects be a bit smarted in how they perform their actions to decrease the amount of work that needs to be implemented by the programmer.

For this exercise, you will be creating an updated version of your ordering system from part 1 of this lab. First copy your code from **orderSystem.py** into a new file **smartOrderSystem.py** . Next copy your code from **Product.py** into a new file **SmartProduct.py** . Be sure to update your class names accordingly.

- In your `SmartProduct` class, make the following updates and additions:
    - In your constructor add a private variable to store the product's ID number
    - Provide appropriate getter/setter functions for the ID variable
    - Alter the constructor's parameter list such that it takes in a variable to represent the product's ID number, a variable to store the product's name, a variable to store the number of units of product the user wishes to order, and a variable to store the price of a single unit of the product
        - The constructor should use all of this information to set the appropriate variables in the object
        - The constructor should also calculate the total price of the product and store that value in the appropriate variable
    - All other functions and variables should remain the same
- In your **smartOrderSystem.py** file, make the following updates and additions:
    - Update your `calcTotal` function to take a `List` of `SmartProduct` objects as a parameter. This function should now iterate through all of the products, sum up all of the total costs (which have already been calculated by the objects), and return the summed total
    - In your `main` function:
        - Comment out all of the previous code (You can remove this code once you are done with the updates, but some of the code can be reused when updating)
        - Create a variable to store a `List` of `SmartProduct` objects
        - Prompt the user for how many products they wish to order
        - Using a loop that allows the user to input as many products as they just specified:
            - Prompt the user for the name of the product they wish to order and the number of units they wish to order
            - Create a new `SmartProduct` object with a unique product ID number, the name the user specified, the number of units the user specified, and a price of 9.99, and store the object in the `List`
        - Output to the user, with an appropriate message and in a well-organized fashion, all of the information regarding products they ordered and the sum total for their order. Do not forget that all monetary values are output with a $ in front of the value and with exactly two decimal places of precision.

Complete the program, making sure to execute it to verify that it produces the correct results. Note that you will submit both the **smartOrderSystem.py** and **SmartProduct.py** files to Canvas.

4. Diagramming

Using the classes you defined in **Warehouse.py** and **SmartProduct.py** create UML diagrams of each class. Make sure you include all of the member variables and member functions. Save these diagrams in a PDF file named **UMLDiagrams.pdf** and submit it to Canvas.

Submission

Once you have completed this lab it is time to turn in your work. Please submit the following four sets of files to the Lab 07 assignment in Canvas:

- **orderSystem.py, Product.py**
- **manager.py, Warehouse.py**
- **smartOrderSystem.py, SmartProduct.py**
- **UMLDiagrams.pdf**

Sample Output

orderSystem.py, Product.py

```
Please enter the name of the product you wish to order: Shoes
Please enter the number of units of product you wish to order: 5
You ordered:
Name: Shoes
Units: 5
Price: $9.99
Total Cost: $49.95
```

manager.py, Warehouse.py

```
Please select from the following options:
        1: Add Goods
        2: Remove Goods
        3: Output Total Goods
        4: Quit
        Choice: 1
How many goods would you like to add: 5
Please select from the following options:
        1: Add Goods
        2: Remove Goods
        3: Output Total Goods
        4: Quit
        Choice: 2
How many goods would you like to remove: 3
Please select from the following options:
        1: Add Goods
        2: Remove Goods
        3: Output Total Goods
        4: Quit
        Choice: 1
```

```
How many goods would you like to add: 8
Please select from the following options:
        1: Add Goods
        2: Remove Goods
        3: Output Total Goods
        4: Quit
        Choice: 2
How many goods would you like to remove: 20
You do not have that many goods!
Please select from the following options:
        1: Add Goods
        2: Remove Goods
        3: Output Total Goods
        4: Quit
        Choice: 3
There are 10 goods in the warehouse.
Please select from the following options:
        1: Add Goods
        2: Remove Goods
        3: Output Total Goods
        4: Quit
        Choice: 4
Good bye!
```

smartOrderSystem.py, SmartProduct.py

```
How many products would you like to order: 2
Please enter the name of the product you wish to order: Shirts
Please enter the number of units of product you wish to order: 5

Please enter the name of the product you wish to order: Pants
Please enter the number of units of product you wish to order: 8

You ordered:
ID: 1
Name: Shirts
Units: 5
Price: $9.99
Total Cost: $49.95
ID: 2
Name: Pants
Units: 8
Price: $9.99
Total Cost: $79.92
The total cost of your order is: $129.87
```

Rubric

For each program in this lab, you are expected to make a good faith effort on all requested functionality. Each submitted program will receive a score of either 100, 75, 50, or 0 out of 100 based upon how much of the program you attempted, regardless of whether the final output is correct (See table below). Additionally, it is expected that you will provide a header comment at the beginning of each program, and so 10 points will be deducted from each program that is missing the header comment. The scores for all of your submitted programs for this lab will be averaged together to calculate your grade for this lab. Keep in mind that labs are practice both for the exams in this course and for coding professionally, so make sure you take the assignments seriously.

Score	**Attempted Functionality**
100	100% of the functionality was attempted
75	<100% and >=75% of the functionality was attempted
50	<75% and >=50% of the functionality was attempted
0	<50% of the functionality was attempted