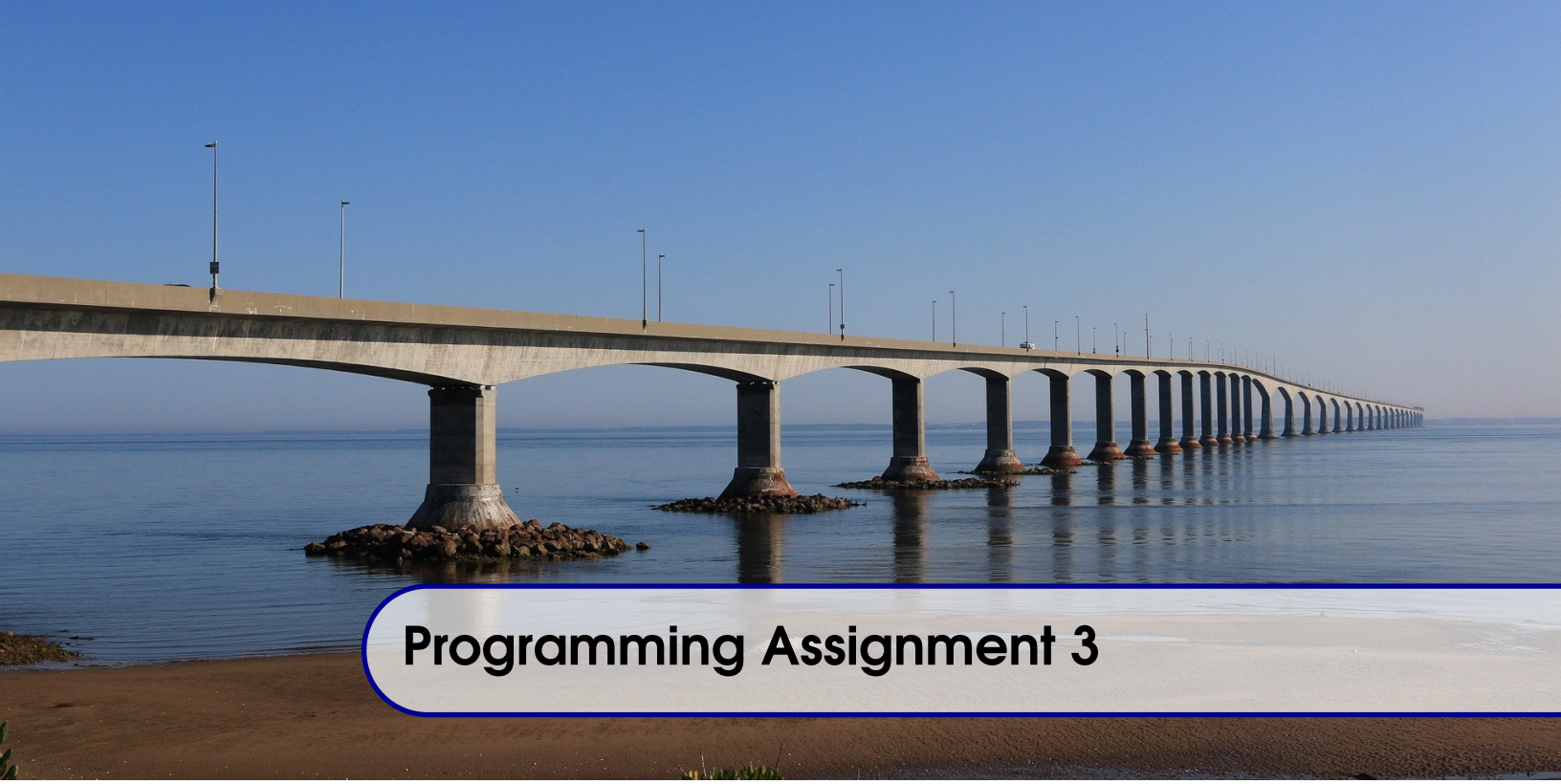

Programming Assignment 3

Assignment Objectives

By completing this assignment you are demonstrating your ability to:

- Design your own related set of classes in the Python programming language.
- Create a number of different classes with a variety of relationships.
- Implement accessor and mutator methods that allow access to your class data
- Create subclasses and reuse code through inheritance.

Scenario

In the previous assignments, you worked on a Product class, a Store class, and a StoreDB (database) class. In this assignment, we are working on something similar but using inheritance. You are given a Product class that will store information about the products. You are also given a Store class that will store information about the stores. Your objective is to design a MyStore class that is going to create a database of your store's branches and products.

Given

As part of this assignment, you have been given the following classes:

Product class

The Product class generates a unique product ID for every new product if its constructor receives a string "*Missing*" as a value of the product id, otherwise it use the input product id. The constructor receives the following product information: product id (string), product name (string), products sold (int), products available (int), product cost (float), and product retail price (float). The *__str__* method represents the product information. In addition, the Product class has the following methods: get_id():String - returns the product id.

get_name():String - returns the product name.
get_num_sold():String - returns the number of products sold.
get_num_available():String - returns the number of products available.
get_cost():String - returns the cost of the product.
get_price():String - returns the retail price of the product

Store class

The Store class's constructor receives the following store information: store id(string), store(branch) name (string), store city (string), store area (string). Each store maintains a dictionary of products where unique product ids act as keys and instances of product class act as values. The *__str__* method represents the store information. In addition, the Store class has the following methods:
get_store_id():String - returns the store id.
get_store_name():String - returns the store(branch) name.
get_city():String - returns the city name.
get_area():String - returns the store area.
get_products():String - returns a dictionary of products.
add_product(pid, product_name, sold, available, cost, price):String - creates a new Product object and add it to the product dictionary.
remove_product(pid):String - removes the product with pid as its product id from the product dictionary.
count_products():Integer - returns the number of products in the product dictionary.

storeInventory.csv

This supporting file contains the database of stores and products. Driver code is going to read this file and pass the data as a list to the constructor of the MyStore class. Any row may contain the store data or the product data. There are 4 columns with labels: RecordID, Branch_or_StoreRecordID, City_or_Sold-Available, Area_or_Cost-RetailPrice. If "RecordID" starts with "S' then the respective row contains the store data (RecordID i.e. store ID, BranchName, City, Area - all as strings). If "RecordID" starts with "P' then the respective row contains the product data (RecordID i.e. product ID as a string, StoreID as a string, products sold and products available as a compound string separated by "-', price and retail cost as a compound string separated by "-').

Objective - MyStore class

You need to write a MyStore class that inherits the Store class, and maintains the database of all stores and products. As part of this assignment, you have been given a MyStore class with driver code, and two dictionary variables - store_obj_list (*a dictionary to store unique store objects*) and product_id_mapping (*a dictionary to store mapping of input product ids to the system-generated product ids*). You need to write your code in the given MyStore class. It must contain:

1. The class constructor will receive a list *dataList* that contains the rows of CSV file as its elements. So, *dataList* is a list of list elements. The constructor will then process the list elements (one at a time) to add new stores and products to the dictionary variables using the following instructions:

- You will check if the list element represents the Store data or the Product data (*hint: check if the RecordID starts with "S" or "P"*).
- If the list element represents the Store data, then call the constructor of the super class (Store) to create its new instance. If the store with the given store_id does not already exists in the dictionary *store_obj_list*, then add this new store object to the dictionary. You will use the input store ids as keys to add the store objects as values to the dictionary.
- This step will explain the use of the dictionary variable *product_id_mapping*. You will use these instructions in the next step while adding products to the store objects. Any row of the CSV file that represents the product data will have an input product id given as the RecordID (that starts with "*P*"). When you call the *add_product()* method of the super class to create a Product object, it requires that you pass the product id as the first argument. If you pass a string "*Missing*" as product id, then the method will generate a new unique product id. Otherwise, it will create a Product object with the received value of the product id. The objective is to maintain a record of all unique products across all stores. So, every time you have a new product, you will pass "Missing" as the product id. The system will generate a unique product id for that product. You will store the mapping of system-generated products ids (*as dictionary values*) to the input product ids i.e. RecordIDs of the CSV (*as dictionary keys*) in the dictionary *product_id_mapping*. When you get the next product, you will check if the id mapping already exists in the dictionary *product_id_mapping* or not. If it already exists, then it means that the product is not unique and you will pass its previously used system-generated id as input to the *add_product()* method. If the mapping does not exist, then you will pass "*Missing*" as input to the *add_product()* method.
- If the list element represents the Product data, then you will add the Product object to its respective store object. You will need to split the composite strings to get the values of the product attributes (*see storeInventory.csv section for more details*). If the store with the given store id does not exist, then you will print "*Store ID XX does not exist! Adding Product is not possible!*", where *XX* is the store id given in the Product data. If the store with the given store id already exists in the dictionary *store_obj_list*, then you will add the product to its respective store object using the *add_product()* method of the super class. You need to follow the instructions given in the previous step to add a product object to the store object by making use of mapping stored in the dictionary *product_id_mapping*.
- In the end, the constructor will print "*Constructor added XX store objects to the dictionary store_obj_list!*", where *XX* is the number of store objects added to the *store_obj_list* dictionary.

2. The *print_stores* method will print all Store objects from the *store_obj_list* dictionary. Please note that the super class has a *__str__* method that represents/returns the store data as a string.
3. The *print_prod_id_mapping* method will print all mappings of the system-generated products ids (*dictionary values*) to the input product ids (*dictionary keys*) from the *product_id_mapping* dictionary. Please see the given screenshot of the sample run for the required print format.
4. The *print_product_in* will receive a product id, check if the product exists in the database, and print the product information as shown in the screenshot with the output of a sample run. You will check if the mapping of the input product id exists in the product_id_mapping dictionary or not. If the mapping does not exist, then you will print "*PID does not exist in*

the database!". If mapping exists, then you will iterate through all the store objects of the *store_obj_list* dictionary, and print the product information for all stores that carry the product in question (*i.e. product that has the input value as its product id*). Please note that the product information can be printed using the *__str__* method of the Product class. You can concatenate the value of the store id with the string returned from the *__str__* method.

5. The *add_new_store* will receive store id, store name, store city, and store area. You will create a store object and add it to the *store_obj_list* if it does not exist already. If a new store object is added to the list, then you will print two statements - a) "*New store object with store id XX successfully added to the database!*", where *XX* is the input store id, and b) "*Updated dictionary store_obj_list size: XX*", where *XX* is the size of the *store_obj_list*. If the store already exists, then you will print "*Store with store id XX already exists!*", where *XX* is the input store id.
6. The *add_new_product* will receive store id, product id, product name, products sold, products available, product cost, and product retail price. If a store with input store id does not exist, then you will print "*Store ID XX does not exist! Adding Product is not possible!*", where *XX* is the input store id. Otherwise, the method will create a product object and add it to the store object using the *add_product* method. Please note that you will still use and maintain the mapping of product ids from the dictionary *print_prod_id_mapping* identical to what you did in the constructor design.
7. The *delete_store* method will receive a store id, search the respective Store object in the *store_obj_list* dictionary, and delete the Store object if it exists. On successful deletion, you will print "Successfully deleted store with Store ID XX.", where XX is the store id that method received. If the *store_obj_list* dictionary does not have a store with the given store id, then you will print "Store ID *XX* does not exist! Deletion is not possible!", where *XX* is the store id that method received.
8. You should provide appropriate docstrings and in-line commenting for your implemented methods.

Submitting your assignment

Your submission will consist of one zip file containing solutions to all problems (see below). This file will be submitted via a link provided on our Moodle page just below the assignment description. This file must be uploaded by 5pm (Moodle time) on the due date in order to be accepted. You can submit your solution any number of times prior to the cutoff time (each upload overwrites the previous one). Therefore, you can (and should) practice uploading your solution and verifying that it has arrived correctly.

Example:

Submission file: `asn3_studentnum.zip` , where `studentnum` is your student ID number.

Your zip file should contain a single Python file `myStore.py`, and a pdf file with the screenshots of a single sample run of your code.

Driver Script and Sample Run

The *myStore.py* file has been provided as part of this assignment. This file includes the necessary driver code. You should write your solution in the given *myStore.py* file. A sample run of the

completed code is shown below:

```
Store ID S1015 does not exist! Adding Product is not possible!
Constructor added 10 store objects to the dictionary store_obj_list!
**************************************************************************
Printing Store objects:
StoreID:S1001, Branch:Downtown, City:Gerakaroú, Area:5434 Daystar Circle
StoreID:S1002, Branch:Uptown, City:Kangar, Area:689 Vera Pass
StoreID:S1003, Branch:CityPlaza, City:Jiukeng, Area:634 Jackson Parkway
StoreID:S1004, Branch:Downtown, City:Hekou, Area:5995 Merchant Trail
StoreID:S1005, Branch:HydePark, City:Ettelbruck, Area:84269 Crescent Oaks Way
StoreID:S1006, Branch:VictoriaPark, City:Mnich, Area:14 Mayfield Alley
StoreID:S1007, Branch:LoganPlaza, City:Bov, Area:89058 Monument Hill
StoreID:S1008, Branch:HaltonHills, City:Purral, Area:4496 Killdeer Court
StoreID:S1009, Branch:SouthPark, City:Cercal, Area:4 Charing Cross Plaza
StoreID:S1010, Branch:Downtown, City:Mnich, Area:14 Mayfield Alley
**************************************************************************
Printing ProductID mapping:
InputID:P2001, SystemGeneratedID:PDK1
InputID:P2002, SystemGeneratedID:PDK2
InputID:P2003, SystemGeneratedID:PDK3
InputID:P2004, SystemGeneratedID:PDK4
InputID:P2005, SystemGeneratedID:PDK5
InputID:P2006, SystemGeneratedID:PDK6
InputID:P2007, SystemGeneratedID:PDK7
InputID:P2008, SystemGeneratedID:PDK8
InputID:P2009, SystemGeneratedID:PDK9
InputID:P2010, SystemGeneratedID:PDK10
**************************************************************************
Printing all-stores inventory of Product ID P2005:
Input PID:P2005 is mapped to SystemGeneratedPID PDK5..start printing..
StoreID:S1005, Product Name:Soda; Product ID:PDK5; Sold:323; Available:32; Cost:1896.99; RetailPrice:1232.12
StoreID:S1006, Product Name:Soda; Product ID:PDK5; Sold:248; Available:754; Cost:4071.16; RetailPrice:3690.65
**************************************************************************
```

Figure 1: Sample run - part 1

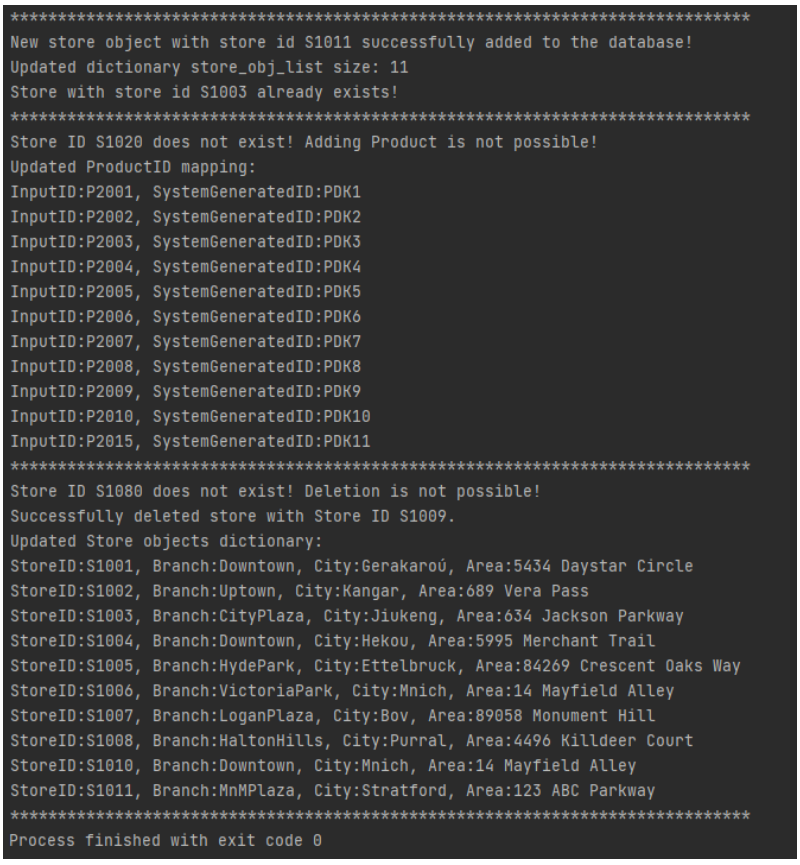

Figure 2: Sample run - part 2.