Madison Li

Github link: https://github.com/mli254/FlowerShop-Class

FlowerShop Class Documentation

Class Description:

This program creates a class modeled after the real-world object, a flower shop. Upon instantiation, an object will have a randomized inventory, with each type of flower having 0-10 items in stock. The class will also have a “default inventory", which is referenced to refresh the inventory of any FlowerShop object. The main function of the class is to simulate purchasing flowers through the use of a shell. The shell should have error handling, and should only exit when the checkout command is called.

Class Attribute:

`default_inventory` `= {"Calla Lily":10, "Rose":10, "Stargazer Lily":10, "Hydrangea":10, "Zinnia":10}`

This class attribute stores a default inventory. All FlowerShop objects will be able to reference this attribute in their `refresh()` method to restock any instance of a FlowerShop. This variable effectively represents what the inventory of a fully-stocked flower shop would look like.

Data Attributes:

`self.__name`

This data attribute represents the name of the flower shop. When creating a FlowerShop object, the user will need to pass in a name, which will be stored within this private attribute.

`self.__flowers`

`= ["Calla Lily", "Rose", "Stargazer Lily", "Hydrangea", "Zinnia"]`

This data attribute represents the types of flowers that are sold by default. The variable is a list of the same flowers that serve as keys in the class’s `default_inventory` attribute. The value should not be changed.

`self.__special_flower`

This data attribute is left empty by default, but may be assigned a value by the `set_special_offer()` method. The variable represents a type of flower not typically sold in the store.

`self.__inventory`

This data attribute stores the inventory of the flower shop via a dictionary. Using the `__flowers` attribute, a random number between 0-10 is generated to indicate the number of that particular flower in stock. As a result, each instance of FlowerShop will start with a randomly generated inventory.

`self.__flower_price`

`= $1.75`

This data attribute represents the price of a single flower. The default value is $1.75, but can be edited through the `set_flower_price()` method.

`self.__bouquet_price`
`= $1.50`
This data attribute represents the price of a single flower within a bouquet. The default value is $1.50, but can be edited through the `set_bouquet_price()` method.

`self.__vase_price`
`= $4.00`
This data attribute represents the price of a vase. The default value is $4.00, but can be edited through the `set_vase_price()` method.

`self.__special_offer`
`= False`
This data attribute tells the class whether a custom flower has been added. By default, it is set to `False`. It will be set to `True` if the `set_special_offer()` method is called.

Methods:

`get_name``(`*`self`*`):`
Returns the current name of the FlowerShop object. No further arguments necessary.

`get_flowers``(`*`self`*`):`
Returns the default flowers of the FlowerShop object. No further arguments necessary.

`get_inventory``(`*`self`*`):`
Returns the current inventory of the FlowerShop object. No further arguments necessary.

`get_flower_price``(`*`self`*`):`
Returns the current price of a single flower within the FlowerShop object. No further arguments necessary.

`get_bouquet_price``(`*`self`*`):`
Returns the current price of a single flower within a bouquet in the FlowerShop object. No further arguments necessary.

`get_vase_price``(`*`self`*`):`
Returns the current price of a vase within the FlowerShop object. No further arguments necessary.

`set_name``(`*`self, name`*`):`

Renames the FlowerShop object to the name specified in the arguments. Requires one string argument, which will serve as the new name for the FlowerShop object. The method does not return anything.

`set_flower_price``(`*`self, price`*`):`
Sets the price of a single flower within the FlowerShop object to a new value. Requires one float or int argument, which will be assigned as the new price for a single flower. This method does not return anything.

`set_bouquet_price``(`*`self, price`*`):`
Sets the price of a single flower within a bouquet in the FlowerShop object to a new value. Requires one float or int argument, which will be assigned as the new price for a single flower in a bouquet. This method does not return anything.

`set_vase_price``(`*`self, price`*`):`
Sets the price of a vase within the FlowerShop object to a new value. Requires one float or int argument, which will be assigned as the new price for a vase. This method does not return anything.

`set_amount``(`*`self, flower, amount`*`):`
Sets the amount of stock for the specified flower to a new value. Requires one string argument for the specified flower and one int argument for the new amount of stock. An error will occur if the amount specified is not of int type. The flower specified also cannot be a flower that is not already being sold. This method does not return anything.

`set_special_offer``(`*`self, flower, amount`*`):`
Adds a new type of flower to the inventory, along with the specified stock. Requires one string argument for the special flower and one int argument for the amount of stock. The flower specified cannot be a flower already being sold. Furthermore, special offers will disappear after the inventory is refreshed. Adding a special offer flower will print a special message in the `shop()` shell, identifying the special flower indicated. This method does not return anything.

`refresh``(`*`self`*`):`
Refreshes the FlowerShop's inventory to default. This method does not accept any arguments, and does not return anything.

`display_prices``(`*`self`*`):`
Displays the current prices for the FlowerShop object by printing to the terminal. This method does not accept any arguments, and does not return anything.

`__str__``(`*`self`*`):`
The class's string magic method. Returns the FlowerShop object's current inventory.

`shop``(`*`self`*`):`

A shell that simulates purchasing from a flower shop. Once called, the user can interact with the shell using commands within the terminal. The program will only close when the `checkout` command is called.

Available commands are:

`help`

Lists the available commands, and adds additional clarification.

`buy`

Prompts the user to select between purchasing 1) a single flower 2) a flower bouquet or 3) a flower bouquet in a vase. Depending on the option, the shell will ask the user for their choice of flower or flowers, and indicate that a purchase has been successful. The user will be prompted again if the flower chosen is out of stock. An incorrect input will result in the order being cancelled, and the user will be prompted to type their order again. The shell will print the current total cost into the terminal at the end of the transaction.

`refresh`

Restocks the store. Keep in mind that any special offers will be ended.

`inventory`

Shows the user the store's current inventory.

`checkout`

Returns the total cost of purchases and the amount of each type of purchase, and then exits the shell.

Demo Program

The demo program starts by creating a new FlowerShop object, and then tests each of the methods within the class. These tests should be accompanied by corresponding headers and descriptions to indicate to the user what is being tested. The `shop()` method is the last method called, and allows the user to test out the shell and its commands. The user will then have to exit the program themself by calling the `checkout` command within the shell.

Instructions:

To run the program, the user needs to download `flower_shop.py` and navigate to the file's location through the terminal. Then the user just needs to input `python3 flower_shop.py` and the program should run, and end with the `shop()` shell waiting for the user's input. The user can have the shell run as long as they want, and then exit through the checkout command.