Intermediate Python Programming

Description: Fun with Design Patterns

In this project, you will be using several design patterns together to emulate a task – a restaurant creating meals for customers. Unlike previous labs, this one is for you to figure out mostly on your own. As a result, it will require a little less code, but it does require a descent understanding of the design patterns employed (please focus on these lessons):

Singleton:	A class of which only a single instance can exist
Factory:	Creates an instance of several derived classes
Façade:	Wrap a complicated subsystem with a simpler interface

Instructions

- Please follow all instructions closely and read this in its entirety before writing ANY code. There is no penalty for going "above and beyond" the assignment requirements. However, failure to meet all required parts can affect your grade. Please consult the rubric for each section.
- For this assignment you will create a single "raw" Python file, or Jupyter Notebook file.

Part I – Setting up the Singleton

The singleton class for this lab is called Restaurant. The idea is that there can only ever be one restaurant that the customer goes to. It will have at least the following methods:

- `__new__(cls, *args, **kwargs)`
  - This method intercepts any call where a new Restaurant is being created.
  - Use the Singleton creational design pattern example to ensure only one instance of Restaurant is ever created and return that instance.
  - Optionally, assign two variables to that instance called _orders and _total_sales that will keep track of the number of orders placed in the restaurant and their cumulative price. This will be part of the extra credit.
- `__str__(self)`
  - This is only used in the extra credit portion of the assignment. If attempted, return a string that informs the user how many orders were placed and how much money the restaurant brought in.
  - Use the variables discussed in the __new__() method.
- `order_food(self, food_type)`
  - This method is the gateway to the factory design pattern.
  - For now, the only thing you can effectively put in this method is reference to code that will be added later.
  - Looking forward, it will call the static order_food method of the Food class, passing it food_type as its only argument.

- Additionally, this is where you will have to alter the _orders and _total_sales variables if you are going down that route.

Scoring:

Criteria	Points
__new__(): ensure only 1 can be created!	10
order_food(): wrapper call to Food.order_food()	10
TOTAL	**20**
EXTRA CREDIT	
_orders and _total_sales variables	3
__str__() method	2

Part II – Food Factory Base Class

This class acts as the base class for other concrete Food classes, like Cheeseburger and Pasta (see Part III). As the title hints at, this class will also serve as the factory for those derivative classes. This class should have the following methods:

- `__init__(self)`
  - This method is mostly a placeholder. You may want to print a message when it is called while debugging.
- `price(self)`
  - Returns the price of the item.
  - This is also a placeholder for use by derived classes. It only needs to return 0.
- `prepare(self)`
  - This method is a placeholder for use by derived classes.
  - In derived classes, this method acts as a façade that encapsulates the complex process for making this specific food.
- `order_food(food_type)  # Notice no 'self'`
  - This method acts as the factory method for making food objects.
  - This method should be static so mark it with the @staticmethod decorator.
  - Here's the process for kicking off the factory:
    - Make sure food_type is a string and trim and convert it to lower case
    - Create the object of the type called for by the string
      - So if it is "cheeseburger":
        - food = Cheeseburger()
    - Once your food object is created, you will call its prepare() method

Scoring:

Criteria	Points
Food class	
__init__()	5
price()	5
prepare()	5

order_food()	10
TOTAL	**25**

Part III – Making the Food derivatives and their façade processes

Now, you need to create the Food class derivatives and the logic for their prepare() methods. AT a minimum, provide two derivative classes, maybe Cheeseburger and Pasta.

- `__str__(self)`
  - This should return a string showing the food type and its price. Example:
    - "Cheeseburger: 5.99"
  - Do not hard-code the values. They should come from the class itself.
    - Hint: what does __class__.__name__ display?
- `price(self)`
  - Returns the price of the item. This will be whatever value you think is fair for the item.
- `prepare(self)`
  - This method encapsulates the façade algorithm for the specific type of food. Typically, each step would be a completely different method, like boil_water(). However, for this lab, you may simply use a print statement and a sleep() call to emulate these sub-processes. Example:

```
print("Pasta: boiling water.")
sleep(2)
```

Extra credit: Add a third Food derivative of your choice.

Now, create a main() method, or just a module-level block of code within the (__name__ == "__main__") test and make some food!

Here is some code that you can use (assuming you made Cheeseburger and Pasta classes):

```
def main():
    r = Restaurant()
    food = r.order_food("cheeseburger")
    if food:
        print(food)

    food = r.order_food("pasta")
    if food:
        print(food)

    food = r.order_food("mac and cheese")  # doesn't exist, prints failure message
    if food:
        print(food)
    print(r) # If you did extra credit, it will show number of orders and total sales

    # Use this test to prove we have a single instance of Restaurant:
    r2 = Restaurant()
    print(r2)

if __name__ == "__main__":
    main()
```

The output generated by this (yours may differ based on the steps each food type takes in your code):

```
Cheeseburger: grill all-beef patty
Cheeseburger: flip patty
Cheeseburger: put cheese on patty
Cheeseburger: put patty on bun and add toppings
Cheeseburger: All done!
Cheeseburger: 5.99
Pasta: boil water for noodles
Pasta: saute onions, garlic and tomatoes for sauce
Pasta: put noodles in water
Pasta: season the sauce
Pasta: plate noodles and add sauce on top
Pasta: All done!
Pasta: 8.99
Sorry, the restaurant does not make 'mac and cheese'
This restaurant had 2 orders today for a total of $14.98 in sales
This restaurant had 2 orders today for a total of $14.98 in sales
```

Scoring:

Criteria	**Points**
For each Food derivative (x2):	
__str__()	5
price()	5
prepare()	10
Successful test and output	15
TOTAL	**55**
EXTRA CREDIT	
Extra Food class derivative	5