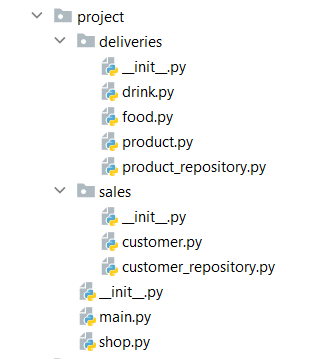
**Python OOP Exam - 18 April 2021**

Maria is expanding her business and today she is opening a grocery shop. You are hired to write a program which keeps track of the shop's inventory. You will be provided with a skeleton which includes all the folders and files that you will need.

***Note: You are not allowed to change the folder and file structure and change their names!***

[](https://user-images.githubusercontent.com/67734870/115264966-74d41600-a13f-11eb-9992-a592c0767d35.png)

**Judge Upload**

***For the***[***first 2 problems***](https://github.com/borislavstoychev/exam_preparation/tree/main/exams_OOP/grocery_shop/project)***, create a zip file with the name project and upload it to the judge system For the last problem, create a zip file with the name tests and upload it to the judge system Structure (Problem 1) and Functionality (Problem 2)***

Your first task is to implement the structure and functionality of all the classes (properties, methods, inheritance, etc.)

**1. Class Product**

In the product.py file the class Product should be implemented. It is a base class for any type of food and drink, and it should not be able to be instantiated. ##$ Structure

**The class should have the following attributes:**

• name: str - passed upon initialization. If the name of the product is empty string, raise a ValueError with message "The product cannot be an empty string."

• quantity: int - passed upon initialization. If the quantity is equal to or below 0, raise a ValueError with message "Quantity cannot be equal to or below zero."

**Methods**

\_\_init\_\_(name: str, quantity: int)

The **init** method should have a name and a quantity.

**2. Class Drink**

In the file drink.py the class Drink should be implemented.

**Structure**

The class should inherit from the Product class.

**Methods**

\_\_init\_\_(name: str)

An instance of the Drink class will have name and quantity of 10.

**3. Class Food**

In the food.py file the Food class should be implemented

**Structure**

The class should inherit from the Product class.

**Methods**

\_\_init\_\_(name: str)

An instance of the Food class will have name and quantity of 15.

**4. Class ProductRepository**

In the product\_repository.py file the class ProductRepository should be implemented. It is a repository for all the products that are delivered to the grocery shop.

**Structure**

**The class should have the following attributes:**

• products: list – empty list upon initialization that will contain all products (objects).

**Methods**

\_\_init\_\_()

The **init** method should have an empty list of products.

**add(product: Product)**

• If a product already exists, raise ValueError with the message "Product {product\_name} already exists.".

• Otherwise, add the product to the products list and return a message "Product {product\_name} successfully added to inventory."

**decrease(product: Product, quantity: int)**

• Decrease the quantity of the given product and return a message "Left quantity of {product\_name}: {product\_quantity}".

**find(product\_name: str)**

• Returns a product (object) with that name.

• If the product does not exist, returns the message "None".

**5. Class Customer**

In the customer.py file the Customer class should be implemented.

**Structure**

**The class should have the following attributes:**

• name: str – passed upon initialization. If the name of the customer is empty string, raise ValueError with message "The customer's name cannot be an empty string."

• products: dict[str, int] - empty dictionary upon initialization that will contain all bought product's names as key argument and bought quantity as value argument.

**Methods**

\_\_init\_\_(name: str)

An instance of the Customers class will have name and products.

**6. Class CustomerRepository**

In the customer\_repository.py file the CustomerRepository class should be implemented. It is a repository for the customers shopping in the grocery.

**Structure**

**The class should have the following attributes:**

• customers: list – empty list upon initialization that will contain all customers (objects)

**Methods**

\_\_init\_\_()

The **init** method should have an empty list of customers.

**add(customer: Customer)**

• If a customer already exists, raise ValueError with the message "Customer {customer\_name} already exists.".

• Add the customer to the customers list.

**remove(customer\_name: str)**

• If a customer with that name does not exist, raise ValueError with the message "Customer {customer\_name} does not exist.".

• Otherwise, remove the customer from the customers list and return a message "Removed customer: {customer\_name}".

**find(customer\_name: str)**

• Returns a customer (object) with that name.

• If the customer does not exist, returns the message "None".

**7. Class Shop**

In the shop.py file the Shop class should be implemented.

**Structure**

**The class should have the following attributes:**

• product\_repository: ProductRepository – new product repository upon initialization

• customer\_repository: CustomerRepository - new customer repository upon initialization

**Methods**

\_\_init\_\_()

An instance of the Shop class will have product\_repository and customer\_repository.

**deliver(product\_type: str, name: str)**

• Creates a product with the provided type (Drink or Food) and name.

• Then, add the product to the product repository:

◦ If a product already exists, raise ValueError with the message "Product {product\_name} already exists.".

◦ Otherwise, add the product to the products list and return a message "Product {product\_name} successfully added to inventory."

**sell(customer\_name: str, \*\*shopping\_list)**

**shopping\_list contains product name (str) as key argument and quantity (int) as value argument.**

• First, you should check if there is such customer in the repository and if not - create a customer with that name and add him/ her to the customer repository's list.

• Next, you should check if each of the products from the shopping list exists in the product repository. If it exists, check if there is enough quantity of the product. If there is not enough quantity the customer should buy as many as there are in the repository.

• Finally, add the bought products to the customer's products and decrease its quantity in the product's repository. Note: if a customer bought everything available of a specific product, remove the product from the repository and still return the desired message below.

• At the end return a message with the quantity left for each of the bought products in the following format:

"Left quantity of {product\_name}: {product\_quantity}

Left quantity of {product\_name}: {product\_quantity}

Left quantity of {product\_name}: {product\_quantity}…".

**Problem 3. Unit Tests**

***You will be provided with another skeleton for this problem. Import the new skeleton and write tests for the Survivor class. The class will have some methods, fields and one constructor, which are working properly. You are NOT ALLOWED to change any class. Cover the whole class with unit tests to make sure that the class is working as intended. Submit only the tests folder.***