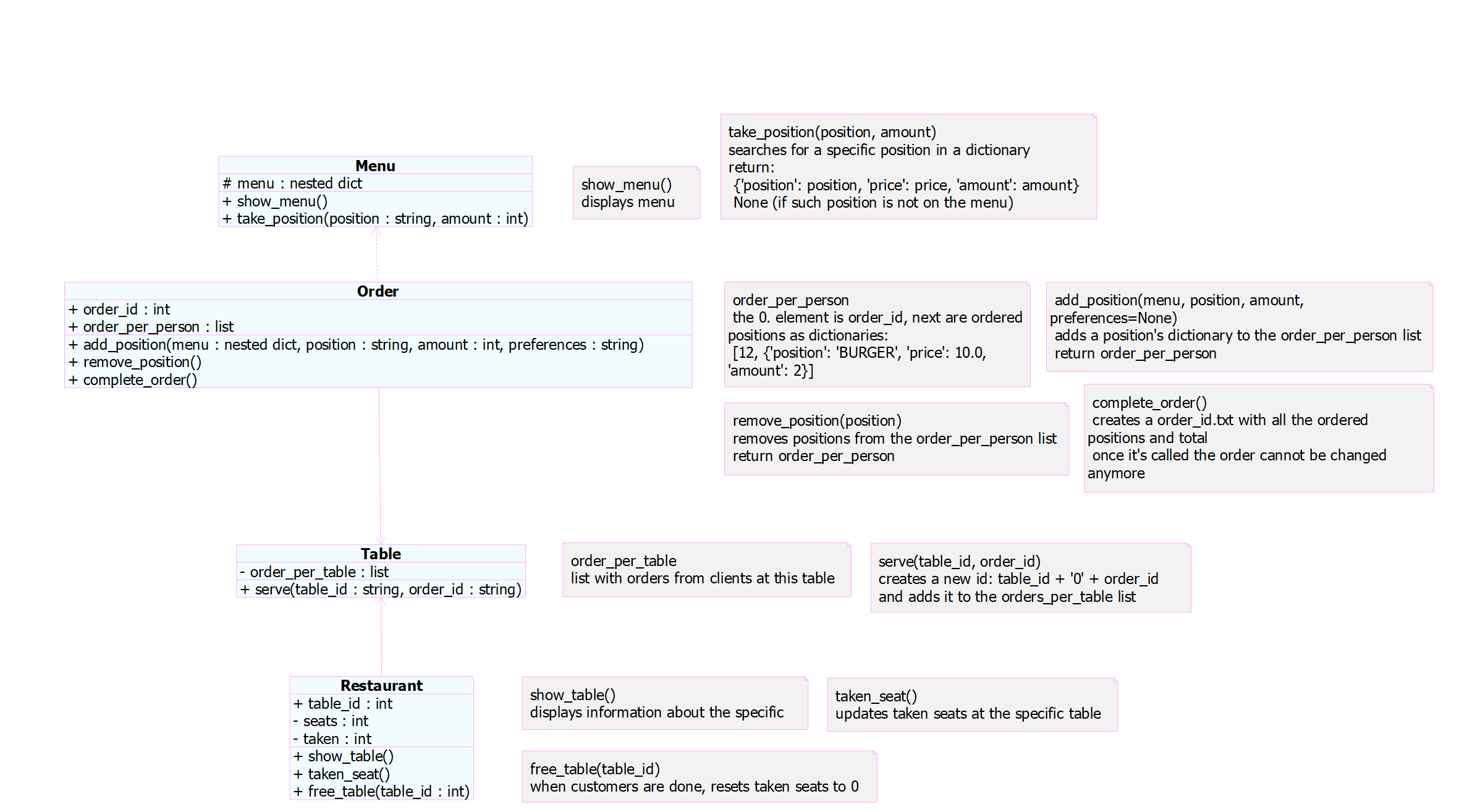
**Documentation**

The following program simulates incomplete functionality of a restaurant app, namely:

* Assigning available in the restaurant tables.
* Seating and vacating of tables.
* Taking orders.
* Reading menu from a excel file.

There are two files included:

1. teamwork.py – contains 4 classes that that simulate the key objects of the restaurant.



# **class Menu**

Using the pandas module automatically reads the excel file with menu (food.xlsx) from the same directory and creates a nested dictionary with the following structure:

{‘categorie’: {‘type’}: {‘position’: float(price)}}.

Its function *show\_menu* allows to print the menu legibly. The second function *take\_position* searches for a specific item in the menu recursively (as it’s a nested dictionary) and returns a dictionary with the structure:

{‘position’: str(), ‘price’: float(), ‘amount’: int()}.

# **class Order**

The class represents an order from one visitor and has 2 attributes: order\_id (unique for every order) and order\_per\_person (list). it turns out that order\_per\_person is a list with the order\_id (0. element) and dictionaries (ordered positions):

[12; {'position': 'BURGER', 'price': 10.0, 'amount': 1}, {'position': 'COLA 0.5', 'price': 3.5, 'amount': 1}].

It supports adding positions to the order, removing them and invoicing. After creating an invoice no further changes can be in the order.

To order food or drinks add\_position runs take\_position (class Menu) and appends the returned dictionary to the order\_per\_person list. Optional are preferences that could be added to a dish. If preferences are specified, they must include ‘no’ (no paprika, no garlic etc.) or ‘extra’ (extra cheese, extra tomatoes etc.). All ‘extra’ preferences increase price of the dish by 1 € and all ‘no’ preferences do not influence the price.

Remove\_position simply removes the position’s dictionary from the order\_per\_person list.

The complete\_order function creates a order\_id.txt file, where all the ordered items are listed together with their prices and quantities as well as the total.

# **class Table**

Tables also usually have their personal numbers, so there is a unique table\_id for every table in the restaurant. Besides that, a table stores orders (once they are completed), made by all customers sitting at this table, in a list orders\_per\_table.

The only function serve basically only combines table\_id with each order\_id through ‘0’, so waiters could bring food and drinks more easily:

301, 302, 303, 304 – orders 1, 2, 3, 4 from the table number 3.

It appends a new extended id and returns the updated order\_per\_table list.

# **class Restaurant**

The class serves to manage tables in general. Each table is assigned with a table\_id (table\_number), number of seats (seats) and number of customers (taken), which are respectively attributes of the class.

The function show\_table displays on the screen the information above, taken\_seat updates the number of customers at the table and free\_table resets to 0 occupied seats.

1. execute.py – console UI that controls performance of functions.