# OOP Basics Exam – SoftUniRestaurant



### Overview

As we all love food, today you were chosen to build a simple restaurant system. This system must have support for **food**, **drinks** and **tables** in the restaurant. The project will consist of **model classes** and a **controller class**, which manages the **interaction** between the **food**, **drinks** and **tables**.

### Setup

Open the provided skeleton. Don’t change the given namespaces. **Not following this rule will lead to your code not compiling in the Judge system**.

**Name your fields and properties exactly how they are given in this document.**

## Task 1: Structure (50 points)

You are given **3** interfaces, which you have to implement **by yourself** and then implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Tables**, **Food** and **Drinks**:

### Food

The Food is a **base class** for any **type of food** and it **should not be able to be instantiated**.

#### Data

* Name – **string (If the name is null or whitespace throw an ArgumentException with message "Name cannot be null or white space!")**
* ServingSize – **int (can’t be less or equal to 0. In these cases, throw an ArgumentException with message "**Serving size cannot be less or equal to zero**")**
* **Price** – **decimal (can’t be less or equal to 0. In these cases, throw an ArgumentException with message "**Price cannot be less or equal to zero!"**)**

#### Behavior

##### string ToString()

Returns a string with information about **each food**. The returned string must be in the following format:

**"{current food name}: {current serving size}g - {current price - formatted to the second digit}"**

#### Constructor

A **food** should take the following values upon initialization:

string name, int servingSize, decimal price

#### Child Classes

There are several concrete types of **food**:

* Dessert – with constant value for InitialServingSize - 200
* Salad - with constant value for InitialServingSize - 300
* Soup - with constant value for InitialServingSize - 245
* MainCourse - with constant value for InitialServingSize - 500

### Drink

The Drink is a **base class** for any **type of drink** and it **should not be able to be instantiated**.

#### Data

* Name – **string (If the name is null or whitespace throw an ArgumenException with message "Name cannot be null or white space!")**
* ServingSize – **int (if the serving size is less than or equal to 0, throw an ArgumentException with message "**Serving size cannot be less or equal to zero**")**
* **Price** – **decimal (if the price is less than or equal to 0, throw an ArgumentException with message "**Price cannot be less or equal to zero**")**
* **Brand - string (If the brand is null or whitespace thrown ArgumentException with message "Brand cannot be null or white space!")**

#### Behavior

##### string ToString()

Returns a string with information about **each drink**. The returned string must be in the following format:

**"{current drink name} {current brand name} - {current serving size}ml - {current price - formatted to the second digit}lv"**

#### Constructor

A **drink** should take the following values upon initialization:

string name, int servingSize, decimal price, string brand

#### Child Classes

There are several concrete types of **drink**, which have **different prices**:

* FuzzyDrink – with constant value for FuzzyDrinkPrice - 2.50
* Juice – with constant value for JuicePrice - 1.80
* Water – with constant value for WaterPrice - 1.50
* Alcohol – with constant value for AlcoholPrice - 3.50

### Table

The Table is a base **class** for different types of tables and **should not be able to be instantiated**.

#### Data

* FoodOrders – collection of foods accessible only by the base class.
* DrinkOrders – collection of drinks accessible only by the base class.
* TableNumber – **int** the table number
* Capacity – **int** the table capacity(capacity can’t be less than zero. In these cases, throw an ArgumentException **with message** "Capacity has to be greater than 0")
* NumberOfPeople – int the count of people who want a table (number of people cannot be less or equal to 0. In these cases, throw an ArgumentException **with message** "Cannot place zero or less people!")
* PricePerPerson – decimal the price per person for the table
* IsReserved – bool returns true if the table is reserved
* Price – calculated property, which calculates the price for all people

#### Constructor

A **table** should take the following values upon initialization:

int tableNumber, int capacity, decimal pricePerPerson

#### Child Classes

There are several concrete types of **tables**, which have **different prices**:

* InsideTable – with constant value for InitialPricePerPerson - 2.50
* OutsideTable – with constant value for InitialPricePerPerson - 3.50

#### Behavior

##### void Reserve(int numberOfPeople)

Reserves the table with the count of people given.

##### void OrderFood(IFood food)

Orders the provided food (think of a way to collect all the food which is ordered).

##### void OrderDrink(IDrink drink)

Orders the provided drink (think of a way to collect all the drinks which are ordered).

##### decimal GetBill()

Returns the bill for all of the ordered drinks and food.

##### void Clear()

Removes all of the ordered drinks and food and finally frees the table and sets the count of people to 0.

##### string GetFreeTableInfo()

Return a string with the following format:

"Table: {table number}"

"Type: {table type}"

"Capacity: {table capacity}"

"Price per Person: {price per person for the current table}"

##### string GetOccupiedTableInfo()

Return a string with the following format:

"Table: {table number}"

"Type: {table type}"

"Number of people: {table number of people}"

If there aren’t any food orders append the following message to the text above:

"Food orders: None"

If there are food orders:

"Food orders: {food orders count}"

Finally append each food ToString() method

The same logic you can use for the ordered drinks. If there aren’t any drink orders just append the message:

"Drink orders: None"

But in the other case:

"Drink orders: {drink orders count}"

Finally append each drink ToString() method. If you got confused just look in the input output examples and you will understand it 😊

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. Implement a class called RestaurantController, which will hold the **main functionality**.

**Note: The** RestaurantController **class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

The main functionality is represented by these **public** **methods**:

|  |
| --- |
| RestaurantController.cs |
| public string AddFood(string type, string name, decimal price)  {  throw new NotImplementedException();  }  public string AddDrink(string type, string name, int servingSize, string brand)  {  throw new NotImplementedException();  }  public string AddTable(string type, int tableNumber, int capacity)  {  throw new NotImplementedException();  }  public string ReserveTable(int numberOfPeople)  {  throw new NotImplementedException();  }  public string OrderFood(int tableNumber, string foodName)  {  throw new NotImplementedException();  }  public string OrderDrink(int tableNumber, string drinkName, string drinkBrand)  {  throw new NotImplementedException();  }  public string LeaveTable(int tableNumber)  {  throw new NotImplementedException();  }  public string GetFreeTablesInfo()  {  throw new NotImplementedException();  }  public string GetOccupiedTablesInfo()  {  throw new NotImplementedException();  }  public string GetSummary()  {  throw new NotImplementedException();  } |

Also, the controller holds all **foods**, **drinks** and **tables**:

* **menu** – List of foods – foods offered by the restaurant
* **drinks** – List of drinks – the drinks the restaurant offers
* **tables** – List of tables – all tables in the restaurant

**NOTE: The** RestaurantController **class should not handle any exceptions. That should be the responsibility of the class, which reads the commands and passes them to the** RestaurantController**.**

### Commands

There are several commands, which control the business logic of the application. They are stated below.

#### AddFood Command

##### Parameters

* Type – string
* Name – string
* Price – decimal

##### Functionality

Creates a food with the correct type. If the food is created successful, returns:

"Added {food name} ({food type}) with price {food price:f2} to the pool"

#### AddDrink Command

##### Parameters

* Type – string
* Name – string
* ServingSize – int
* Brand - string

##### Functionality

Creates a drink with the correct type. If the drink is created successful, returns:

#### "Added {drink name} ({drink brand}) to the drink pool"

#### AddTable Command

##### Parameters

* Type - string
* TableNumber – int
* Capacity - int

##### Functionality

Creates a table with the correct type and returns:

"Added table number {table number} in the restaurant"

#### ReserveTable Command

##### Parameters

* NumberOfPeople – int

##### Functionality

Finds a table which is not reserved, and its capacity is enough for the number of people provided. If there is no such table returns:

"No available table for {numberOfPeople} people"

In the other case reserves the table and returns:

"Table {table number} has been reserved for {numberOfPeople} people"

#### OrderFood Command

##### Parameters

* TableNumber - int
* FoodName - string

##### Functionality

Finds the table with that number and the food with that name in the menu. If there is no such table returns:

"Could not find table with {tableNumber}"

If there is no such food returns:

"No {foodName} in the menu"

In other case orders the food for that table and returns:

"Table {tableNumber} ordered {foodName}"

#### OrderDrink Command

##### Parameters

* TableNumber - int
* DrinkName – string
* DrinkBrand - string

##### Functionality

Finds the table with that number and finds the drink with that name and brand. If there is no such table, it returns:

"Could not find table with {tableNumber}"

If there isn’t such drink, it returns:

"There is no {drinkName} {drinkBrand} available"

In other case, it orders the drink for that table and returns:

**"Table {tableNumber} ordered {drinkName} {drinkBrand}"**

#### LeaveTable Command

##### Parameters

* TableNumber - int

##### Functionality

Finds the table with the same table number. Gets the bill for that table and clears it. Finally returns:

"Table: {tableNumber}"

"Bill: {table bill:f2}"

#### GetFreeTablesInfo Command

##### Functionality

Finds all not reserved tables and for each table returns the table info.

#### GetOccupiedTablesInfo Command

##### Functionality

Finds all reserved tables and for each table returns the table info.

#### GetSummary Command

Returns the total income for the restaurant for all orders.

"Total income: {income:f2}lv"

## Task 3: Input / Output

### Input

* You will receive commands **until you receive “END”** as a command.

Below, you can see the **format** in which **each command** will be given in the input:

* AddFood {type} {name} {price}
* AddDrink {type} {name} {servingSize} {brand}
* AddTable {type} {tableNumber} {capacity}
* ReserveTable {numberOfPeople}
* OrderFood {tableNumber} {foodName}
* OrderDrink {tableNumber} {drinkName} {drinkBrand}
* LeaveTable {tableNumber}
* GetFreeTablesInfo
* GetOccupiedTablesInfo

### Output

Print the output from each command when issued. When the END command is received, print the summary for the restaurant

### Constraints

* The commands will always be in the provided format.

### Examples

|  |
| --- |
| **Input** |
| AddFood Dessert Toffifee 2.90  AddDrink Water Spring 500 Divna  AddTable Inside 1 10  AddTable Outside 2 20  ReserveTable 5  OrderFood 1 Toffifee  OrderDrink 1 Spring Divna  GetOccupiedTablesInfo  GetFreeTablesInfo  LeaveTable 1  END |
| **Output** |
| Added Toffifee (Dessert) with price 2.90 to the pool  Added Spring (Divna) to the drink pool  Added table number 1 in the restaurant  Added table number 2 in the restaurant  Table 1 has been reserved for 5 people  Table 1 ordered Toffifee  Table 1 ordered Spring Divna  Table: 1  Type: InsideTable  Number of people: 5  Food orders: 1  Toffifee: 200g - 2.90  Drink orders: 1  Spring Divna - 500ml - 1.50lv  Table: 2  Type: OutsideTable  Capacity: 20  Price per Person: 3.50  Table: 1  Bill: 16.90  Total income: 16.90lv |

|  |
| --- |
| **Input** |
| AddFood Dessert Toffifee 2.90  AddFood Salad Shopska 12.90  AddFood Soup Bob 12.90  AddFood MainCourse Chushki -90  AddDrink Water Spring -500 Divna  AddDrink Alcohol Rakia 200 YambolskaPerla  AddDrink FuzzyDrink PeachSchnapps 200 Monin  AddTable Inside 1 10  AddTable Inside 2 12  AddTable Inside 3 11  AddTable Outside 4 20  AddTable Outside 5 -2  AddTable Outside 6 10  ReserveTable 5  ReserveTable 1  ReserveTable 2  OrderFood 1 Toffifee  OrderFood 1 Shopska  OrderFood 2 Bob  OrderFood 3 Bob  OrderFood 4 Bob  OrderDrink 1 Spring Divna  OrderDrink 2 Spring Divna  OrderDrink 2 Spring YambolskaPerla  OrderDrink 3 Spring Monin  GetOccupiedTablesInfo  GetFreeTablesInfo  LeaveTable 1  LeaveTable 2  END |
| **Output** |
| Added Toffifee (Dessert) with price 2.90 to the pool  Added Shopska (Salad) with price 12.90 to the pool  Added Bob (Soup) with price 12.90 to the pool  Price cannot be less or equal to zero!  Serving size cannot be less or equal to zero  Added Rakia (YambolskaPerla) to the drink pool  Added PeachSchnapps (Monin) to the drink pool  Added table number 1 in the restaurant  Added table number 2 in the restaurant  Added table number 3 in the restaurant  Added table number 4 in the restaurant  Capacity has to be greater than 0  Added table number 6 in the restaurant  Table 1 has been reserved for 5 people  Table 2 has been reserved for 1 people  Table 3 has been reserved for 2 people  Table 1 ordered Toffifee  Table 1 ordered Shopska  Table 2 ordered Bob  Table 3 ordered Bob  Table 4 ordered Bob  There is no Spring Divna available  There is no Spring Divna available  There is no Spring YambolskaPerla available  There is no Spring Monin available  Table: 1  Type: InsideTable  Number of people: 5  Food orders: 2  Toffifee: 200g - 2.90  Shopska: 300g - 12.90  Drink orders: None  Table: 2  Type: InsideTable  Number of people: 1  Food orders: 1  Bob: 245g - 12.90  Drink orders: None  Table: 3  Type: InsideTable  Number of people: 2  Food orders: 1  Bob: 245g - 12.90  Drink orders: None  Table: 4  Type: OutsideTable  Capacity: 20  Price per Person: 3.50  Table: 6  Type: OutsideTable  Capacity: 10  Price per Person: 3.50  Table: 1  Bill: 28.30  Table: 2  Bill: 15.40  Total income: 43.70lv |

## Task 4: Unit Testing (100 points)

You will receive a skeleton with one class inside. The class will have some methods, properties, fields and constructors. Cover the whole class with unit test to make sure that the class is working as intended.