Java OOP Exam – 15.08.2021

Restaurant "Health"

Overview

Spring came and Alex decided to open a restaurant "Health", for which you need to make a software system. This system must have support for healthy foods, beverages and tables. The project will consist of model classes and a controller class that manages the interaction between healthy foods, beverages and tables.

Setup

- Upload **only the `restaurant`** package in every problem **except Unit Tests**
- **Do not modify the classes, interfaces or their packages**
- Use **strong cohesion** and **loose coupling**
- **Use inheritance and the provided interfaces wherever possible**
    - This includes **constructors**, **method parameters** and **return types**
- **Do not** violate your **interface implementations** by adding **more public methods** in the concrete class than the interface has defined
- Make sure you have **no public fields** anywhere

Task 1: Structure (50 points)

You are given 8 interfaces, and you must implement their functionality in the **correct classes**.

It is not required to implement your structure with **`Engine`**, **`ConsoleReader`**, **`ConsoleWriter`** and enc. It's good practice but it's not required.

There are 3 types of entities and 3 repositories in the application: **`Table`**, **`HealthyFood`**, **`Beverages`** and a **`Repository`** for each of them:

HealthyFood

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

Data

- **`name - String`**
  - If the name is null or whitespace, throw an **`IllegalArgumentException`** with message **`"Name cannot be null or white space!"`**
- **`portion - double`**
  - If the portion is less or equal to 0, throw an **`IllegalArgumentException`** with message **`"Portion cannot be less or equal to zero!"`**
- **`price - double`**
  - If the price is less or equal to 0, throw an **`IllegalArgumentException`** with message **`"Price cannot be less or equal to zero!"`**

Constructor

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

`String name, double portion, double price`

Child Classes

There are several concrete types of **`HealthyFood`**:

Salad

The **Salad has** constant value for **`InitialSaladPortion - 150`**

VeganBiscuits

The **VeganBiscuits has** constant value for **`InitialVeganBiscuitsPortion – 205`**

Beverages

`BaseBeverage` is a **base class** for any **type of `Beverages`** and it **should not be able to be instantiated**.

Data

- **`name - String`**
  - If the name **is null or whitespace**, throw an **`IllegalArgumentException`** with message **`"Name cannot be null or white space!"`**
- **`counter - int`**
  - If the counter is less or equal to 0, throw an **`IllegalArgumentException`** with message **`"Counter cannot be less or equal to zero!"`**
- **`price - double`**
  - If the price is less or equal to 0, throw an **`IllegalArgumentException`** with message **`"Price cannot be less or equal to zero!"`**
- **`brand - String`**
  - If the brand is null or whitespace, throw an **`IllegalArgumentException`** with message **`"Brand cannot be null or white space!"`**

Constructor

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

String name, int counter, double price, String brand

Child Classes

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

Smoothie

The **Smoothie** has **constant value** for **smoothiePrice - 4.50**

Fresh

The **Fresh** has **constant value** for **freshPrice - 3.50**

Table

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

Data

- **healthyFood - Collection<HealthyFood>** accessible only by the **base class**
- **beverages – Collection<Beverages>** accessible only by the **base class**
- **number – int** the table number
- **size - int** the table size
    - It can't be **less than zero**. In these cases, throw an **IllegalArgumentException** with message **"Size has to be greater than 0!".**
- **numberOfPeople - int** the counter of people who want a table
    - It can't be **less** than or **equal to 0**. In these cases, throw an **IllegalArgumentException** with message **"Cannot place zero or less people!".**
- **pricePerPerson - double** the **price per person** for the table
- **isReservedTable - boolean** returns **true** if the **table** is **reserved**, otherwise **false**
- **allPeople - double** calculates the **price for all people**

Behavior

void reserve(int numberOfPeople)

Reserves the table with the counter of people given.

void orderHealthy(HealthyFood food)

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

void orderBeverages(Beverages beverages)

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

`double bill()`

Returns the bill for all orders.

`void clear()`

Removes all the ordered drinks and food and finally frees the table, the table is not reserved, sets the count of people and price to 0.

`String tableInformation()`

Return a String with the following format:

```
"Table - {table number}"
"Size - {table size}"
"Type - {table type}"
"All price - {price per person for the current table}"
```

Constructor

A **`BaseTable`** should take the following values upon initialization:

```
int number, int size, double pricePerPerson
```

Child Classes

There are several concrete types of **`Table`**:

InGarden

The **InGarden** table has **constant value** for **`pricePerPerson - 4.50`**

Indoors

The **Indoors** table has **constant value** for **`pricePerPerson - 3.50`**

Repository

The repository holds information about the entity.

Data

- **`entities`** - A **collection of T**

Behavior

`void add(T entity)`

Adds an entity in the collection.

`Collection<T> getAllEntites()`

Returns all entities (unmodifiable).

Child Repositories

TableRepository

`T byNumber(int tableNumber)`

Returns an entity with that name.

HealthFoodRepository

`T foodByName(String name)`

Returns an entity with that name.

BeveragesRepository

`T beverageByName(String name)`

Returns an entity with that name.

Child Classes

Create **`TableRepositoryImpl`**, **`HealthFoodRepositoryImpl`** and **`BeveragesRepositoryImpl`** repositories.

Task 2: Business Logic (150 points)

The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you must implement in the correct classes.

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

The first interface is **`Controller`**. You must implement a **`ControllerImpl`** class, which implements the interface and implements all its methods. The given methods should have the following logic:

Commands

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

addHealthyFood Command

Important! **Firstly create the corresponding object, if possible, and then check if it exists in the records.**

Parameters

- **`type - String`**
- **`price - double`**
- **`name - String`**

Functionality

Creates a food with the correct type. If the food is created successfully **add it to the food repository** and returns:

`"Added {name} to the healthy menu!"`

If a healthy food with the given name already exists in the food repository, throw an **`IllegalArgumentException`** with message **`"{name} is already in the healthy menu!"`**

addBeverage Command

Important! Firstly create the corresponding object, if possible, and then check if it exists in the records.

Parameters

- **`type - String`**
- **`counter - int`**
- **`brand - String`**
- **`name - String`**

Functionality

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

`"Added {type} - {brand} to the beverage menu!"`

If a beverage with the given name already exists in the beverage repository, throw an **`IllegalArgumentException`** with message **`"{name} is already in the beverage menu!"`**

addTable Command

Important! Firstly create the corresponding object, if possible, and then check if it exists in the records.

Parameters

- **`type - String`**
- **`tableNumber - int`**
- **`capacity - int`**

Functionality

Creates a table with the correct type and returns:

`"Added table number {number} in the healthy restaurant!"`

If a table with the given name already exists in the table repository, throw an **`IllegalArgumentException`** with message **`"Table {number} is already added to the healthy restaurant!"`**

reserve Command

Parameters

- **numberOfPeople - int**

Functionality

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

"There is no such table for {numberOfPeople} people."

In the other case reserves the table and returns:

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

orderHealthyFood Command

Parameters

- **tableNumber - int**
- **healthyFoodName - String**

Functionality

Finds the table with that number and the food with that name in the menu. **You first check if the table exists.** If there is no such table returns:

"Could not find table {tableNumber}."

If there is no such food returns:

"No {healthyFoodName} in the healthy menu."

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

"{healthyFoodName} ordered from table {tableNumber}."

orderBeverage Command

Parameters

- **tableNumber - int**
- **name - String**
- **brand - String**

Functionality

Finds the table with that number and finds the beverage with that name and brand. **You first check if the table exists.** If there is no such table, it returns:

"Could not find table {tableNumber}."

If there isn't such beverage, it returns:

"No {name} - {brand} in the beverage menu."

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

"{name} ordered from table {tableNumber}."

closedBill 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} with bill: {table bill formatted to the second digit}."

totalMoney Command

Returns the money earned for the restaurant for all completed bills.

"Total money for the restaurant: {money formatted to the second digit}lv."

Input / Output

You are provided with one interface, which will help with the correct execution process of your program. The interface is **Engine** and the class implementing this interface should read the input and when the program finishes, this class should print the output.

Input

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

- **addHealthyFood {type} {price} {name}**
- **addBeverage {type} {counter} {brand} {name}**
- **addTable {type} {number} {capacity}**
- **reserve {numberOfPeople}**
- **orderHealthyFood {tableNumber} {healthyFoodName}**
- **orderBeverage {tableNumber} {name} {brand}**
- **closedBill {tableNumber}**
- **totalMoney**
- **END**

Output

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

Examples

Input
addHealthyFood Salad 2.90 Greece

```
addBeverage Smoothie 3 Summer Sugar
addTable Indoors 1 10
addTable InGarden 2 20
reserve 5
orderHealthyFood 1 Burger
orderBeverage 1 Cola Coca-Cola
closedBill 1
totalMoney
END
```

Output

```
Added Greece to the healthy menu!
Added Smoothie - Summer to the beverage menu!
Added table number 1 in the healthy restaurant!
Added table number 2 in the healthy restaurant!
Table 1 has been reserved for 5 people.
No Burger in the healthy menu.
No Cola – Coca-Cola in the beverage menu.
Table: 1 with bill: 17.50
Total money for the restaurant: 17.50lv.
```

Input

```
addHealthyFood Salad 2.90 Greece
addHealthyFood VeganBiscuits 5.90 Abi
addHealthyFood Salad 4.90 Caezar
addHealthyFood VeganBiscuits -5.40 Lemonsy
addBeverage Smoothie 3 Summer Sugar
addBeverage Fresh 5 Orange original
addBeverage Fresh -22 Strawberry herbal
addBeverage Fresh 7 Natural original
```


```
addTable Indoors 1 12
addTable Indoors 4 3
addTable Indoors 2 2
addTable InGarden 2 -20
addTable InGarden 4 3
addTable InGarden 5 12
reserve 5
reserve 1
reserve 4
orderHealthyFood 1 Pasta
orderHealthyFood 2 Pizza
orderHealthyFood 3 VeganBuiscuits
orderHealthyFood 5 Invalid
orderBeverage 1 Fresh Natural
orderBeverage 4 Frape Sugar
orderBeverage 5 Juice Strawberry
orderBeverage 2 Fresh Orange
closedBill 1
closedBill 5
totalMoney
END
```

Output

```
Added Greece to the healthy menu!
Added Abi to the healthy menu!
Added Caezar to the healthy menu!
Price cannot be less or equal to zero!
Added Smoothie - Summer to the beverage menu!
Added Fresh - Orange to the beverage menu!
Counter cannot be less or equal to zero!
Added Fresh - Natural to the beverage menu!
```

```
Added table number 1 in the healthy restaurant!
Added table number 4 in the healthy restaurant!
Added table number 2 in the healthy restaurant!
Size has to be greater than 0!
Table 4 is already added to the healthy restaurant!
Added table number 5 in the healthy restaurant!
Table 1 has been reserved for 5 people.
Table 4 has been reserved for 1 people.
Table 5 has been reserved for 4 people.
No Pasta in the healthy menu.
No Pizza in the healthy menu.
Could not find table 3.
No Invalid in the healthy menu.
No Fresh - Natural in the beverage menu.
No Frape - Sugar in the beverage menu.
No Juice - Strawberry in the beverage menu.
No Fresh - Orange in the beverage menu.
Table: 1 with bill: 17,50
Table: 5 with bill: 18,00
Total money for the restaurant: 35,50lv.
```

Task 3: Unit Tests (100 points)

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