# Java OOP Exam – 10 December 2022

# Christmas Pastry Shop



## Overview

As we all love delicacies, today you were chosen to build a simple Christmas pastry shop software system. This system must have support for **delicacies**, **cocktails,** and **booths** in the pastry shop. The project will consist of **model classes** and a **controller class**, which manages the **interaction** between the **delicacies**, **cocktails,** and **booths**.

## Setup

* Upload **only the** christmasPastryShoppackage in every task **except** **Unit Tests.**
* **Do not modify the 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: **Booth, Delicacy, Cocktail, and a Repository for each of them**:

### Delicacy

BaseDelicacy is a **base class** for any **type of Delicacy,** and it **should not be able to be instantiated**.

#### Data

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

#### Behavior

##### String toString()

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

"{current delicacy name}: {current portion - formatted to the second digit}g - {current price - formatted to the second digit}"

#### Constructor

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

String name, double portion, double price

#### Child Classes

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

##### Gingerbread

The **Gingerbread has a** constant value for InitialGingerbreadPortion – 200.

##### Stolen

The **Stolen has a** constant value for InitialStolenPortion – 250.

### Cocktail

BaseCocktail is a **base class** for any **type of Cocktail** and it **should not be able to be instantiated**.

#### Data

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

#### Behavior

##### String toString()

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

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

#### Constructor

A BaseCocktailshould take the following values upon initialization:

String name, int size, double price, String brand

#### Child Classes

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

##### MulledWine

The **MulledWine has а constant value** formulledWinePrice – 3.50**.**

##### Hibernation

The **Hibernation** has а **constant value** forhibernationPrice – 4.50**.**

### Booth

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

#### Data

* **delicacyOrders** – **Collection<Delicacy>**
* **cocktailOrders** – **Collection<Cocktail>**
* **boothNumber** – int the booth number
* **capacity** – int the booth capacity.
  + It can’t be **less than zero**. In these cases, throw an **IllegalArgumentException** with themessage "Capacity has to be greater than 0!"
* **numberOfPeople** – int the count of people who want a booth.
  + Cannot be **less** 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 booth.
* **isReserved** – **boolean** returns **true** if the **booth** is **reserved**, otherwise **false.**
* **price** – **double** calculates the price for all people.

#### Behavior

##### void reserve(int numberOfPeople)

Reserve the booth with the count of people given and calculate the price of the booth.

##### double getBill()

Returns the bill for the booth, all of the ordered cocktails and delicacies.

##### void clear()

Removes all the ordered cocktails and delicacies and finally frees the booth, sets the count of people and price to 0.

#### Constructor

A BaseBoothshould take the following values upon initialization:

int boothNumber, int capacity, double pricePerPerson

#### Child Classes

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

##### OpenBooth

The **OpenBooth has a constant value** for pricePerPerson – 2.50.

##### PrivateBooth

The **PrivateBooth** has a **constant value** forpricePerPerson – 3.50**.**

### Repository

The repository holds information about the entity.

#### Data

* models - A **collection of T (entity)**

#### Behavior

**void add(T model)**

Adds an entity in the collection.

**Collection<T> getAll()**

Returns all entities (unmodifiable)

#### Child Repositories

##### BoothRepository

**T getByNumber(int boothNumber)**

Returns an entity with that name.

##### DelicacyRepository

**T getByName(String name)**

Returns an entity with that name.

##### CocktailRepository

**T getByName(String name)**

#### Child Classes

Create **DelicacyRepositoryImpl**, **CocktailRepositoryImpl** and **BoothRepositoryImpl** 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 that you must implement in the correct classes.

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

The first interface is Controller. You must create a ControllerImplclass, 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**. The **Field** **name** passed to the methods will **always** be **valid**!

#### AddDelicacy Command

##### Parameters

* type – String
* name – String
* price – double

##### Functionality

Creates a delicacy with the correct type. If the delicacy is created successfully, returns:

"Added delicacy {delicacy name} - {delicacy type} to the pastry shop!"

**If a delicacy with the given name already exists in the delicacy repository, throw** an IllegalArgumentException **with the message** "{type} {name} is already in the pastry shop!"

#### AddCocktail Command

##### Parameters

* type – String
* name – String
* portion – int
* brand – String

##### Functionality

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

#### "Added cocktail {cocktailName} - {cocktailBrand} to the pastry shop!”

**If a cocktail with the given name already exists in the cocktail repository, throw** an IllegalArgumentException **with the message** "{type} {name} is already in the pastry shop!"

#### AddBooth Command

##### Parameters

* type – String
* boothNumer – int
* capacity – int

##### Functionality

Creates a booth with the correct type and returns:

"Added booth number {boothNumber} in the pastry shop!"

**If a booth with the given name already exists in the booth repository, throw an **IllegalArgumentException** with the message **"****Booth {boothNumber} is already added to the pastry shop!"

#### ReserveBooth Command

##### Parameters

* numberOfPeople – int

##### Functionality

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

"No available booth for {numberOfPeople} people!"

In the other case reserves the booth and return:

"Booth {boothNumber} has been reserved for {numberOfPeople} people!"

#### LeaveBooth Command

##### Parameters

* boothNumber – int

##### Functionality

Finds the booth with the same booth number. Gets the bill for that booth, clears it and adds the sum to the total store income. The bill is not only the orders, but the reservation for the number of people as well. Finally returns:

"Booth: {boothNumber}"

"Bill: {booth bill:f2}"

#### GetIncome Command

Returns the total income for the pastry shop for **all completed bills**.

"Income: {income:f2}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:

* **AddDelicacy {type} {name} {price}**
* **AddCocktail {type} {name} {size} {brand}**
* **AddBooth {type} {boothNumber} {capacity}**
* **ReserveBooth {numberOfPeople}**
* **LeaveBooth {boothNumber}**
* **GetIncome**
* **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** |
| AddDelicacy Gingerbread Christ 3.90  AddCocktail MulledWine Winter 800 Monin  AddBooth OpenBooth 1 10  AddBooth PrivateBooth 2 20  ReserveBooth 5  LeaveBooth 1  GetIncome  END |
| **Output** |
| Added delicacy Christ - Gingerbread to the pastry shop!  Added cocktail Winter - Monin to the pastry shop!  Added booth number 1 in the pastry shop!  Added booth number 2 in the pastry shop!  Booth 1 has been reserved for 5 people!  Booth: 1  Bill: 12,50  Income: 12,50lv |

|  |
| --- |
| **Input** |
| AddDelicacy Gingerbread Healthy 2.90  AddDelicacy Stolen Choco 5.90  AddDelicacy Stolen Cherry -9.0  AddCocktail MulledWine Winter -300 Monin  AddCocktail MulledWine Autumn 500 Red  AddCocktail Hibernation Strong 250 Lipton  AddCocktail Hibernation Medium 200 Bio  AddBooth OpenBooth 1 10  AddBooth OpenBooth 2 12  AddBooth PrivateBooth 3 20  AddBooth PrivateBooth 4 -2  ReserveBooth 3  ReserveBooth 2  LeaveBooth 1  LeaveBooth 2  GetIncome  END |
| **Output** |
| Added delicacy Healthy - Gingerbread to the pastry shop!  Added delicacy Choco - Stolen to the pastry shop!  Price cannot be less or equal to zero!  Size cannot be less or equal to zero!  Added cocktail Autumn - Red to the pastry shop!  Added cocktail Strong - Lipton to the pastry shop!  Added cocktail Medium - Bio to the pastry shop!  Added booth number 1 in the pastry shop!  Added booth number 2 in the pastry shop!  Added booth number 3 in the pastry shop!  Capacity has to be greater than 0!  Booth 1 has been reserved for 3 people!  Booth 2 has been reserved for 2 people!  Booth: 1  Bill: 7,50  Booth: 2  Bill: 5,00  Income: 12,50lv |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **Toy,** and **ToyStore**. **ToyStore** class will have some methods, fields, and constructors. Cover the whole class with the unit test to make sure that the class is working as intended. In Judge, you upload **.zip** to **toyStore (**with **ToyStoreTests** inside**)** from the **skeleton**.