# Store System

### Преглед на задачата

Като Junior C# разработчици получавате първата си задача във фирмата. Трябва да разработите система за магазини, които трябва да имат функционалността да зареждат и да продават продуктите си в основни линии. Трябва да приложите знанията си по ООП, за да завършите тази задача. Успех!!!

**Забележка**: Преди да започнете със задачата, за да нямате проблеми със системата премахнете **namespaces** от всеки клас.

## Problem 1. Структура

### Product

Продукта е **базов** **клас** за всички видове продукти и **не** **трябва** да може да се **инстанцира**.

#### Data

Всеки продукт има име, количество, доставна цена, процентна надценка и крайна цена.

**Name** – символен низ

**Quantity** – цяло число

**DeliverPrice** – реално число

**PercentageMarkup** – реално число

**FinalPrice** – реално число

* Името на продукта **не** **трябва** да бъде **null** или **празен** **стринг.**

**Съобщение: Product name** **must not be null or empty!**

* Количеството на продукта **не** **може** да бъде **отрицателно** или равно на **0.**

**Съобщение:** Quantity cannot be less or equal to 0!

* Доставната цена на продукта **не може** да бъдат **отрицателна** или равна на **0.**

**Съобщение: Deliver price** cannot be less or equal to 0!

* Процентната надценката на продукта **не може** да бъде **отрицателна** или равна на **0.**

**Съобщение: Percentage markup** cannot be less or equal to 0!

* При подаване на **невалидна** стойност хвърлете изключение – **ArgumentException с конкретното съобщение**.

**Забележка**: Крайната цена се пресмята и трябва да има само (get method). Пресмята се по следната формула:

**{Deliver price} + ({Deliver price} \* {Percentage markup} / 100)**

#### Constructor

Това са параметрите за конструктора:

string name, int quantity, double deliverPrice, double percentageMarkup

#### Behavior

##### override string ToString()

**\*\*\*\*Product: {Name} <{Quantity}>**

**\*\*\*\*Deliver Price: {DeliverPrice}**

**\*\*\*\*Percentage Markup: {PercentageMarkup}**

**\*\*\*\*Final Price: {FinalPrice}**

#### Child Classes

##### Food

Процента надценка може да бъде **най**-**много** **100**%.

При невалидна стойност хвърлете **ArgumentException**.

**Съобщение: Foob percentage markup cannot be above 100%!**

**Забележка**: Първо изпълнете валидацията от базовия клас след това текущата.

##### Drink

Процента надценка може да бъде **най**-**много** **200**%.

При невалидна стойност хвърлете **ArgumentException**.

**Съобщение: Drink percentage markup cannot be above 200%!**

**Забележка**: Първо изпълнете валидацията от базовия клас след това текущата.

### Store

#### Data

Всеки магазин има име, тип, оборот и **поле** със налични продукти:

Name – символен низ

**Type** – символен низ

**Revenue** – реално число

* Името на магазина **не** **трябва** да бъде **null** или **празен** **стринг.**

**Съобщение: Store name** **must not be null or empty!**

* Типът на магазина **не** **трябва** да бъде **null** или **празен** **стринг.**

**Съобщение: Store type** **must not be null or empty!**

* При подаване на **невалидна** стойност хвърлете изключение – **ArgumentException с конкретното съобщение**.

**Забележка**: Не правете **СВОЙСТВО** за списъка. **Ще бъде проверен!!!**

#### Constructor

Това са параметрите за конструктора:

string name, string type

#### Behavior

##### bool ReceiveProduct(Product product)

В метода трябва да проверите дали този продукт е наличен. Ако вече съществува продукт с това име, магазина счита, че не е необходимо да зарежда още стока и не приема продукта(false). В противен случай го добавете в колекцията с налични продукти(true).

##### bool SellProduct(string name, int quantity)

В метода трябва да проверите дали този продукт с това име е наличен. Ако съществува и има достатъчно количество, намалявате количеството на продукта. Ако количеството на продукта стане 0 го премахнете. След това увеличавате оборота на магазина със стойността на сметката:

**{quantity} \* {product final price}**

##### Product GetProduct(string name)

В метода трябва да проверите дали продукт с това име съществува. Ако съществува връщате обекта. В противен случай върнете **NULL**.

##### override string ToString()

**\*\*\*\*Store: {Name} <{Type}>**

**\*\*\*\*Available products: <{availableProducts count}>**

**\*\*\*\*\*\* {availableProduct1 Name} ({availableProduct1 Quantity})**

**\*\*\*\*\*\* {availableProduct2 Name} ({availableProduct2 Quantity})**

**…**

**\*\*\*\*\*\* {availableProductN Name} ({availableProductN Quantity})**

**\*\*\*\*Revenue: <{Revenue}BGN>**

**Забележка**: Оборота трябва да се закръгли до втория знак след десетичната запетая.

## Problem 2. Бизнес логика

### The Controller Class

Бизнес логиката на програмата трябва да бъде концентрирана около няколко команди. Внедрете клас, наречен **StoreController**, който ще притежава главната функционалност, представена от тези публични методи:

|  |
| --- |
| StoreController.cs |
| public string CreateStore(List<string> args)  {  //TODO: Add some logic here …  }    public string ReceiveProduct(List<string> args)  {  //TODO: Add some logic here …  }    public string SellProduct(List<string> args)  {  //TODO: Add some logic here …  }    public string StoreInfo(List<string> args)  {  //TODO: Add some logic here …  }    public string Shutdown()  {  //TODO: Add some logic here …  } |

**ЗАБЕЛЕЖКА: Не трябва да променяте нищо по методите. Трябва да имплементирате логиката на самите методи. Не прихвайщайте никакви изключения!**

### Команди

Има няколко команди, които контролират бизнес логиката на приложението и трябва да ги имплементирате.

Те са посочени по-долу.

#### CreateStore Команда

Създава магазин и го регистрира в системата.

**Не всички данни ще бъдат валидни!!!**

##### Параметри

* name – символен низ
* type – символен низ

#### ReceiveProduct Команда

Създава продукт и го добавя в наличните продукти на дадения магазин.

**Не всички данни ще бъдат валидни!!!**

##### Параметри

* **type** – символен низ
* name – символен низ
* **quantity** – цяло число
* **price** – реално число
* **percentageMarkup** – реално число
* **storeName** – символен низ

#### SellProduct Команда

Купувате продуктите от дадения магазин.

**Не всички данни ще бъдат валидни!!!**

##### Параметри

* name – символен низ
* **quantity** – цяло число
* **storeName** – символен низ

#### StoreInfo Команда

При тази команда се връща състоянието на магазина във формат описан долу в секцията за вход и изход.

**Не всички данни ще бъдат валидни!!!**

##### Параметри

* **storeName** – символен низ

#### Shutdown Команда

Тази команда прекратява изпълнението на програмата и връща всички магазини, подредени по оборот в низходящ ред и след това по име в нарастващ ред.

## Problem 3. Вход / Изход

Преди да продължите с тази секция от изпита, трябва да разясним, че Startup и Engine класовете не се тестват по никакъв начин. Задачата ви е да довършите приложението, така че да заработи. Няма правила какво трябва да бъде изписано в изброените по-горе класове.

### Вход

* Четете редове с различни команди, докато не получите команда за приключване на програмата.

По-долу можете да видите формата, в който всяка команда ще бъде дадена във входа:

* CreateStore:{name}:{type}
* ReceiveProduct:{type}:{name}:{quantity}:{price}:{percentageMarkup}:{storeName}
* SellProduct:{name}:{quantity}:{storeName}
* **StoreInfo:{storeName}**
  + Shutdown

### Изход

По – долу може да видите кой изход трябва да бъде предоставен от командите.

#### CreateStore Команда

При успешно регистриране върнете:

**Съобщение**: **Store {store name} was successfully registerd in the system!**

В случай, че магазин с това име вече съществува, просто я пропуснете и върнете:

**Съобщение**: **Store {store name} is already registered!**

#### ReceiveProduct Команда

При успешно зареждане на магазина върнете:

**Съобщение**: **Product {product name} was successfully delivered to {store name}!**

В случай, че магазина не съществува върнете:

**Съобщение**: **Invalid Store: {store name}. Cannot find it in system!**

В случай, че типът на продукта е невалиден върнете:

**Съобщение**: **Product {product type} is invalid!**

В случай, че не сме успели да заредим магазина върнете:

**Съобщение**: **Product {product name} was already delivered to {store name}!**

#### SellProduct

При успешно закупуване на продукта върнете:

**Съобщение**: **Product {product name} was successfully bought from {store name}!**

В случай, че магазина не съществува върнете:

**Съобщение**: **Invalid Store: {store name}. Cannot find it in system!**

В случай, че не сме успели да закупим магазина върнете:

**Съобщение**: **Product {product name} was already sold out!**

#### StoreInfo

При успешно намиране на компания върнете информация за нея във формата:

**\*\*\*\*Store: {Name} <{Type}>**

**\*\*\*\*Available products: <{availableProducts count}>**

**\*\*\*\*\*\* {availableProduct1 Name} ({availableProduct1 Quantity})**

**\*\*\*\*\*\* {availableProduct2 Name} ({availableProduct2 Quantity})**

**…**

**\*\*\*\*\*\* {availableProductN Name} ({availableProductN Quantity})**

**\*\*\*\*Revenue: <{Revenue}BGN>**

В случай, че магазин не съществува върнете:

**Съобщение**: **Invalid Store: {store name}. Cannot find it in system!**

#### Shutdown

След завършване на командата трябва да върнете информация за всички магазини в системата подредени по оборот в низходящ ред и след това по име в нарастващ ред във формата, описан в **StoreInfo командата. Преди това добавете следния ред:**

**Stores: {stores count}**

**Накрая добавете следния ред:**

**System Store Revenues: {store revenue sum}BGN**

**Забележка**: Оборота трябва да се закръгли до втория знак след десетичната запетая.

### Ограничения

* Имената на магазините и продуктите ще бъдат символни низове, които може да съдържат всеки ASCII символ, с изключение на двоеточие(U,{721fcb02-321b-457b-bbf4-507934782836}{13},3.125,3.125.
* Винаги ще получавате команда за приключване на програмата.
* Входните данни ще бъдат валидни от страна на типове данни.

### Примери

|  |  |
| --- | --- |
| **Input** | **Output** |
| CreateStore:FreshStore:FoodStore  ReceiveProduct:Food:Chicken breasts:10:3.29:50:FreshStore  SellProduct:Chicken breasts:5:FreshStore  SellProduct:Chicken breasts:3:FreshStore  StoreInfo:FreshStore  Shutdown | Store FreshStore was successfully registerd in the system!  Product Chicken breasts was successfully delivered to FreshStore!  Product Chicken breasts was successfully bought from FreshStore!  Product Chicken breasts was successfully bought from FreshStore!  \*\*\*\*Store: FreshStore <FoodStore>  \*\*\*\*Available products: <1>  \*\*\*\*\*\* Chicken breasts (2)  \*\*\*\*Revenue: <39.48BGN>  Stores: 1  \*\*\*\*Store: FreshStore <FoodStore>  \*\*\*\*Available products: <1>  \*\*\*\*\*\* Chicken breasts (2)  \*\*\*\*Revenue: <39.48BGN>  System Store Revenues: 39.48BGN |
| CreateStore::FoodStore  CreateStore:FreshStore:FoodStore  ReceiveProduct:Food::10:3.29:50:FreshStore  ReceiveProduct:Food:Chicken breasts:-10:3.29:50:FreshStore  ReceiveProduct:Food:Chicken breasts:10:3.29:50:FreshStore  SellProduct:Chicken wings:5:FreshStore  SellProduct:Chicken breasts:3:FreshStore  StoreInfo:InvalidStore  Shutdown | Store name must not be null or empty!  Store FreshStore was successfully registerd in the system!  Product name must not be null or empty!  Quantity cannot be less or equal to 0!  Product Chicken breasts was successfully delivered to FreshStore!  Product Chicken wings was already sold out!  Product Chicken breasts was successfully bought from FreshStore!  Invalid Store: InvalidStore. Cannot find it in system!  Stores: 1  \*\*\*\*Store: FreshStore <FoodStore>  \*\*\*\*Available products: <1>  \*\*\*\*\*\* Chicken breasts (7)  \*\*\*\*Revenue: <14.81BGN>  System Store Revenues: 14.81BGN |