**ShoppingSystem - Structure**

**Avaliable from**: Sunday, 31 October 2021, 10:00  
**Due date**: Sunday, 31 October 2021, 16:00  
**Requested files**: StartUp.cs, Engine.cs, Product.cs, PhysicalProduct.cs, ServiceProduct.cs, Receipt.cs, Controller.cs ([Download](https://it-kariera.mon.bg/e-learning/mod/vpl/views/downloadrequiredfiles.php?id=6129))  
**Type of work**: Individual work

**ShoppingSystem**

**Преглед на задачата**

Като Junior C# разработчици получавате първата си  обучителна задача във фирмата. Трябва да създадете програма, която виртуализира търговията в един мол. Поради сегашните обстоятелства, системата работи само с 1 човек в даден момент. Този човек може да си закупи физически продукт или да плати за услуга. Всичките неща, които той е закупил или заплатил в сесията си в мола, ще бъдат регистрирани, на изхода, като виртуална фактура. Успех!!!

**Забележка**: Класовете Product, PhysicalProduct, ServiceProduct и Receipt поставете в namespace ShoppingSystem. За класовете Controller, Engine и StartUp НЕ посочвайте namespace. Следвайте структурата посочена във файловете от скелета.

* **Структура**

**Product**

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

**Data**

Всеки продукт има име и цена.

**Name** – символен низ

**Price**– реално число

* Името на продукта **трябва** да бъде между 3 и 30 символа включително**.**

**Съобщение: Name should be between 3 and 30 characters!**

* Цената на продукта **не** **може** да бъдат **отрицателна.**

**Съобщение: Price should be 0 or positive!**

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

**Constructor**

Параметрите за конструктора са както следва:

string name, double price

**Behavior**

**override string ToString()**

**Name: {Name}**

**Price: {Price}**

**Child Classes**

**PhysicalProduct**

**Data**

Всеки физически продукт има име, цена и тегло.

**Name** – символен низ

**Price**– реално число

**Weight** – реално число

* Теглото **не** **може** да бъде **отрицателна или 0.**

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

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

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

**Constructor**

Параметрите за конструктора са както следва:

string name, double price, double weight

**Behavior**

**override string ToString()**

**Name: {Name}**

**Price: {Price}**

**Weight: {Weight}**

**ServiceProduct**

**Data**

Всеки продукт от тип услуга има име, цена и време за изпълнение.

**Name** – символен низ

**Price**– реално число

**Time** – реално число

* Времето **не** **може** да бъде **отрицателна или 0.**

**Съобщение: Time should be greater than 0!**

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

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

**Constructor**

Параметрите за конструктора са както следва:

string name, double price, double time

**Behavior**

**override string ToString()**

**Name: {Name}**

**Price: {Price}**

**Time: {Time}**

**Receipt**

**Data**

Всяка виртуална фактура има име на клиента и **поле** със **закупени продукти (колекция от закупени продукти, физически и услуги)**:

**CustomerName**– символен низ

* Името **трябва** да бъде **между 2 и 40 символа включително.**

**Съобщение: Customer Name should be between 2 and 40 characters!**

**Забележка**: Не правете **СВОЙСТВО**за списъка.

**Constructor**

Параметрите за конструктора са както следва:

string customerName

**Behavior**

**void AddProduct(Product product)**

В метода трябва да добавите новия продукт към фактурата.

Виртуалната фактура трябва също така да визуализира сумираната цена на всичките продукти.

**override string ToString()**

**Receipt of {CustomerName}**

**Total Price: {sumOfProductPrices}**

**Products:**

**{product1.ToString()}**

**{product2.ToString()}**

**…**

**{productN.ToString()}**

* **Бизнес логика**

**The Controller Class**

Бизнес логиката на програмата трябва да бъде концентрирана около няколко команди. Бизнес логиката се имплементира в клас **Controller**, който ще притежава главната функционалност, представена от тези публични методи:

|  |
| --- |
| **Controller.cs** |
| public string ProcessProductCommand(List<string> args)  {      //TODO: Add some logic here …  }    public string ProcessServiceCommand(List<string> args)  {      //TODO: Add some logic here …  }    public string ProcessCheckoutCommand(List<string> args)  {      //TODO: Add some logic here …  }    public string ProcessInfoCommand(List<string> args)  {      //TODO: Add some logic here …  }    public string ProcessEndCommand()  {      //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.

**Product Команда**

Създава **PhysicalProduct**и го добавя към системата (към сегашния клиент).

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

**Параметри**

* **name** – символен низ (винаги ще бъде уникално)
* **price** – реално число
* **weight** – реално число

**Service Команда**

Създава **ServiceProduct**и го добавя към системата (към сегашния клиент).

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

**Параметри**

* **name** – символен низ (винаги ще бъде уникално)
* **price** – реално число
* **time** – реално число

**Checkout Команда**

Тук трябва да обслужим сегашния клиент и да приемем следващия.

Тази команда създава **Receipt** и го добавя към системата.

* Receipt-а се създава с подадения **customerName** и с**продуктите**(**PhysicalProduct**) и **услугите**(**ServiceProduct**) на сегашния клиент (**продуктите** и **услугите** добавени към момента).
* След това всички продукти (**PhysicalProduct**) и услуги (**ServiceProduct**) се премахват от системата (и се пазят само в **Receipt**-а и вече всеки следващ продукт (**PhysicalProduct**) или услуга (**ServiceProduct**) ще бъде към нов клиент.

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

**Параметри**

* **customerName** – символен низ

**Info Команда**

Ако **info** параметъра се равнява на "**Customer**", трябва да се изведе информация за сегашния клиент - броя на **продуктите** (**PhysicalProduct**) и **услугите**(**ServiceProduct**), които е поръчал до момента и общата им сумирана цена.

Ако **info** параметъра се равнява на "**Shop**", трябва да се изведе информация за целия мол – всичките **Receipts**.

**Параметри**

* **info** – символен низ

**End Команда**

Тази команда прекратява изпълнението на програмата и връща **общия брой** на клиентите, които са заплатили за своите **продукти** или **услуги**.

**Забележка: Уверете се, че за всеки от класовете, методите и свойствата задавате правилен модификатор за достъп.**

* **Вход / Изход**

**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:

* **Product {name} {price} {weight}**
* **Service {name} {price} {time}**
* **Checkout {customerName}**
* **Info {info}**
* **End**

**Изход**

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

**Product Команда**

При успешно добавяне върнете:

**Съобщение**: **The current customer has bought {productName}.**

**Service Команда**

При успешно добавяне върнете:

**Съобщение**: **The current customer has applied for {serviceName} service.**

**Checkout Команда**

При успешно добавяне върнете:

**Съобщение**: **Customer checked out for a total of ${sumOfProductPrices}.**

**Info Команда**

**Customer**

При изпълнение с параметър **info** = "**Customer**" върнете:

**Съобщение**:

**Current customer has:  
Products: {countOfPhysicalProductsAndServiceProducts}  
Total Bill: ${sumOfProductPrices}**

**Shop**

При изпълнение с параметър **info** = "**Shop**" върнете:

**Съобщение**:

**Receipts:  
{receipt1.ToString()}**

**{receipt2.ToString()}  
...**

**{receiptN.ToString()}**

В случай, че няма виртуални фактури в системата, изведете: **Съобщение**: **Receipts: No receipts**

**End Команда**

При изпълнение върнете:

**Съобщение**:

**Total customers today: {count}**

**!!! ВАЖНО !!!: ВСИЧКИ реални числа**, трябва да бъдат закръгляни до 2 символа след десетичната запетая.

**Ограничения**

* Имената на продуктите, услугите и клиентите ще бъдат символни низове, които ще съдържат само английски букви и цифри и **dash** (**долна черта**) "**\_**".
* Винаги ще получавате команда за приключване на програмата.
* Входните данни ще бъдат валидни от страна на типове данни.
* **Checkout** командата, никога няма да се извиква без да има продукти в системата.
* **ВСИЧКИ реални числа**, трябва да бъдат закръглени до 2 символа след десетичната запетая.

**Примери**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Product Zara\_Jeans 49.95 0.35**  **Product Adidas\_Sneekers\_Z40 195.99 0.12**  **Service Fitness 29.95 1.25**  **Info Customer**  **Info Shop**  **Checkout Pesho**  **Info Shop**  **End** | **The current customer has bought Zara\_Jeans.**  **The current customer has bought Adidas\_Sneekers\_Z40.**  **The current customer has applied for Fitness service.**  **Current customer has:**  **Products: 3**  **Total Bill: $275.89**  **Receipts: No receipts**  **Customer checked out for a total of $275.89.**  **Receipts:**  **Receipt of Pesho**  **Total Price: 275.89**  **Products:**  **Name: Zara\_Jeans**  **Price: 49.95**  **Weight: 0.35**  **Name: Adidas\_Sneekers\_Z40**  **Price: 195.99**  **Weight: 0.12**  **Name: Fitness**  **Price: 29.95**  **Time: 1.25**  **Total customers today: 1** |
| **Product Sport\_Jacket 199.95 1.35**  **Product Thomas\_Shirt 95.84 1.01**  **Service Salt\_Room 9.95 1.15**  **Checkout Alex**  **Service Fitness 29.35 1.15**  **Checkout Donald**  **Info Shop**  **End** | **The current customer has bought Sport\_Jacket.**  **The current customer has bought Thomas\_Shirt.**  **The current customer has applied for Salt\_Room service.**  **Customer checked out for a total of $305.74.**  **The current customer has applied for Fitness service.**  **Customer checked out for a total of $29.35.**  **Receipts:**  **Receipt of Alex**  **Total Price: 305.74**  **Products:**  **Name: Sport\_Jacket**  **Price: 199.95**  **Weight: 1.35**  **Name: Thomas\_Shirt**  **Price: 95.84**  **Weight: 1.01**  **Name: Salt\_Room**  **Price: 9.95**  **Time: 1.15**  **Receipt of Donald**  **Total Price: 29.35**  **Products:**  **Name: Fitness**  **Price: 29.35**  **Time: 1.15**  **Total customers today: 2** |

* **Точкуване**

Всяка различна задача Ви дава точки:

* Структура - 30 точки
* Бизнес логика - 50 точки
* Вход/Изход - 20 точки

**Requested files**

**StartUp.cs**

using System;

class StartUp

{

static void Main(string[] args)

{

Controller controller = new Controller();

Engine engine = new Engine(controller);

engine.Run();

}

}

**Engine.cs**

using System;

using System.Collections.Generic;

using System.Linq;

public class Engine

{

private Controller controller;

private bool isRunning;

public Engine(Controller controller)

{

this.controller = controller;

this.isRunning = true;

}

public void Run()

{

while (isRunning)

{

string[] splittedInput = Console.ReadLine().Split();

string command = splittedInput[0];

List<string> arguments = splittedInput

.Skip(1)

.ToList();

string result = "";

switch (command)

{

case "Product":

result = controller.ProcessProductCommand(arguments);

break;

case "Service":

result = controller.ProcessServiceCommand(arguments);

break;

case "Checkout":

result = controller.ProcessCheckoutCommand(arguments);

break;

case "Info":

result = controller.ProcessInfoCommand(arguments);

break;

case "End":

result = controller.ProcessEndCommand();

this.isRunning = false;

break;

default:

result = "Invalid command";

break;

}

Console.WriteLine(result);

}

}

}

**Product.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ShoppingSystem

{

public abstract class Product

{

private string name;

private double price;

protected Product(string name, double price)

{

this.Name = name;

this.Price = price;

}

public string Name

{

get { return this.name; }

private set

{

if (value.Length < 3 || value.Length > 30)

{

throw new ArgumentException("Name should be between 3 and 30 characters!");

}

this.name = value;

}

}

public double Price

{

get { return this.price; }

private set

{

if (value < 0)

{

throw new ArgumentException("Price should be 0 or positive!");

}

this.price = value;

}

}

public override string ToString()

{

return $"Name: {this.Name}" + Environment.NewLine +

$"Price: {this.Price}";

}

}

}

**PhysicalProduct.cs**

using System;

namespace ShoppingSystem

{

public class PhysicalProduct

{

// TODO: Implement me...

}

}

**ServiceProduct.cs**

using System;

namespace ShoppingSystem

{

public class ServiceProduct

{

// TODO: Implement me...

}

}

**Receipt.cs**

using System;

using System.Collections.Generic;

using System.Linq;

namespace ShoppingSystem

{

public class Receipt

{

// TODO: Implement me...

}

}

**Controller.cs**

using ShoppingSystem;

using System;

using System.Collections.Generic;

public class Controller

{

public Controller()

{

}

public string ProcessProductCommand(List<string> args)

{

// TODO: Implement me...

return null;

}

public string ProcessServiceCommand(List<string> args)

{

// TODO: Implement me...

return null;

}

public string ProcessCheckoutCommand(List<string> args)

{

// TODO: Implement me...

return null;

}

public string ProcessInfoCommand(List<string> args)

{

// TODO: Implement me...

return null;

}

public string ProcessEndCommand()

{

// TODO: Implement me...

return null;

}

}