**Великотърновски университет „Св. Св. Кирил и Методий“**

**Факултет «Математика и информатика»**

**Курсова работа**

**по дисциплината: “Шаблони за софтуерен дизайн”**

Тема: **Product Store**

|  |  |
| --- | --- |
| **Студент:**  Йоанна Емилова Йорданова  Специалност: Информатика  фак.№: 2009010568 | **Ръководители:**  доц. д-р Златко  Георгиев Върбанов |
|  |  |

**Велико Търново**

**2023**

**Съдържание:**

1. Цел на разработката.
2. Използвани шаблони за софтуерен дизайн.

2.1 Класификация на шаблоните.

1. Основни етапи на разработката.
2. Демонстрация на функционалностите на проекта.
3. Приложение на получените резултати.

**1.Цел на разработката**

Цел на разработката е да се създаде конзолно приложение, включващо един от трите вида шаблони за софтуерен дизайн – creational, structural и behavioural (създаващ, структурен и поведенчески). Проектът е реализиран в интегрираната среда за разработка на софтуерни приложения Microsoft Visual Studio 2022 и e написан на езика C#. Той представлява симулация на електронен магазин за техника.

**2.Използвани шаблони за софтуерен дизайн**

За изпълнението на проекта са използвани шаблоните :

1. Singleton - в класа **Store** се използва шаблонът Singleton за гарантиране, че има само една инстанция на класа в системата. Това се постига чрез използването на статична променлива **instance** и статичен метод **GetInstance()**, който създава инстанция на класа. Така се осигурява достъп до единствената инстанция на **Store**.
2. Adapter - в класа **ProductAdapter** се използва шаблонът Adapter. Той служи за преобразуване на интерфейса на класа ( (**List<Product>**) към друг интерфейс (**IProductTarget**) ), така че да отговаря на нуждите на клиентския код. В случая **ProductAdapter** адаптира списъка с продукти в магазина, като предоставя методи за извеждане на продуктите на конзолата и извличане на цените на продуктите.
3. Strategy - в класа **ShoppingCart** се използва шаблонът Strategy за избор на различни стратегии за отстъпка. Класът има променлива от тип **IDiscountStrategy**, която съхранява конкретната стратегия за отстъпка. Това позволява динамично да се променя стратегията, без да се променя кодът на класа **ShoppingCart**.
4. Factory - (в класовете **PhoneCreator**, **LaptopCreator**, **HeadphonesCreator**, **TVCreator** и интерфейса **IProductCreator**) - този шаблон се използва, за да се създадат обекти от различни типове продукти, като се изнесе процесът на създаване на обекта в отделни класове (**PhoneCreator**, **LaptopCreator**, **HeadphonesCreator**, **TVCreator**). Интерфейсът **IProductCreator** дефинира метода **CreateProduct()**, който създава и връща конкретен продукт.

**2.2.Kласификация на шаблоните**

Създаващ шаблон:

* Singleton;
* Factory.

Поведенчески шаблон:

* Strategy.

Структурен шаблон:

* Adapter.

**3.Основни етапи на разработката**

//ShoppingCart

using System;

using ProductStore.Adapters;

using ProductStore.Strategies;

namespace ProductStore

{

class ShoppingCart

{

private List<Product> products;

private IDiscountStrategy discountStrategy;

private ProductAdapter productAdapter;

public ShoppingCart()

{

products = new List<Product>();

productAdapter = new ProductAdapter(products);

}

public void SetDiscountStrategy(IDiscountStrategy strategy)

{

discountStrategy = strategy;

}

public void AddProduct(Product product, int quantity)

{

products.Add(product);

product.Quantity = quantity;

}

public void CalculateTotalPrice()

{

decimal totalPrice = 0;

foreach (var product in products)

{

var quantity = product.Quantity;

totalPrice += product.Price \* quantity;

}

if (discountStrategy != null)

{

decimal discount = discountStrategy.CalculateDiscount(totalPrice);

totalPrice -= discount;

}

Console.WriteLine($"Total Price: {Math.Round(totalPrice, 2):F2}");

Console.WriteLine("Пълна количка с продукти:");

productAdapter.DisplayProducts();

Console.WriteLine();

}

}

}

//Program

using ProductStore.Factories;

using ProductStore.Services;

using ProductStore.Strategies;

using System;

namespace ProductStore

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Продукти в магазина:");

var storeManager = new StoreManager(); //шаблонът Dependency Injection (внедряване на зависимости)

storeManager.InitializeStore();

storeManager.DisplayStoreProducts();

var cart = new ShoppingCart();

cart.SetDiscountStrategy(new BlackFridayDiscountStrategy()); //шаблонът Strategy (стратегия за отстъпка)

bool addToCart = true;

while (addToCart)

{

Console.WriteLine("Изберете продукт от магазина, като въведете името му. За да приключите покупката въведете 'край'. ");

var input = Console.ReadLine();

if (input.ToLower() == "край")

{

addToCart = false;

}

else

{

Product product = null;

if (input == "Phone")

{

product = new PhoneCreator().CreateProduct();

}

else if (input == "Laptop")

{

product = new LaptopCreator().CreateProduct();

}

else if (input == "Headphones")

{

product = new HeadphonesCreator().CreateProduct();

}

else if (input == "TV")

{

product = new TVCreator().CreateProduct();

}

if (product != null)

{

Console.WriteLine("Въведете желано количество: ");

var quantityInput = Console.ReadLine();

if (int.TryParse(quantityInput, out int quantity))

{

if (storeManager.IsProductAvailable(product.Name, quantity))

{

cart.AddProduct(product, quantity);

storeManager.RemoveProductQuantity(product.Name, quantity);

Console.WriteLine($"{quantity} броя от {product.Name} са добавени в кошницата.");

}

else

{

Console.WriteLine($"Недостатъчно количество от {product.Name} в магазина.");

}

}

else

{

Console.WriteLine("Невалидно количество!");

}

}

else

{

Console.WriteLine("Невалиден продукт!");

}

}

}

cart.CalculateTotalPrice();

Console.WriteLine("Благодарим за поръчката! Заповядайте отново при нас!");

}

}

}

//Product

using System;

namespace ProductStore

{

abstract class Product

{

public string Name { get; set; }

public decimal Price { get; set; }

public int Quantity { get; set; }

}

class Phone : Product { }

class Laptop : Product { }

class Headphones : Product { }

class TV : Product { }

}

//IDiscountStrategy

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ProductStore.Strategies

{

interface IDiscountStrategy

{

decimal CalculateDiscount(decimal price);

}

}

//BlackFridayDiscountStrategy

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ProductStore.Strategies

{

class BlackFridayDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(decimal price)

{

return price \* 0.2m; // 20% отстъпка

}

}

}

//Store

using ProductStore.Factories;

using System;

using System.Collections.Generic;

using System.Linq;

namespace ProductStore.Services

{

class Store

{

//шаблонът Singleton

private static Store instance;

private List<Product> products;

private Store()

{

products = new List<Product>();

}

public static Store GetInstance()

{

if (instance == null)

{

instance = new Store();

}

return instance;

}

public void Initialize()

{

products.Add(new PhoneCreator().CreateProduct());

products.Add(new LaptopCreator().CreateProduct());

products.Add(new HeadphonesCreator().CreateProduct());

products.Add(new TVCreator().CreateProduct());

}

public void DisplayProducts()

{

foreach (var product in products)

{

Console.WriteLine($"Product: {product.Name}, Price: {product.Price}, Quantity: {product.Quantity}");

}

}

public bool IsProductAvailable(string productName, int quantity)

{

var product = products.FirstOrDefault(p => p.Name == productName);

if (product == null)

return false;

return product.Quantity >= quantity;

}

public void RemoveProductQuantity(string productName, int quantity)

{

var product = products.FirstOrDefault(p => p.Name == productName);

if (product != null)

product.Quantity -= quantity;

}

internal Product GetProductByName(string productName)

{

return products.FirstOrDefault(p => p.Name == productName);

}

public List<Product> GetAllProducts()

{

return products;

}

}

}

//StoreManager

using System;

using System.Collections.Generic;

using System.Linq;

using ProductStore.Adapters;

using ProductStore.Factories;

using static System.Formats.Asn1.AsnWriter;

namespace ProductStore.Services

{

class StoreManager

{

private Store store; //шаблонът Singleton

private IProductTarget productTarget;

private IProductCreator productCreator; // Добавяме интерфейс за Factory Method

public StoreManager()

{

store = Store.GetInstance(); //шаблонът Singleton

productTarget = new ProductAdapter(store.GetAllProducts());

productCreator = new PhoneCreator(); // Използваме Factory Method

}

public void InitializeStore()

{

store.Initialize();

}

public void DisplayStoreProducts()

{

productTarget.DisplayProducts();

}

public Dictionary<string, decimal> GetStoreProductPrices()

{

return productTarget.GetProductPrices();

}

public bool IsProductAvailable(string productName, int quantity)

{

var product = store.GetProductByName(productName) as Product;

if (product != null && product.Quantity >= quantity)

{

return true;

}

return false;

}

public void RemoveProductQuantity(string productName, int quantity)

{

var product = store.GetProductByName(productName) as Product;

if (product != null)

{

product.Quantity -= quantity;

}

}

// Factory Method, използван в метода AddProduct

private Product CreateProduct()

{

return productCreator.CreateProduct();

}

}

}

//IProductCreator

using System;

namespace ProductStore.Factories

{

interface IProductCreator

{

Product CreateProduct();

}

}

//ProductCreators

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ProductStore.Factories

{

class PhoneCreator : IProductCreator

{

public Product CreateProduct()

{

return new Phone { Name = "Phone", Price = 500, Quantity = 7 };

}

}

class LaptopCreator : IProductCreator

{

public Product CreateProduct()

{

return new Laptop { Name = "Laptop", Price = 1000, Quantity = 20 };

}

}

class HeadphonesCreator : IProductCreator

{

public Product CreateProduct()

{

return new Headphones { Name = "Headphones", Price = 100, Quantity = 15 };

}

}

class TVCreator : IProductCreator

{

public Product CreateProduct()

{

return new TV { Name = "TV", Price = 1500, Quantity = 1 };

}

}

}

//IProductTarget

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ProductStore.Adapters

{

public interface IProductTarget

{

void DisplayProducts();

Dictionary<string, decimal> GetProductPrices();

}

}

//ProductAdapter

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ProductStore.Adapters

{

class ProductAdapter : IProductTarget

{

private List<Product> products;

public ProductAdapter(List<Product> products)

{

this.products = products;

}

public void DisplayProducts()

{

foreach (var product in products)

{

Console.WriteLine($"Product: {product.Name}, Price: {product.Price}, Quantity: {product.Quantity}");

}

}

public Dictionary<string, decimal> GetProductPrices()

{

var productPrices = new Dictionary<string, decimal>();

foreach (var product in products)

{

productPrices.Add(product.Name, product.Price);

}

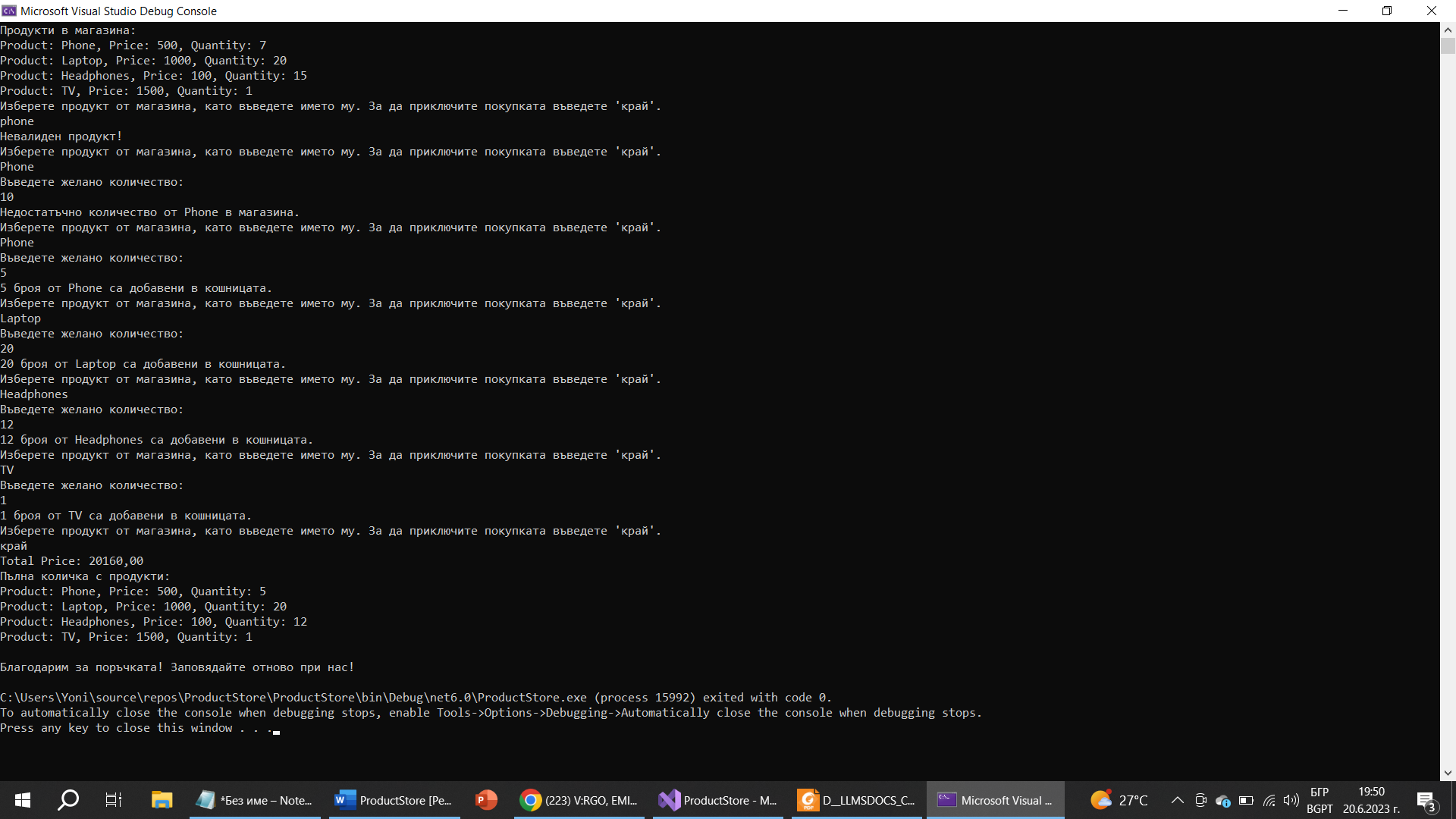
return productPrices;

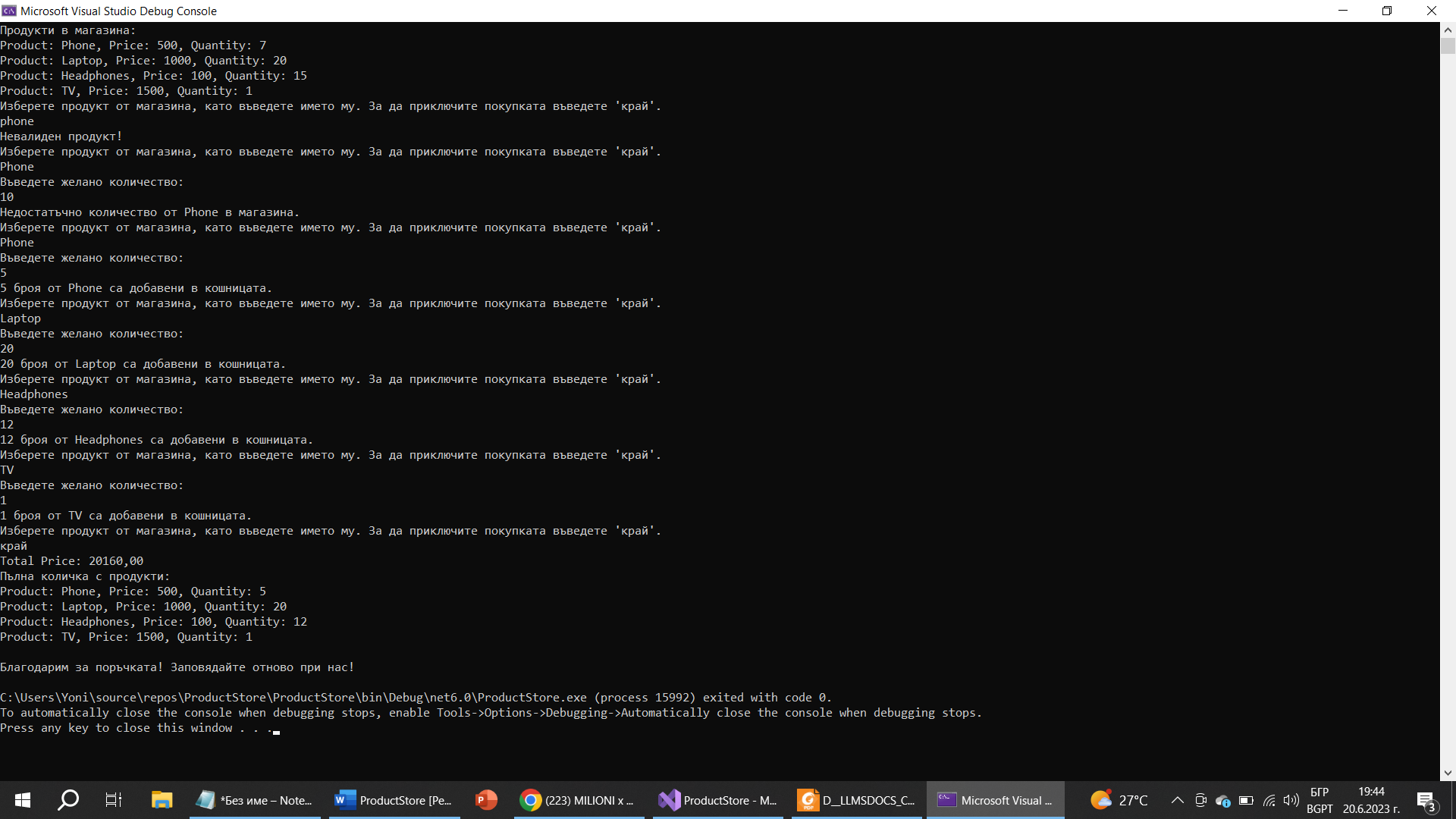
}

}

}

**4. Демонстрация на функционалностите на проекта**

 При стартиране на програмата можем да видим списък със стоки, които се предлагат в магазина, тяхната цена и количеството, което има в наличност.

Клиентът, за да избере продукт от списъка, трябва да го изпише по абсолютно същият начин, в противен случай на конзолата ще се изведе съобщението - „Невалиден продукт“. След като сме избрали продукт и той е отишъл в нашата количка, следващата стъпка е да изберем желаното количество. Ако надвишим количеството конзолата отново ще сигнализира „Недостатъчно количество от … в магазина.“. Зададеното количество от определен продукт автоматично се намаля в магазина и отива в нашата количка. При приключване с поръчката е достатъчно клиентът да изпише „край“ на конзолата. Ще се изведе списък с продуктите в количката, тяхната стойност и бройка. От крайната цена, която клиентът дължи, се прави отстпка -20% „Черен петък“.

**5.Приложение на получените резултати**

Това софтуерно приложение представлява проста имплементация на магазин за продукти, където потребителите могат да разглеждат наличните продукти и да ги добавят в кошницата за покупка. Приложението предлага и функционалност за пресмятане на общата цена на продуктите в кошницата, като се вземе предвид и отстъпката.

След като приложението бъде стартирано, потребителят ще види списък с наличните продукти в магазина. Той може да избере продукт от списъка, като въведе името му. След това потребителят може да въведе желаното количество от избрания продукт. Ако продуктът е наличен в достатъчно количество, той се добавя в кошницата, а наличността му в магазина се намалява съответно.

Когато потребителят приключи с покупката, той може да въведе "край", за да излезе от цикъла на добавяне на продукти в кошницата. След това се изчислява общата цена на продуктите в кошницата, като се прилага и отстъпката, определена от избраната стратегия. Крайната цена се извежда на екрана, а под нея – списък с продукти в кошницата със съответното избрано количество. Приложението приключва.