ВТУ "Св. Св. Кирил и Методий"
Факултет "Математика и Информатика"

КУРСОВА РАБОТА
ПО ДИСЦИПЛИНАТА
ШАБЛОНИ ЗА СОФТУЕРЕН ДИЗАЙН I ЧАСТ
На тема:
СКЛАД КЪМ ЕЛЕКТРОНЕН МАГАЗИН

Изготвил:
Пламенна Викторова Петрова
Специалност:
Софтуерно инженерство
Факултетен № 1909011132

Проверил:
доц. д-р.
Златко Георгиев
Върбанов

Велико Търново
2021

Изисквания

Проектът трябва да съдържа адекватна имплементация на поне 3 шаблона (поне един създаващ, поне един структурен и поне един поведенчески). Проектът се публикува в GitHub, заедно с текстова документация към него (детайлно описание какво съдържа проектът, как е реализиран всеки един от шаблоните и кои класове в имплементацията на кои класове от схематичното описание на шаблона отговарят), най-късно до 3 дни преди датите за изпит.

Съдържание:

1. Цел и изпълнение на разработката

Цел на разработката е да се създаде конзолно приложение, включващо поне 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 2019 и е написан на езика C#. Той представлява симулация на склад на електронен магазин, а за изпълнението му са използвани шаблоните : Factory (създаващ), Command (поведенчески), Adapter (структурен) и Memento (поведенчески). Добавянето на нови продукти в склада става посредством шаблона Factory. В папката Models са поместени абстрактният клас StoredProduct, играещ ролята на Product (интерфейс/абстрактен клас, служещ за дефиниране на обекти, които се създават от factory метода) и класовете BoardGame, ClothingMerch и GamingPeripherals, които наследяват StoredProduct и препокриват пропъртита му както и абстрактният void метод PrintStoredProductInfo(). Още конструкторите на класовете наследници наследяват параметрите от базовия конструктор. Няма нови уникални полета/пропъртита, които да участват в ConcreteProduct класовете – BoardGame, ClothingMerch и GamingPeripherals. Във всеки ConcreteProduct клас е прибавено полето string description, което намира употреба при override-ването на пропъртито public abstract string Description { get; set; }.Особеност на StoredProduct конструктора е match-ването директно на пропъртита към параметрите му. Затова и изписването на this пред пропъртитата в конструктора се явява ненужно. Но не всички пропъртита от StoredProduct са абстрактни. Изключение правят именно Barcode, Name и Price, които се съпоставят на параметрите на конструктора на абстрактния клас. Тъй като неабстрактните модели представляват отделни продуктови категории, при override-ване на абстрактното пропърти public abstract string Category { get; } от StoredProduct в getter-ът връща името на класа чрез reflection GetType().Name. Това е и единственото пропърти без setter. Методът PrintStoredProductInfo() извежда информация за баркода, името на продукта и цената му. В папката Creators се съхраняват Creator интерфейсът IStoredProductCreator. В него е деклариран абстрактният метод public abstract StoredProduct CreateStoredProduct(), който бива override-нат от ConcreteCreator класовете BoardGameCreator, ClothingMerchCreator и GamingPeripheralsCreator. При всяко едно препокриване се следва една и съща схема : баркодът, името и

цената на продукта се изписват на конзолата от клавиатурата и се return-ва нов обект, в чийто конструктор се вземат input стойностите като аргументи. Така ConcreteCreator-ът работи с обекти от ConcreteProduct класовете, наследяващи абстрактния Product клас. Следващият реализиран шаблон е Command (поведенчески). За Receiver класа е избран интерфейсът IWarehouse от папката WarehouseRepository, за да може в последствие да се демонстрира и depencency injection в ConcreteCommand класа – WarehouseCommand. В repository интерфейса IWarehouse са декларирани методите :

1) List<StoredProduct> GetStoredProducts();
2) void AddStoredProduct(StoredProduct storedProduct);
3) StoredProduct FindStoredProduct(string barcode);
4) void EditStoredProduct(StoredProduct oldStoredProduct, StoredProduct newStoredProduct);
5) void RemoveStoredProduct(StoredProduct storedProduct);
6) void PrintStoredProducts();
7) void IncreasePrice(StoredProduct storedProduct, int amount);
8) bool DecreasePrice(StoredProduct storedProduct, int amount);

Те са имплементирани в repository класа Warehouse. Там е деклариран списък от складови продукти от тип абстрактния клас StoredProduct (Factory Product class) като поле. Конструкторът на Warehouse се build-ва без параметри, т.е в него директно се инстанцира нов списък от StoredProduct продукти. Първият метод от по-горе номерираните връща ToList-нат целия инвентар от продукти (полето private List<StoredProduct> inventory), като нивото на достъп private не позволява предефинирането на field-а от други клас. На втория метод трябва да се подаде като аргумент обект от тип StoredProduct при извикване. В тялото на метода се извършва проверка дали този обект има стойност null. Ако не, той бива добавен към inventory. Третият метод намира определен продукт по баркод. Използван е Linq изразът List<T>.Find(Predicate<T> match), където списъкът е инвентарът от продукти, а условието на предиката е аргументът да съответства на баркода на обекта, взет от пропъртито му. В четвъртия метод трябва да бъдат подадени два аргумента според параметрите от тип StoredProduct – стар продукт, който да се премахне от склада и нов, който да се добави към складовия инвентар. В този метод участва и последно описания. Старият продукт се намира по баркода му. След това се извършва валидация за null стойности на обектите. При успешна валидация старият продукт се remove-ва от списъка и новият се add-ва. Петият метод предполага изтриването на StoredProduct обект от инветара, като първо се проверява дали съществува с Find метода по баркода му.

Шестият метод обхожда всички обекти от List-а с един for цикъл. Условието за length се покрива от готовия метод inventory.Count(); Информацията за всеки i-ти обект се принтира чрез override-натия абстрактен метод PrintStoredProductInfo() на Factory Product класа. Предназначението на седмия метод е да увеличи цената на даден обект от тип StoredProduct с определен количество (integer стойност), ако обектът не е nullable. Осмият метод е булев и връща true, ако цената на продукта е по-голяма от количество, с което трябва да се намали. Извършва се понижаването й. В противен случай методът връща false и не се стига до неправилно понижаване (цената да придобие отрицателна стойност). После са оформени Command и ConcreteCommand в лицето на интерфейса IWarehouseCommand и имплементиращия го клас WarehouseCommand. В интерфейса се декларират методите void ExecuteAction(); и void UndoAction(); Класът WarehouseCommand си служи с WarehouseAction enum, в който са изброени действията Add, Remove, Edit, Print, Increase и Decrease. Декларирани са полетата :

1) private readonly IWarehouse warehouse; - repository интерфейса на склада
2) private readonly WarehouseAction warehouseAction; - enum action
3) private readonly StoredProduct storedProduct; - обект на складиран продукт
4) private readonly StoredProduct oldStoredProduct; - стар продукт
5) private readonly StoredProduct newStoredProduct; - нов продукт
6) private readonly int amount; - количество

Всички те са само за четене (readonly).
Пропъртито public bool IsCommandExecuted { get; private set; } проверява дали командата е изпълнена вътрешно за класа, но може да се достъпва външно. Използван е constructor overloading. Основният конструктор има параметри IWarehouse обект и WarehouseAction enum item. Вторият конструктор overload-ва основния с допълнителен параметър – обект от тип StoredProduct и наследява параметрите му. Този конструктор генерира командите Add и Remove. Третият конструктор е насочен към инстанция на командата Edit. Участват два екстра параметъра – StoredProduct oldStoredProduct и StoredProduct newStoredProduct. И последният (четвърти) конструктор обосновава Increase и Decrease Price командите със специални параметри StoredProduct обект и целочислено количество, с което да се увеличи/намали цената на конкретния обект, ако е възможно. В тялото на ExecuteAction() метода се switch-ват WarehouseAction enum item-ите. Ако става въпрос например за Add се изпълнява Warehouse метода AddStoredProduct и IsCommandExecuted приема стойност true.

Аналогично според останалите изброени действия, се execute-ват прилежащите им методи от repository класа. В имплементацията на UndoActon() метода се проверява първоначално дали IsCommandExecuted има default-ната си false стойност. Ако проверката е правилна, изпълнението на метода завършва с return; В противен случай процедира по следния начин : търсят се кои WarehouseAction действия са посочени и се извикват Warehouse методи, които да ги анулират. Примерно при намерен WarehouseAction Add, добавеният продукт в склада се изтрива и обратно, ако има премахнат продукт, той се добавя наново или ако стар продукт е заменен с нов, се връща старият. Същото важи и за увеличаване и намаляване на цените. При Increase цената се намалява до първоначалната, а при Decrease цената се увеличава до първоначалната. В Program класа на проекта е даден пример и за hard delete стъпка по стъпка за целия склад спрямо дефинирани всички възможни Action, т.е изчиства се целият склад от досегашните си продукти. Invoker класът WarehouseInvoker поисква командата да изпълни request-а. Декларирани са две private readonly полета List<IWarehouseCommand> warehouseCommands (списък от команди) и IWarehouseCommand warehouseCommand (конкретна команда). В конструктора на Invoker-а се инстанцира нов списък от команди. В метода setCommand(IWarehouseCommand command) се set-ва конкретната команда. В Invoke() пък се добавя дадена команда към списъка и се извиква ExecuteAction() за нея. И в последния метод за класа UndoActions() са foreach-нати всички команди в обратен ред чрез Enumerable.Reverse(warehouseCommands) за списъка, като за всяка от тях се изпълнява UndoAction(); Именно този метод на Invoker-а играе ролята на hard delete за Warehouse инвентара. Третият шаблон, имплементиран в проекта е Adapter. За Target е взет IWarehouse интерфейсът. Функцията на Target-а е да дефинира специфичния за домейна интерфейс, използван от клиента. IDelivery интерфейсът представлява Adaptee. За една доставка се отделят определени продукти от склада, които се натоварват на камион, за да бъдат закарани до разпределителен пункт. В интерфейса са декларирани методите:

1) List<StoredProduct> CheckProductsBeforeLoading();
2) void LoadTruck(StoredProduct productToDeliver);
3) StoredProduct LocateProductToDeliver(string barcode);
4) void ReplaceProductBeforeLoading(StoredProduct oldProductToDeliver, StoredProduct newProductToDeliver);
5) void RemoveFromTruck(StoredProduct productToDeliver);
6) void PrintLoadedProducts();

7) void IncreaseDeliveryPrice(StoredProduct productToDeliver, int amount);

8) bool DecreaseDeliveryPrice(StoredProduct productToDeliver, int amount);

Те имат подобна функционалност на методите от Warehouse класа, но идеята тук е да се раздели логиката на две – да има отделни CRUD, Get, Print, Increase Price и Decrease Price операции за целия склад и други за продуктите, които се зареждат за доставка, като операциите за продуктите, участващи в доставката, се адаптират спрямо общите за всички складирани продукти. Delivery класът имплементира тези методи. Първият извършва проверка на продуктите преди натоварването им, извеждайки ги в списък. Вторият добавя нов продукт в камиона. Третият намира даден продукт по баркода. Четвъртият разменя два продукта, преди да бъдат поставени в превозното средство. Петият премахва продукт от камиона, а шестият принтира всички заредени продукти. Седмият увеличава цената на доставка и накрая осмият намалява цената на доставка. В Adapter класа DeliveryAdapter се работи с поле от тип IDelivery. То се inject-ва в конструктора. Понеже DeliveryAdapter-ът имплементира методите от Target-а IWarehouse, в тялото им се извикват съответстващите им методи от Delivery класа, като параметрите на методите от IWarehouse се подават като аргументи на IDelivery методите. Например :

```
public void IncreasePrice(StoredProduct productToDeliver, int amount)
{
   delivery.IncreaseDeliveryPrice(productToDeliver, amount);
}
```

Последният шаблон, вграден в конзолното приложение, е Memento (поведенчески). Ролята на Memento класа е да съхрани вътрешното състояние на Originator обекта. Затова в WarehouseMemento класа на са обозначени поле и readonly пропърти за това състояние. Полето бива и параметър на конструктора. Originator класът съдържа методите CreateWarehouseMemento() и SetMemento(). Първият връща нов обект от тип WarehouseMemento с текущото вътрешно състояние на мементото, а вторият възстановява първоначалното му състояние. И не на последно място част от шаблона е и Caretaker класът, който дефинира поле от тип

WarehouseMemento и пропърти с getter, return-ващ WarehouseMemento обект и setter, задаващ стойност на обект от посочения тип. В проекта за Memento States се използват Non-Fragile (нечуплив за продукт) – default-ният – или този, който се restore-ва в SetMemento() метода на Originator класа и Fragile (чуплив), който се set-ва в Originator пропъртито, когато в описанието на продукта е открита думата “fragile”.

2. Основни етапи на разработката

2.1. Създаване на Factory Product класа и ConcreteProduct наследниците

StoredProduct - Product

```
// Abstract Product
61 references
public abstract class StoredProduct
{
    10 references
    public string Barcode { get; set; }
    13 references
    public string Name { get; set; }
    10 references
    public double Price { get; set; }
    6 references
    public abstract string Category { get; }
    13 references
    public abstract string Description { get; set; }

    3 references
    protected StoredProduct(string barcode, string name, double price)
    {
        Barcode = barcode;
        Name = name;
        Price = price;
    }

    5 references
    public abstract void PrintStoredProductInfo();
}
```

BoardGame - ConcreteProduct

```
// Concrete Product
2 references
public class BoardGame : StoredProduct
{
    public string description;

    1 reference
    public BoardGame(string barcode, string name, double price)
        : base(barcode, name, price)
    {

    }

    6 references
    public override string Category => GetType().Name;

    13 references
    public override string Description { get => description; set => description = value; }

    5 references
    public override void PrintStoredProductInfo()
    {
        Console.WriteLine($"Printing the most important information about the product...");
        Console.WriteLine($"Barcode: {Barcode}, Name: {Name}, Price: {Price}, Category: {Category}");
    }
}
```

ClothingMerch – ConcreteProduct

```
// Concrete Product
2 references
public class ClothingMerch : StoredProduct
{
    public string description;

    1 reference
    public ClothingMerch(string barcode, string name, double price)
        : base(barcode, name, price)
    {

    }

    6 references
    public override string Category => GetType().Name;

    13 references
    public override string Description { get => description; set => description = value; }

    5 references
    public override void PrintStoredProductInfo()
    {
        Console.WriteLine($"Printing the most important information about the product...");
        Console.WriteLine($"Barcode: {Barcode}, Name: {Name}, Price: {Price}, Category: {Category}");
    }
}
```

GamingPeripherals – ConcreteProduct

```
// Concrete Product
2 references
public class GamingPeripherals : StoredProduct
{
    public string description;

    1 reference
    public GamingPeripherals(string barcode, string name, double price)
        : base(barcode, name, price)
    {

    }

    6 references
    public override string Category => GetType().Name;

    13 references
    public override string Description { get => description; set => description = value; }

    5 references
    public override void PrintStoredProductInfo()
    {
        Console.WriteLine($"Printing the most important information about the product...");
        Console.WriteLine($"Barcode: {Barcode}, Name: {Name}, Price: {Price}, Category: {Category}");
    }
}
```

2.2. Добавяне на Factory Creator интерфейса и имплементиращите го ConcreteCreator-и

IStoredProduct – ConcreteCreator

```
namespace Design_Patterns_Course_Project_Warehouse.Factory.Creators
{
    // Creator Interface
    4 references
    public interface IStoredProductCreator
    {
        12 references
        public abstract StoredProduct CreateStoredProduct();
    }
}
```

BoardGameCreator – ConcreteCreator

```
// Concrete Creator
3 references
public class BoardGameCreator : IStoredProductCreator
{
    12 references
    public StoredProduct CreateStoredProduct()
    {
        Console.WriteLine("Enter new product info -----------");
        Console.Write("Enter stored product barcode : ");
        string barcode = Console.ReadLine();
        Console.Write("Enter stored product name : ");
        string name = Console.ReadLine();
        Console.Write("Enter stored product price : ");
        double price = double.Parse(Console.ReadLine());
        Console.WriteLine("---------------------------------");
        return new BoardGame(barcode, name, price);
    }
}
```

ClothingMerchCreator – ConcreteCreator

```
// Concrete Creator
2 references
public class ClothingMerchCreator : IStoredProductCreator
{
    12 references
    public StoredProduct CreateStoredProduct()
    {
        Console.WriteLine("Enter new product info -----------");
        Console.Write("Enter stored product barcode : ");
        string barcode = Console.ReadLine();
        Console.Write("Enter stored product name : ");
        string name = Console.ReadLine();
        Console.Write("Enter stored product price : ");
        double price = double.Parse(Console.ReadLine());
        Console.WriteLine("---------------------------------");
        return new ClothingMerch(barcode, name, price);
    }
}
```

GamingPeripheralsCreator – ConcreteCreator

```
// Concrete Creator
4 references
public class GamingPeripheralsCreator : IStoredProductCreator
{
    12 references
    public StoredProduct CreateStoredProduct()
    {
        Console.WriteLine("Enter new product info -----------");
        Console.Write("Enter stored product barcode : ");
        string barcode = Console.ReadLine();
        Console.Write("Enter stored product name : ");
        string name = Console.ReadLine();
        Console.Write("Enter stored product price : ");
        double price = double.Parse(Console.ReadLine());
        Console.WriteLine("--------------------------------");
        return new GamingPeripherals(barcode, name, price);
    }
}
```

2.3. Създаване на Command Receiver – IWarehouse интерфейса плюс имплементиращия го клас

```
8 references
public interface IWarehouse
{
    5 references
    List<StoredProduct> GetStoredProducts();
    5 references
    void AddStoredProduct(StoredProduct storedProduct);
    4 references
    StoredProduct FindStoredProduct(string barcode);
    3 references
    void EditStoredProduct(StoredProduct oldStoredProduct, StoredProduct newStoredProduct);
    5 references
    void RemoveStoredProduct(StoredProduct storedProduct);
    3 references
    void PrintStoredProducts();
    4 references
    void IncreasePrice(StoredProduct storedProduct, int amount);
    4 references
    bool DecreasePrice(StoredProduct storedProduct, int amount);
}
```

Warehouse класът :

```
2 references
public class Warehouse : IWarehouse
{
    private List<StoredProduct> inventory;

    1 reference
    public Warehouse()
    {
        inventory = new List<StoredProduct>();
    }

    4 references
    public void IncreasePrice(StoredProduct storedProduct, int amount)
    {
        if (storedProduct != null)
        {
            storedProduct.Price += amount;
            Console.WriteLine($"The price for the stored product {storedProduct.Name} has been increased by {amount}");
        }
    }

    4 references
    public bool DecreasePrice(StoredProduct storedProduct, int amount)
    {
        if (amount < storedProduct.Price)
        {
            storedProduct.Price -= amount;
            Console.WriteLine($"The price for the stored product {storedProduct.Name} has been decreased by {amount}");
            return true;
        }
        return false;
    }
```

```
5 references
public void AddStoredProduct(StoredProduct storedProduct)
{
    if (storedProduct != null)
    {
        inventory.Add(storedProduct);
    }
}

5 references
public void RemoveStoredProduct(StoredProduct storedProduct)
{
    if (storedProduct == FindStoredProduct(storedProduct.Barcode))
    {
        inventory.Remove(storedProduct);
    }
}

4 references
public StoredProduct FindStoredProduct(string barcode)
{
    return inventory.Find(sp => sp.Barcode == barcode);
}

3 references
public void EditStoredProduct(StoredProduct oldStoredProduct, StoredProduct newStoredProduct)
{
    oldStoredProduct = FindStoredProduct(oldStoredProduct.Barcode);
    if (oldStoredProduct != null && newStoredProduct != null)
    {
        inventory.Remove(oldStoredProduct);
        inventory.Add(newStoredProduct);
    }
}
```

```
5 references
public List<StoredProduct> GetStoredProducts()
{
    return inventory.ToList();
}

3 references
public void PrintStoredProducts()
{
    for (int i = 0; i < inventory.Count(); i++)
    {
        inventory[i].PrintStoredProductInfo();
    }
}
```

2.4. Добавяне на Command и ConcreteCommand

IWarehouseCommand – Command

```
6 references
public interface IWarehouseCommand
{
    2 references
    void ExecuteAction();
    2 references
    void UndoAction();
}
```

WarehouseAction – Enum

```
41 references
public enum WarehouseAction
{
    Add,
    Remove,
    Edit,
    Print,
    Increase,
    Decrease
}
```

WarehouseCommand – ConcreteCommand

```
30 references
public class WarehouseCommand : IWarehouseCommand
{
    private readonly IWarehouse warehouse;
    private readonly WarehouseAction warehouseAction;
    private readonly StoredProduct storedProduct;
    private readonly StoredProduct oldStoredProduct;
    private readonly StoredProduct newStoredProduct;
    private readonly int amount;

    7 references
    public bool IsCommandExecuted { get; private set; }

    // Constructor overloading

    // For Print command
    11 references
    public WarehouseCommand(IWarehouse warehouse, WarehouseAction warehouseAction)
    {
        this.warehouse = warehouse;
        this.warehouseAction = warehouseAction;
    }

    // For Add and Remove commands
    13 references
    public WarehouseCommand(IWarehouse warehouse, WarehouseAction warehouseAction, StoredProduct storedProduct)
        : this(warehouse, warehouseAction)
    {
        this.storedProduct = storedProduct;
    }
```

```
// For Edit Command
1 reference
public WarehouseCommand(IWarehouse warehouse, WarehouseAction warehouseAction, StoredProduct oldStoredProduct, StoredProduct newStoredProduct)
    : this (warehouse, warehouseAction)
{
    this.oldStoredProduct = oldStoredProduct;
    this.newStoredProduct = newStoredProduct;
}

// For Increase and Decrease Commands
4 references
public WarehouseCommand(IWarehouse warehouse, WarehouseAction warehouseAction, StoredProduct storedProduct, int amount)
    : this (warehouse, warehouseAction)
{
    this.storedProduct = storedProduct;
    this.amount = amount;
}
```

```
2 references
public void ExecuteAction()
{
    switch (warehouseAction)
    {
        case WarehouseAction.Add:
            warehouse.AddStoredProduct(storedProduct);
            IsCommandExecuted = true;
            break;
        case WarehouseAction.Remove:
            warehouse.RemoveStoredProduct(storedProduct);
            IsCommandExecuted = true;
            break;
        case WarehouseAction.Edit:
            warehouse.EditStoredProduct(oldStoredProduct, newStoredProduct);
            IsCommandExecuted = true;
            break;
        case WarehouseAction.Print:
            warehouse.PrintStoredProducts();
            IsCommandExecuted = true;
            break;
        case WarehouseAction.Increase:
            warehouse.IncreasePrice(storedProduct, amount);
            IsCommandExecuted = true;
            break;
        case WarehouseAction.Decrease:
            IsCommandExecuted = warehouse.DecreasePrice(storedProduct, amount);
            break;
    }
}
```

```
2 references
public void UndoAction()
{
    if (!IsCommandExecuted)
        return;

    if (warehouseAction == WarehouseAction.Add)
    {
        Console.WriteLine($"Removing {storedProduct.Name}");
        warehouse.RemoveStoredProduct(storedProduct);
    }
    else if (warehouseAction == WarehouseAction.Remove)
    {
        Console.WriteLine($"Adding {storedProduct.Name}");
        warehouse.AddStoredProduct(storedProduct);
    }
    else if (warehouseAction == WarehouseAction.Edit)
    {
        Console.WriteLine("Undo edit");
        Console.WriteLine($"Removing {newStoredProduct.Name}");
        warehouse.RemoveStoredProduct(newStoredProduct);
        Console.WriteLine($"Adding {oldStoredProduct.Name}");
        warehouse.AddStoredProduct(oldStoredProduct);
    }
    else if (warehouseAction == WarehouseAction.Increase)
    {
        warehouse.DecreasePrice(storedProduct, amount);
    }
    else
    {
        warehouse.IncreasePrice(storedProduct, amount);
    }
}
```

2.5. Добавяне и на Invoker клас

WarehouseInvoker - Invoker

```
// Invoker
4 references
public class WarehouseInvoker
{
    private readonly List<IWarehouseCommand> warehouseCommands;
    private IWarehouseCommand warehouseCommand;

    2 references
    public WarehouseInvoker()
    {
        warehouseCommands = new List<IWarehouseCommand>();
    }

    1 reference
    public void SetCommand(IWarehouseCommand command)
    {
        warehouseCommand = command;
    }

    1 reference
    public void Invoke()
    {
        warehouseCommands.Add(warehouseCommand);
        warehouseCommand.ExecuteAction();
    }

    1 reference
    public void UndoActions()
    {
        foreach (var warehouseCommand in Enumerable.Reverse(warehouseCommands))
        {
            warehouseCommand.UndoAction();
        }
    }
}
```

2.6. Задаване на Adaptee от шаблона Adapter

IDelivery - Adaptee

```
4 references
public interface IDelivery
{
    2 references
    List<StoredProduct> CheckProductsBeforeLoading();
    2 references
    void LoadTruck(StoredProduct productToDeliver);
    4 references
    StoredProduct LocateProductToDeliver(string barcode);
    2 references
    void ReplaceProductBeforeLoading(StoredProduct oldProductToDeliver, StoredProduct newProductToDeliver);
    2 references
    void RemoveFromTruck(StoredProduct productToDeliver);
    2 references
    void PrintLoadedProducts();
    2 references
    void IncreaseDeliveryPrice(StoredProduct productToDeliver, int amount);
    2 references
    bool DecreaseDeliveryPrice(StoredProduct productToDeliver, int amount);
}
```

Delivery – класът, имплементиращ интерфейса IDelivery

```
private List<StoredProduct> productsToDeliver;

1 reference
public Delivery()
{
    productsToDeliver = new List<StoredProduct>();
}

2 references
public void IncreaseDeliveryPrice(StoredProduct productToDeliver, int amount)
{
    if (productToDeliver != null)
    {
        productToDeliver.Price += amount;
        Console.WriteLine($"The price for the product that will be delivered {productToDeliver.Name} has been increased by {amount}");
    }
}

2 references
public bool DecreaseDeliveryPrice(StoredProduct productToDeliver, int amount)
{
    if (amount < productToDeliver.Price)
    {
        productToDeliver.Price -= amount;
        Console.WriteLine($"The price for the stored product {productToDeliver.Name} has been decreased by {amount}");
    }
    return false;
}
```

```
2 references
public void LoadTruck(StoredProduct productToDeliver)
{
    if (productsToDeliver != null)
    {
        productsToDeliver.Add(productToDeliver);
    }
}

4 references
public StoredProduct LocateProductToDeliver(string barcode)
{
    return productsToDeliver.Find(ptd => ptd.Barcode == barcode);
}

2 references
public void ReplaceProductBeforeLoading(StoredProduct oldProductToDeliver, StoredProduct newProductToDeliver)
{
    oldProductToDeliver = LocateProductToDeliver(oldProductToDeliver.Barcode);
    if (oldProductToDeliver != null && newProductToDeliver != null)
    {
        productsToDeliver.Remove(oldProductToDeliver);
        productsToDeliver.Add(newProductToDeliver);
    }
}

2 references
public List<StoredProduct> CheckProductsBeforeLoading()
{
    return productsToDeliver.ToList();
}
```

```
2 references
public void RemoveFromTruck(StoredProduct productToDeliver)
{
    if (productToDeliver == LocateProductToDeliver(productToDeliver.Barcode))
    {
        productsToDeliver.Remove(productToDeliver);
    }
}

2 references
public void PrintLoadedProducts()
{
    for (int i = 0; i < productsToDeliver.Count(); i++)
    {
        productsToDeliver[i].PrintStoredProductInfo();
    }
}
```

2.7. Добавяне на Adapter класа

DeliveryAdapter – Adapter, Target-ът се явява IWarehouse

```
2 references
public class DeliveryAdapter : IWarehouse
{
    private IDelivery delivery;

    1 reference
    public DeliveryAdapter(IDelivery delivery)
    {
        this.delivery = delivery;
    }

    4 references
    public void IncreasePrice(StoredProduct productToDeliver, int amount)
    {
        delivery.IncreaseDeliveryPrice(productToDeliver, amount);
    }

    4 references
    public bool DecreasePrice(StoredProduct productToDeliver, int amount)
    {
        return delivery.DecreaseDeliveryPrice(productToDeliver, amount);
    }

    5 references
    public void AddStoredProduct(StoredProduct productToDeliver)
    {
        delivery.LoadTruck(productToDeliver);
    }

    4 references
    public StoredProduct FindStoredProduct(string barcode)
    {
        return delivery.LocateProductToDeliver(barcode);
    }
```

```
3 references
public void EditStoredProduct(StoredProduct oldProductToDeliver, StoredProduct newProductToDeliver)
{
    delivery.ReplaceProductBeforeLoading(oldProductToDeliver, newProductToDeliver);
}

5 references
public List<StoredProduct> GetStoredProducts()
{
    return delivery.CheckProductsBeforeLoading();
}

5 references
public void RemoveStoredProduct(StoredProduct productToDeliver)
{
    delivery.RemoveFromTruck(productToDeliver);
}

3 references
public void PrintStoredProducts()
{
    delivery.PrintLoadedProducts();
}
```

2.8. Имплементация на шаблона Memento

WarehouseMemento – Memento

```
6 references
public class WarehouseMemento
{
    public string state;

    1 reference
    public WarehouseMemento(string state)
    {
        this.state = state;
    }

    1 reference
    public string State
    {
        get
        {
            return state;
        }
    }
}
```

Originator

```
2 references
public class Originator
{
    public string state;

    3 references
    public string State
    {
        get
        {
            return state;
        }
        set
        {
            state = value;
            Console.WriteLine("State = " + state);
        }
    }

    1 reference
    public WarehouseMemento CreateWarehouseMemento()
    {
        return (new WarehouseMemento(state));
    }

    1 reference
    public void SetWarehouseMemento(WarehouseMemento warehouseMemento)
    {
        Console.WriteLine("Restoring state");
        State = warehouseMemento.State;
    }
}
```

Caretaker

```
2 references
public class Caretaker
{
    public WarehouseMemento warehouseMemento;

    2 references
    public WarehouseMemento WarehouseMemento
    {
        get
        {
            return warehouseMemento;
        }
        set
        {
            warehouseMemento = value;
        }
    }
}
```

3. Демонстрация на функционалностите на проекта

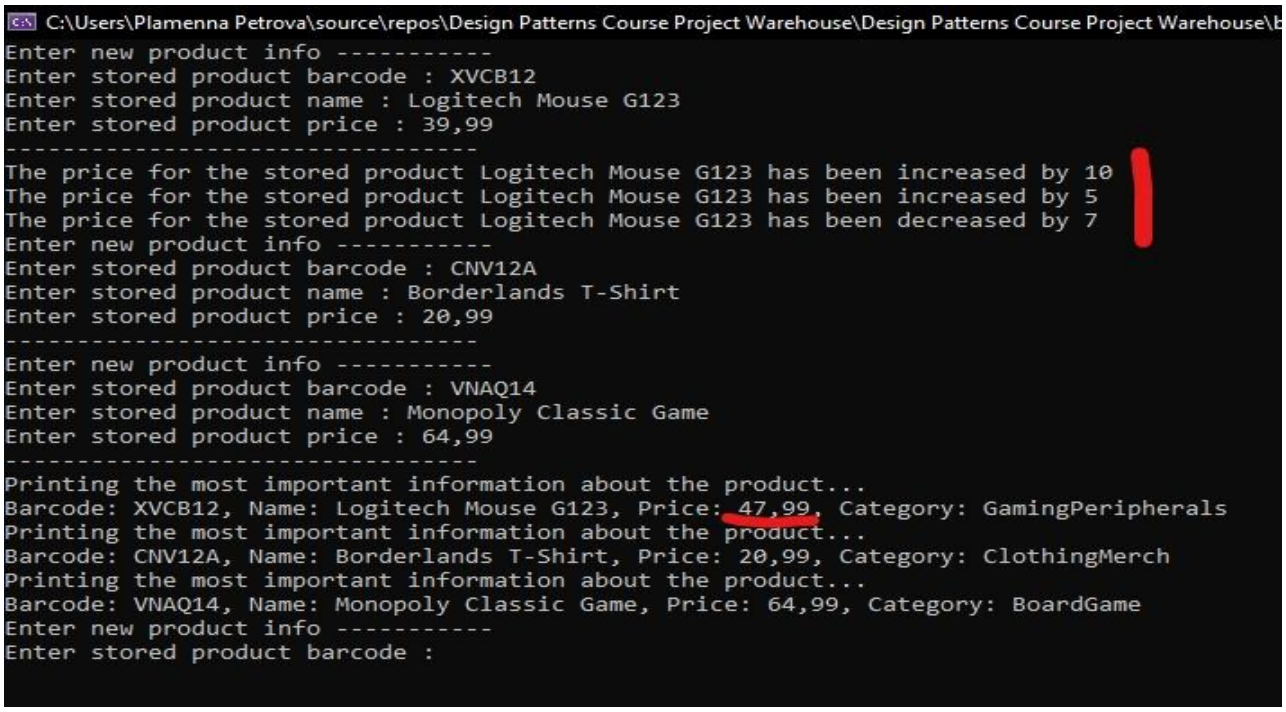
Program класът служи като Client както за шаблона Command така и за шаблона Adapter.

```
0 references
public static void Main(string[] args)
{
    // Factory
    IStoredProductCreator storedProductCreator = new GamingPeripheralsCreator();
    var logitechMouse = storedProductCreator.CreateStoredProduct();
    logitechMouse.Description = "A highly innovative mouse with 16 000 DPI";

    // Command
    var warehouse = new Warehouse();
    var warehouseInvoker = new WarehouseInvoker();
    CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, logitechMouse));
    CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Increase, logitechMouse, 10));
    CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Increase, logitechMouse, 5));
    CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Decrease, logitechMouse, 7));

    storedProductCreator = new ClothingMerchCreator();
    var borderlandsTShirt = storedProductCreator.CreateStoredProduct();
    borderlandsTShirt.Description = "A T-Shirt featuring the Claptrap robot";
    CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, borderlandsTShirt));

    storedProductCreator = new BoardGameCreator();
    var monopolyClassic = storedProductCreator.CreateStoredProduct();
    monopolyClassic.Description = "The Classic Monopoly game";
    CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, monopolyClassic));

    CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Print));
```

На горния code snippet е показано инстанцирането на обект от тип IStoredProductCreator (Factory Creator) чрез полиморфизъм на ConcreteCreator класовете. Един и същи Creator обект се създава няколко пъти, но спада към различен ConcreteCreator клас. Хардкоднато е Description пропъртито на обектите, създадени посредством извикването на CreateStoredProduct() метода на Creator-а. Инстанцира се и обект от тип Warehouse и друг от тип WarehouseInvoker. С помощта на Helper метода ExecuteCommand от CommandHelper-класа се посочват кои команди да бъдат изпълнени за StoredProduct (Factory Product) обектите – logitechMouse, borderlandsTShirt и monopolyClassic.

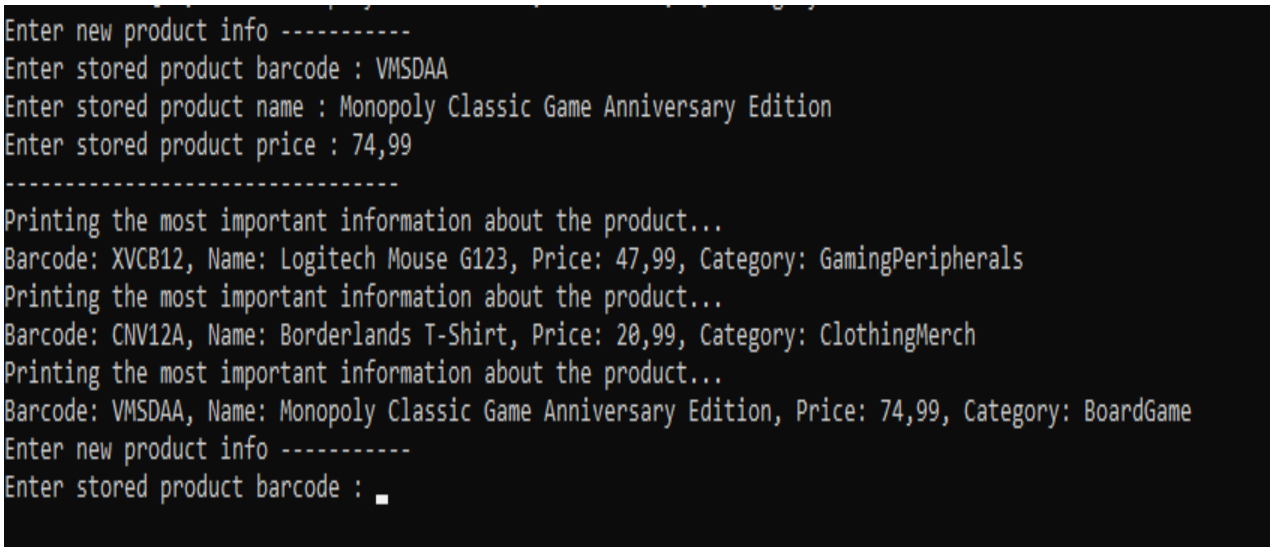
```
26 references
public class CommandHelper
{
    26 references
    public static void ExecuteCommand(WarehouseInvoker warehouseInvoker, IWarehouseCommand warehouseCommand)
    {
        warehouseInvoker.SetCommand(warehouseCommand);
        warehouseInvoker.Invoke();
    }
}
```

В ExecuteCommand се извикват два метода от Command Invoker-а – SetCommand() и Invoke(). Подават се току-що инстанцираният WarehouseInvoker обект и един от overload-натите WarehouseCommand конструктори. Последователно обектът logitechMouse се добавя в склада, цената му се увеличава с 10, с 5 и накрая се понижава с 7. Обектите borderlandsTShirt и monopolyClassic се добавят в склада. Принтира се най-важната информация за складираните продукти.



Демонстрирана е и Edit командата : Обектът monopolyClassic е заменен с новосъздадения обект monopolyClassicAnniversaryEdition. С команда Print се извежда промененият инвентар в склада.

```
storedProductCreator = new BoardGameCreator();
var monopolyClassicAnniversaryEdition = storedProductCreator.CreateStoredProduct();
monopolyClassicAnniversaryEdition.Description = "The Classic Monopoly with New Design for the Anniversary";
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Edit, monopolyClassic, monopolyClassicAnniversaryEdition));

CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Print));
```



Обектът aulaKeyboard се добавя към склада, след което се премахва от него. Принтират се междинните състояния на складовия инвентар с и без обекта :

```
storedProductCreator = new GamingPeripheralsCreator();
var aulaKeyboard = storedProductCreator.CreateStoredProduct();
aulaKeyboard.Description = "Aula Assault Keyboard";
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, aulaKeyboard));
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Print));
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Remove, aulaKeyboard));
Console.WriteLine("Printing without the last removed object");
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Print));
```

```
Enter new product info -----------
Enter stored product barcode : XMAWE3
Enter stored product name : Aula Keyboard
Enter stored product price : 79,87
--------------------------------
Printing the most important information about the product...
Barcode: XVCB12, Name: Logitech Mouse G123, Price: 47,99, Category: GamingPeripherals
Printing the most important information about the product...
Barcode: CNV12A, Name: Borderlands T-Shirt, Price: 20,99, Category: ClothingMerch
Printing the most important information about the product...
Barcode: VMSDAA, Name: Monopoly Classic Game Anniversary Edition, Price: 74,99, Category: BoardGame
Printing the most important information about the product...
Barcode: XMAWE3, Name: Aula Keyboard, Price: 79,87, Category: GamingPeripherals
Printing without the last removed object
Printing the most important information about the product...
Barcode: XVCB12, Name: Logitech Mouse G123, Price: 47,99, Category: GamingPeripherals
Printing the most important information about the product...
Barcode: CNV12A, Name: Borderlands T-Shirt, Price: 20,99, Category: ClothingMerch
Printing the most important information about the product...
Barcode: VMSDAA, Name: Monopoly Classic Game Anniversary Edition, Price: 74,99, Category: BoardGame
```

WarehouseInvoker обектът извиква UndoActions() метода и поетапно изтрива всички Warehouse обекти. Всъщност командите се анулират поетапно в обратен ред, докато не се изпразни складът.

```
warehouseInvoker.UndoActions();
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Print));
```

```
Adding Aula Keyboard
Removing Aula Keyboard
Undo edit
Removing Monopoly Classic Game Anniversary Edition
Adding Monopoly Classic Game
Removing Monopoly Classic Game
Removing Borderlands T-Shirt
The price for the stored product Logitech Mouse G123 has been increased by 7
The price for the stored product Logitech Mouse G123 has been decreased by 5
The price for the stored product Logitech Mouse G123 has been decreased by 10
Removing Logitech Mouse G123
```

Добавяне на четири нови обекти в склада – dakineBackpack, logitechHeadphones, scrabble, steeringWheel

```
storedProductCreator = new ClothingMerchCreator();
var dakineBackpack = storedProductCreator.CreateStoredProduct();
dakineBackpack.Description = "Patterned Dakine Backpack";
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, dakineBackpack));

storedProductCreator = new GamingPeripheralsCreator();
var logitechHeadphones = storedProductCreator.CreateStoredProduct();
logitechHeadphones.Description = "Logitech Headphones Newest Model, Blue, Fragile";
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, logitechHeadphones));

storedProductCreator = new BoardGameCreator();
var scrabble = storedProductCreator.CreateStoredProduct();
scrabble.Description = "The Classic Game Scrabble";
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, scrabble));

storedProductCreator = new GamingPeripheralsCreator();
var steeringWheel = storedProductCreator.CreateStoredProduct();
steeringWheel.Description = "Convenient Steering Wheel";
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Add, steeringWheel));
```

Следва инстанция на Adaptee обект IDelivery delivery = new Delivery(); Инстанцира се и обектът deliveryAdapter. Той принадлежи към Target типа IWarehouse, но при създаването се използва, конструктора на DeliveryAdapter класа, който приема delivery обекта като аргумент. Създава се и нов WarehouseInvoker обект за доставката.

```
// Adapter - Loading the truck for delivery
IDelivery delivery = new Delivery();
IWarehouse deliveryAdapter = new DeliveryAdapter(delivery);
var warehouseInvokerForAdapter = new WarehouseInvoker();
```

Execute-ват се Add команди за warehouseInvokerForAdapter. Вече има два Invoker-а – за целия склад и за отделените за доставка продукти. В overload-натите конструктори също за аргумент от тип IWarehouse се подава deliveryAdapter-ът. Foreach-ват се всички item-и в склада и ако те се съдържат в списъка на адаптера, се премахват от склада с команда Remove. Изписва се информацията за продуктите, останали в склада и за тези, които вече са нотоварени успешно на камиона за доставка.

```
// Adapter - Loading the truck for delivery
IDelivery delivery = new Delivery();
IWarehouse deliveryAdapter = new DeliveryAdapter(delivery);
var warehouseInvokerForAdapter = new WarehouseInvoker();
CommandHelper.ExecuteCommand(warehouseInvokerForAdapter, new WarehouseCommand(deliveryAdapter, WarehouseAction.Add, dakineBackpack));
CommandHelper.ExecuteCommand(warehouseInvokerForAdapter, new WarehouseCommand(deliveryAdapter, WarehouseAction.Add, logitechHeadphones));
CommandHelper.ExecuteCommand(warehouseInvokerForAdapter, new WarehouseCommand(deliveryAdapter, WarehouseAction.Add, scrabble));
foreach (var warehouseItem in warehouse.GetStoredProducts())
{
    if (deliveryAdapter.GetStoredProducts().Contains(warehouseItem))
    {
        CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Remove, warehouseItem));
    }
}
Console.WriteLine("Printing warehouse products to test");
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(warehouse, WarehouseAction.Print));
Console.WriteLine("Delivery information");
CommandHelper.ExecuteCommand(warehouseInvokerForAdapter, new WarehouseCommand(deliveryAdapter, WarehouseAction.Print));
```

```
--------------------------------
Printing warehouse products to test
Printing the most important information about the product...
Barcode: XDJ981, Name: Ultra Steering Wheel, Price: 199,99, Category: GamingPeripherals
Delivery information
Printing the most important information about the product...
Barcode: CKL123, Name: Dakine Backpack, Price: 89, Category: ClothingMerch
Printing the most important information about the product...
Barcode: XV67A5, Name: Logitech Headphones G332, Price: 98,99, Category: GamingPeripherals
Printing the most important information about the product...
Barcode: VYT34N, Name: Scrabble Board Game, Price: 44,5, Category: BoardGame
```

Четвъртият и последен шаблон, реализиран в приложението, се отнася до разпределянето на състоянието на продуктите : чупливи и нечупливи, като се търси се дали в описанието е включена думата ,,fragile” – чуплив. Инстанцира се Orginator обект и му се set-ва default-ен задава State “Non-Fragile”. Инстанцира се и Caretaker обект. Към пропъртито му WarehouseMemento се закача новосъздаден от Originator-а WarehouseMemento обект. Създава се и нов StoredProduct списък за чупливите продукти. Foreach-ват се всички продукти, съхранени в DeliveryAdapter-а. При условие, че думата “fragile’ е намерена при ToLower()-нат Description, State се променя на “Fragile” се увеличава цената на съответния продукт с 10 и той се добавя към List-а от чупливи продукти. В останалите случаи с метода SetWarehouseMemento, извикан от Originator обекта, се възстановява “Non-Fragile” State-ът. И в самия край на Program класа се принтират delivery item-ите заедно с нанесените промени както и продуктите които участват във fragileProducts списъка.

```
// Memento
Originator originator = new Originator();
Console.WriteLine("Memento: ");
Console.WriteLine("Printing initial state : ");
originator.State = "Non-Fragile";
Caretaker caretaker = new Caretaker();
caretaker.WarehouseMemento = originator.CreateWarehouseMemento();
string stateToFind = "fragile";
List<StoredProduct> fragileProducts = new List<StoredProduct>();
foreach (var deliveryItem in deliveryAdapter.GetStoredProducts())
{
    if (deliveryItem.Description.ToLower().Contains(stateToFind)) {
        originator.State = "Fragile";
        Console.WriteLine("Increasing price for the delivery of a fragile product");
        CommandHelper.ExecuteCommand(warehouseInvokerForAdapter, new WarehouseCommand(deliveryAdapter, WarehouseAction.Increase, deliveryItem, 10));
        fragileProducts.Add(deliveryItem);
    }
    else
    {
        originator.SetWarehouseMemento(caretaker.WarehouseMemento);
    }
}
CommandHelper.ExecuteCommand(warehouseInvoker, new WarehouseCommand(deliveryAdapter, WarehouseAction.Print));
Console.WriteLine("Printing fragile items");
foreach (var fragileItem in fragileProducts)
{
    Console.WriteLine(fragileItem.Name);
}
```

В конзолата :

```
Memento:
Printing initial state :
State = Non-Fragile
Restoring state
State = Non-Fragile
State = Fragile
Increasing price for the delivery of a fragile product
The price for the product that will be delivered Logitech Headphones G332 has been increased by 10
Restoring state
State = Non-Fragile
Printing the most important information about the product...
Barcode: CKL123, Name: Dakine Backpack, Price: 89, Category: ClothingMerch
Printing the most important information about the product...
Barcode: XV67AS, Name: Logitech Headphones G332, Price: 108,99, Category: GamingPeripherals
Printing the most important information about the product...
Barcode: VYT34N, Name: Scrabble Board Game, Price: 44,5, Category: BoardGame
Printing fragile items
Logitech Headphones G332
```