

Технически университет – Варна

**Курсов проект 9**

по ООП – I част

*Програма реализираща информационна система за обслужване на склад*

Изготвил: Тодор Иванов Йорданов Ръководител: доц. Х.Ненов

Спец: СИТ

Курс: II

Група: 4

Ф.Н: 20621600

**1.Увод**

**1.1.Описание и идея на проекта**

Да се напише програма, реализираща информационна система, обслужваща склад. Програмата съхранява и обработва данните за наличността в склада във файл. За всеки продукт се съхранява информация(име, срок на годност, дата на постъпване в склада, име на производител, мерна единица/килограми,литри/, налично количество, местоположение/секция-рафт-номер/, коментар/свободен текст/*)*

**1.2.Цел и задачи на разработката**

След като приложението отвори даден файл(**.xml**), то трябва да може да извършва посочените по-долу операции, в допълнение на общите (open, close, save, save as, help, exit):

|  |
| --- |
| **print –** Извежда информация за наличните продукти в склада |
| **add –** Добавя нов продукт в склада в **диалогов режим**, като въвежда всички данни за продукта и ги проверява за коректност. При добавянето на продукта се спазват следните правила:  ● ако нов продукт е с различен срок на годност от вече съществуващ едноименен продукт, той да бъде поставен на различно място  ● ако имате достатъчно място, еднакви продукти с един и същи срок на годност да бъдат поставени на едно и също място  ● при извеждане на списъка с налични продукти да се изведе общото количество на едноименните продукти, независимо от срока им на годност |
| **remove** **–** Изважда продукт от склада в **диалогов режим**:  ● по дадено име и количество изважда съответните продукти от склада и извежда информация за продукта и къде се е намирал  ● при наличие на повече от една партида, първо намалява тази със най-скоро изтичащ срок на годност, тогава във информацията за извършеното действие се отбелязва количеството и мястото на всяка от партидите, които сме намалили  ● в случай на опит за изваждане на повече от наличното да се дава информация на потребителя за наличността на продукта и срока на годност на партидите и възможност да реши дали все пак не иска да извади това, което е останало |
| **log <from> <to>** **–** Извежда справка за всички промени в наличността в периода от дата <from> до дата <to>, включително зареждания и извеждания на стоки |
| **clean –** Разчиства склада от всички стоки, на които е изтекъл или предстои скоро да изтече срока но годност (до 7 дни), като извежда информация за разчистените стоки |

**1.3.Структура на документацията**

1. Увод

2. Преглед на предметната област

3. Проектиране

4. Реализация, тестване

5. Заключение

**2.Преглед на предметната област**

**2.1.Основни дефиниции, концепции и алгоритми, които ще бъдат използвани**

В проекта съм приложил *Singleton Pattern* при реализацията на основния клас (StorageSystem) касаещ целия проект. С това съм гарантирал, че колкото и пъти потребителя да стартира приложението, винаги ще се работи само с една инстанция на класа StorageSystem. Също така, съм се опитал да приложа част от *SOLID* принципите с цел по-добра структурираност и разбираемост на кода. Освен това, за методите **add** и **remove** съм използвал **GUI**(Графичен потребителски интерфейс) или по-точно **Java Swing**, защото по условие има изискване изброените 2 метода по-горе да бъдат реализирани в “**диалогов режим**”, за разлика от всички останали методи, към които няма определени изисквания и по подразбиране съм реализирал в конзолен режим. Също така съм използвал и **JAXB**(Java Architecture for XML Binding) за запис(marshal) и четене(unmarshal) в/от файл с разширение “**.xml**”. Добавил съм и допълнителна библиотека към проекта за време (**joda-time-2.10.14.jar**), която използвам вместо стандартната библиотека за време, защото записа на дати в “.xml”-файлове е често срещан проблем и в моя случай намерих решение на този проблем именно благодарение на горе споменатата библиотека за време. Но за да бъде приложена успешно в проекта, се наложи да използвам клас “**LocalDateAdapter**”, за да осигуря на JAXB необходимия “mapping” на данните за време/дата, тъй като “joda-time-2.10.14.jar” e допълнително добавена библиотека към проекта от мен както вече споменах. Освен това в два от пакетите(“*xml”*,”*bg.tu\_varna.sit”*) на проекта съм регистрирал на пакетно ниво (**package-info.java)** споменатия по-горе “LocalDateAdapter”. Това съм направил, за да може атрибути/полета/методи, които са от тип принадлежащ на допълнително добавената библиотека(“joda-time-2.10.14.jar”) да бъдат разпознати и приложени успешно.

**2.2.Функционални изисквания и подходи за спазването им**

Всеки един продукт(партида) по условие трябва да има местоположение(секция-рафт-номер) в склада. В условието на проекта конкретно не е упоменато колко количество стока може да се съдържа общо в склада или на конкретно място,но в крайна сметка капацитета не може да бъде безкраен, затова съм задал склада да има 4 сектора(“A”,”B”,”C”,”D”) като всеки сектор има по 10 рафта(1-10) и всеки рафт има по 10 номера(1-10). Задал съм максимален капацитет за всяко място от 1000 кг/л. Като общия капацитет на склада е 400 000 кг/л. Изискванията от всички методи са спазени като съм задал съответните ограничения, които задължават потребителя също да ги спазва при използване на приложението.

**3.Проектиране**

**3.1. Обща структура на проекта (пакети, които са реализирани)**

С цел по-добра структурираност и разбираемост на кода съм разпределил всички класове в 6 различни пакета (**bg.tu\_varna.sit** , **dialogs** , **options** , **xml** , **exceptions** , **META-INF**)

|  |
| --- |
| ● **bg.tu\_varna.sit –** В този пакет се съдържат почти всички основни класове и интерфейси, включително и входната точка на приложението (класът *Main*).  ~ класове: package-info.java , Location , Product , RemovedProduct , Warehouse , StorageHistory , GeneralMenu , StorageSystem , Main ;  ~ интерфейси: XmlAccess , OpenFile , CloseFile , SaveAs , Help , Print , Add , Remove, Log , Clean , Menu ; |

|  |
| --- |
| ● **dialogs –** В този пакет съм сложил форми за добавяне и изваждане на продукт/и от склада в **диалогов режим** както се изисква по условие за методите **add** и **remove**. Съответните форми съм реализирал посредством **Java Swing**, като към всяка една от формите си има и автоматично генерирани класове за работа с тях, към които съм добавил доста функционалности. Също така към пакета съм добавил и още 2 класа за валидиране на входните данни въведени от потребителя във формите, с цел спазване на всички изисквания, зададени предварително в условието на проекта.  ~ клас и форма за метод **add**: DialogAdd , DialogAdd.form ;  ~ клас и форма за метод **remove**: DialogRemove , DialogRemove.form ;  ~ класове за валидиране на данни: ValidateDialogAdd , ValidateDialogRemove ; |

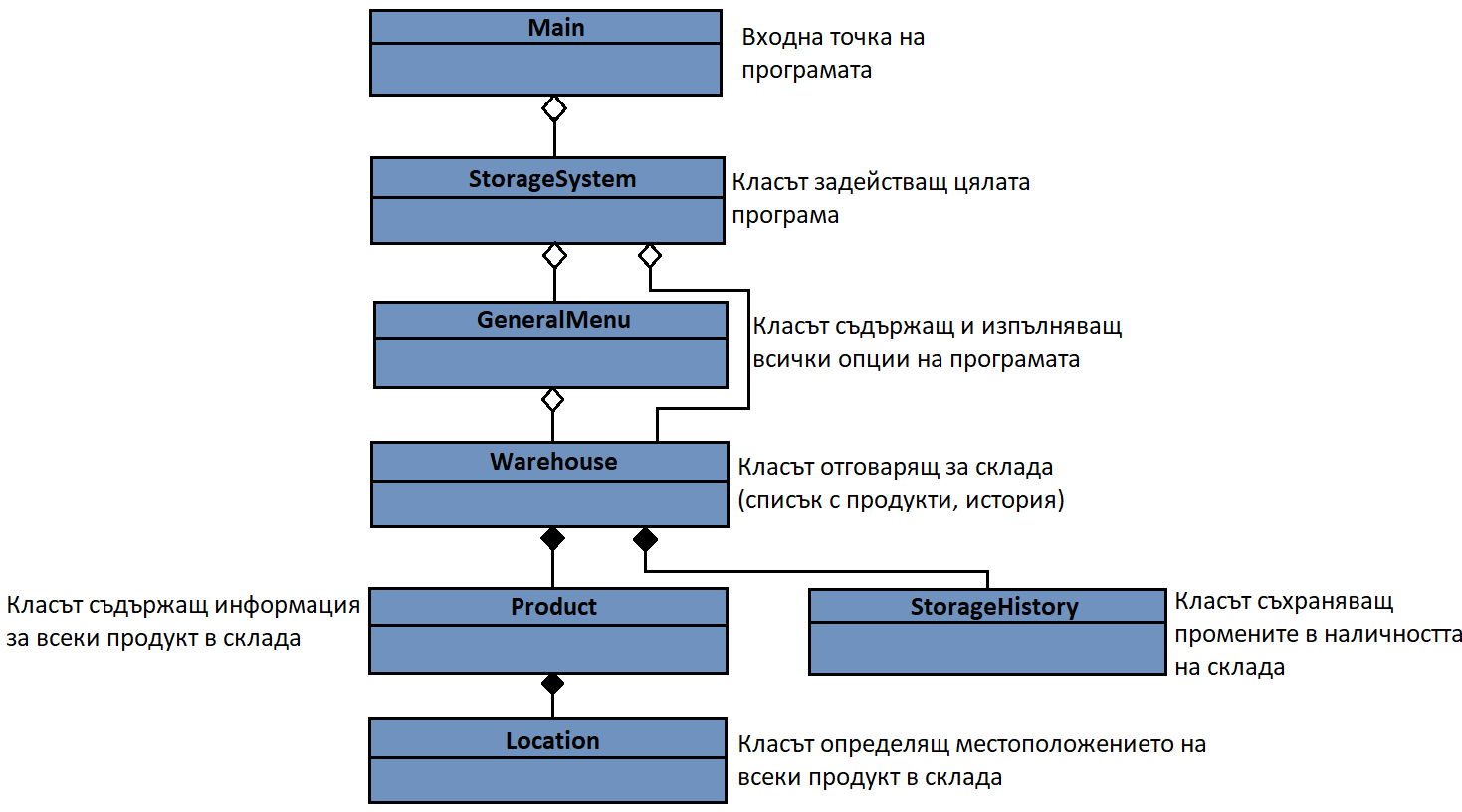
|  |
| --- |
| ● **options –** Този пакет обхваща всички класове, реализиращи методите на работа с командния ред и методите на самия проект.  ~ класове: OpenFile , CloseFile , SaveAs , Help , Print , Add , Remove , Log , Clean ; |

|  |
| --- |
| ● **xml –** В този пакет се съдържат класове, които са изцяло предназначени за работа с “.xml”-файлове и по-специфичните типове данни(дата/време), които ще бъдат записвани и четени в/от тях. По конкретно, предназначението на един от класовете (XmlAccess) е запис(marshal) и четене(unmarshal) в/от “.xml”-файлове посредством JAXB(Java Architecture for XML Binding). В друг клас (LocalDateAdapter) съм разписал определена логика, за да осигуря на JAXB необходимия “mapping” на данните за време/дата тъй като използвам външна библиотека (joda-time-2.10.14.jar). В “package-info.java” съм регистрирал споменатия по-горе клас “LocalDateAdapter”, за да може атрибути/полета/методи, които са от тип принадлежащ на допълнително добавената библиотека(“joda-time-2.10.14.jar”) да бъдат разпознати и приложени успешно.  ~ класове: XmlAccess , LocalDateAdapter , package-info.java ; |

|  |
| --- |
| ● **exceptions –** Този пакет обхваща всички класове, представляващи “custom” грешки, които хвърлям(“throw”) при грешно въведени данни от потребителя във формите за добавяне/изваждане на продукт/и от склада или в опциите от командния ред.  ~ класове: DateException , EmptyFieldException , LocationException , MissingProductException , OptionException , QuantityException , UnitException ; |

|  |
| --- |
| ● **META-INF – “**The META-INF folder is the home for the MANIFEST. MF file. This file **contains meta data about the contents of the JAR**. For example, there is an entry called Main-Class that specifies the name of the Java class with the static main() for executable JAR files.**”**  Източник: <https://www.google.com/search?q=what+is+meta+inf+package+for&oq=What+is+meta+in&aqs=chrome.0.69i59j69i57j0i512l8.5650j0j7&sourceid=chrome&ie=UTF-8> |

**3.1. Блок схема (основна структура) на приложението**

****

**4.Реализация, тестване**

Класът **Location** определя местоположението на всеки продукт/и в склада. Състои се от 3 броя private атрибути, отговарящи за сектора, рафта и номера на всеки един от продуктите. Класът има 2 конструктора – един по подразбиране и един експлицитен, съдържащ атрибутите на класа. Съответно има методи за извличане и задаване стойностите на атрибутите. Освен това класът има 2 метода за “стрингова” репрезентация. Също така класът съдържа и 2 метода - “equals” и “hashCode” за 3те атрибути на класа(sector, shelf, number). Освен всичко изброено съм добавил и анотации към класа, защото програмата работи с “.xml”-файлове.

package bg.tu\_varna.sit;

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

import java.util.Objects;

@XmlRootElement(name = "Location")

@XmlType(propOrder = { "sector", "shelf","number"})

public class Location {

private String sector;

private int shelf;

private int number;

public Location(String sector, int shelf, int number) {

this.sector = sector;

this.shelf = shelf;

this.number = number;

}

public Location(){}

…

Класът **Product** съдържа информация за всеки един от продуктите в склада.( име, срок на годност, дата на постъпване в склада, име на производител, мерна единица/килограми,литри/, налично количество, местоположение/секция-рафт-номер/, коментар/свободен текст/). Класът има 2 конструктора – един по подразбиране и един експлицитен, съдържащ атрибутите на класа(productName, expiryDate, EntryDate, manufacturer, unit, quantity, **location**, comment). Съответно има методи за извличане и задаване стойностите на атрибутите. Освен това класът има 2 метода за “стрингова” репрезентация. Също така класът съдържа и 2 метода - “**equals**” и “**hashCode**”, с които задавам еднаквост на продуктите по : **productName**, **expiryDate**, **entryDate**, **manufacturer** и **unit**. Освен всичко изброено съм добавил и анотации към класа, защото програмата работи с “.xml”-файлове.

package bg.tu\_varna.sit;

**import org.joda.time.LocalDate;**

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

import java.util.Objects;

@XmlRootElement(name = "Product")

@XmlType(propOrder = { "productName", "expiryDate","entryDate","manufacturer","unit","quantity","location","comment"})

public class Product {

private String productName; //ime na produkta

private LocalDate expiryDate; //srok na godnost

private LocalDate entryDate; //data na postupvane v sklada

private String manufacturer; //ime na proizvoditel

private String unit; //merna edinica

private double quantity; //kolichestvo na produkta

private Location location; //mestopolojenie na produkta

private String comment; //komentar

public Product(String productName, LocalDate expiryDate, LocalDate entryDate, String manufacturer, String unit, double quantity, Location location, String comment) {

this.productName = productName;

this.expiryDate = expiryDate;

this.entryDate = entryDate;

this.manufacturer = manufacturer;

this.unit = unit;

this.quantity = quantity;

this.location = location;

this.comment = comment;

}

public Product(){}

…

Класът **StorageHistory** е предназначен да съхранява информация за общото количество заредена и изведена стока от склада за период от време, определен от потребителя. За целта съм използвал две колекции от тип “Map” или по-конкретно “LinkedHashMap” като едната използвам за съхранение общото количество добавена стока, а другата за общото количество извадена стока от склада. Класът има само конструктор по подразбиране и съответните методи за извличане и задаване стойностите на атрибутите(двете колекции от тип “LinkedHashMap”). Освен това, класът съдържа и два по-особени метода, чиято цел е да отразяват промените в наличността на склада при добавяне и изваждане на продукт/и. Също така съм добавил и анотации към класа, защото програмата работи с “.xml”-файлове.

Класът **Warehouse** e един от основните класове в цялата програма. Този клас съдържа в себе си колекция от тип “Map” или по-конкретно “LinkedHashMap”. Колекцията пази в себе си информация за всички продукти в склада както и тяхното местоположение. Също така, класът има атрибут, който е инстанция на класа StorageHistory, за да може класът Warehouse да съдържа информация за количеството добавена и извадена стока от склада. Освен това, класът съдържа в себе си и съответните методи за задаване и извличане стойността на атрибутите на класа. Също така има и метод за “стрингова” репрезентация. Не на последно място, в този клас също съм използвал и анотации, защото програмата работи с “.xml”-файлове.

package bg.tu\_varna.sit;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlRootElement;

import java.util.LinkedHashMap;

import java.util.Map;

@XmlRootElement(name="Warehouse")

@XmlAccessorType(XmlAccessType.FIELD)

public class Warehouse {

private Map<Location,Product> productList = new LinkedHashMap<>();

private StorageHistory storageHistory = new StorageHistory();

…

Класът **GeneralMenu** e един от основните класове в програмата. Този клас съдържа в себе си 3 private атрибута. Единият от тях е инстанция на класа Warehouse, а останалите 2 атрибута са: за име на файла, с който ще работи програмата и дали въобще има отворен файл, с който се работи. Идеята на самия клас е той да изпълнява чрез методи, всички опции, които са зададени в условието на проекта (методи за работа с командния ред + методи по същината на условието). Класът има отделен метод за изпълнение на всяка една команда от програмата. Всеки един от тези методи (open(…),close(),save(),saveas(…),help(),print(),addProduct(),removeProduct,log(…),clean()) създава в себе си инстанция от съответния клас, който отговаря за дадената опция и извиква определената за целта функция, като класовете, които се ползват за изпълнението на опциите съм поместил в пакетите **options** и **dialogs**, които ще разгледам в следващите страници. Също така има и метод “start” обединяващ в себе си всички методи на класа и при подадена команда от потребителя в командния ред, се извиква един от по-горе изброените методи и съответната операция се изпълнява. Ако потребителя въведе грешни данни се продуцира съответната грешка в зависимост дали проблема се отнася за името на самата команда или за изпълнението на съответния метод.

public class GeneralMenu implements Menu {

private Warehouse warehouse;

private String file;

private boolean isFileOpened=false;

public GeneralMenu(Warehouse warehouse) {

this.warehouse=warehouse;

file = null;

}

@Override

public void start(String[] options)throws OptionException

{ switch (options[0])

{

case "open": open(options);break;

case "close": close();break;

case "save": save();break;

case "saveas": saveAs(options);break;

case "help": help();break;

case "print": print();break;

case "add": addProduct();break;

case "remove": removeProduct();break;

case "log": log(options);break;

case "clean": clean();break;

default: throw new OptionException("Invalid option!");

}}

**Методите, които добавят и премахват продукти от склада:**

private void addProduct()  
{  
 DialogAdd dialogAdd = new DialogAdd(warehouse);  
}  
  
private void removeProduct()  
{  
 if(warehouse.getProductList()==null || warehouse.getProductList().isEmpty())  
 {  
 System.*out*.println("There are no products in the warehouse!");  
 }  
  
 else {DialogRemove dialogRemove = new DialogRemove(warehouse);}  
}

Класът **StorageSystem** e може би главният клас на програмата обединяващ в себе си всички компоненти и задействащ цялото приложение. Съдържа 2 private атрибута, които са инстанции на класове. Едната инстанция е от самия клас StorageSystem, защото в класа съм приложил “**Singleton Pattern**”. Другата инстанция е на класа Warehouse. Класът си има private конструктор и статичен метод, който връща обект от типа на самия клас StorageSystem, с цел прилагане на Singleton Pattern-а. Също така има метод “**launch()**”, който е по-особен. В метода “launch()” създавам инстанция на интерфейса Menu, която инстанция си е всъщност на класа GeneralMenu, защото класа GeneralMenu имплементира интерфейса Menu. След това посредством “Scanner” прочитам входа на потребителя и в зависимост какъв е входа се извършва едно от следните действия:

● в случай, че потребителя е въвел “exit” програмата прекратява работата си

● в случай, че потребителя е въвел грешна командна се продуцира грешка за невалидна опция

● в случай, че потребителят е въвел правилна команда, се извиква метода “start(options)” на обекта от тип GeneralMenu като аргумент на метода се подава съответната командна(“options”), която се подава надолу по йерархията от класовете, които вече изброих и се изпълнява съответната опция, която потребителя е пожелал.

public class StorageSystem {  
  
 private static StorageSystem *instance*;  
 private Warehouse warehouse;  
  
 private StorageSystem(){  
 warehouse = new Warehouse();  
 }  
  
 public static StorageSystem getInstance()  
 {  
 if(*instance*==null) {*instance* = new StorageSystem();}  
 return *instance*;  
 }  
  
 public void launch()  
 {  
 Menu menu = new GeneralMenu(warehouse);  
  
 Scanner scanner = new Scanner(System.*in*);  
  
 String input;  
 String[] options;  
  
 while(true)  
 {  
 System.*out*.print('>');  
 input = scanner.nextLine();  
  
 if(input.equals("exit")) break;  
  
 options = input.split(" ", 2);  
  
 try  
 {  
 menu.start(options);  
 }  
 catch (OptionException customException)  
 {  
 System.*out*.println(customException.getMessage());  
 }  
 }  
 }  
  
}

Класът **Main** е входната точка на програмата. В него единствено създавам инстанция на класа **StorageSystem** и извиквам метода му “**launch()**”, чрез който задействам изпълнението на цялата програма.

public class Main {  
  
 public static void main(String[] args) {  
  
 StorageSystem storageSystem = StorageSystem.*getInstance*();  
 storageSystem.launch() }  
}

**В следващите няколко страници ще се опитам да направя подробни разяснения относно логиката на някои от класовете, които съм използвал с цел реализация на основните методи в програмата.**

В пакетът **dialogs** както вече споменах съм поместил класовете и формите отговарящи за добавянето и премахването на продукт/и в диалогов режим. Съответните класове и форми съм реализирал посредством **Java Swing**.

Класът **DialogAdd** е за диалоговият прозорец, който се появява, когато потребителя иска да добави продукт/и в склада. Класът има 2 публични конструктора. Един по подразбиране и експлицитен. Експлицитния конструктор приема като аргумент обект от класа “Warehouse”. В самия експлицитен конструктор първоначално задавам параметрите на диалоговия прозорец:

setContentPane(panelAdd);  
setTitle("Please add product");  
setBounds(700, 200, 500, 570);  
setVisible(true);  
setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);

След това на обекта, който е от типа на бутона за добавяне на продукт/и в диалоговия прозорец, се изпълнява метод “**addActionListener**”. След което, вътре в този метод “override”-вам метода “**public void actionPerformed(ActionEvent event)”** като вътре в този метод в един масив от тип “String” извличам данните от всички полета на диалоговия прозорец, които потребителя е въвел и ги проверявам за коректност. Ако всичко е наред създавам обект от тип клас “**Add**”, който съм поместил в пакета **options** и му викам метод “add()”, чрез който добавям продукт/и в склада. **Забранил съм копирането и поставянето на данни в полетата на формата с цел да не се счупи програмата ако случайно потребителя иска да въведе прекалено голям обем от данни.**

Класът **DialogRemove** е за диалоговия прозорец, който се появява, когато потребителя иска да премахне продукт от склада. Класът има 2 публични конструктора. Един по подразбиране и експлицитен. Експлицитния конструктор приема като аргумент обект от класа “Warehouse”. В самия експлицитен конструктор първоначално задавам параметри на диалоговия прозорец:

setContentPane(panelRemove);  
setTitle("Please remove product");  
setBounds(650, 400, 550, 200);  
setVisible(true);  
setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);

След това на обекта, който е от типа на бутона за премахване на продукт/и в диалоговия прозорец, се изпълнява метод “**addActionListener**”. След което, вътре в този метод “override”-вам метода “**public void actionPerformed(ActionEvent e)”** като вътре в този метод в един масив от тип “String” извличам данните от всички полета на диалоговия прозорец, които потребителя е въвел и ги проверявам за коректност. Ако всичко е наред създавам обект от тип клас “**Remove**”, който съм поместил в пакета **options** и му викам метод “remove()”, чрез който премахвам продукт/и от склада. **Забранил съм копирането и поставянето на данни в полетата на формата с цел да не се счупи програмата ако случайно потребителя иска да въведе прекалено голям обем от данни.**

Източници:

<https://www.youtube.com/watch?v=aIdIXsi1qTU&list=LL&index=30&t=45s>

<https://www.youtube.com/watch?v=ffVY6HGtxH8&list=LL&index=5>

В пакетът “**options**” се съдържат основните методи на програмата под формата на класове

Всеки един от следните класове е уникален сам по себе си:

-**Add** (за добавяне на продукт/и в склада)

-**Clean** (за премахване на продукт/и от склада, които са с изтекъл срок на годност или скоро предстои да изтече /до 7 дни/)

-**CloseFile** (за затваряне на “.xml”-файла, с който потребителя работи)

-**Help** (за извеждане на всички команди, които програмата поддържа)

-**Log** (за извеждане на справка за всички промени в наличността в зададен от потребителя период)

-**OpenFile** (за отваряне на “.xml”-файла, с който потребителя иска да работи)

-**Print** (за извеждане на наличните продукти в склада)

-**Remove** (за премахване на продукт/и от склада)

-**SaveAs** (за записване на данните въведени от потребителя в “.xml”-файл)

По мое мнение най-комплексните класове в програмата са именно : **Add** и **Remove**, защото за тяхната реализация трябваше да спазя най-много условия, да направя най-много проверки и да огранича свободата на потребителя до степен, че това което бъде въведено като данни във формите да спазва основните изисквания на проекта. Всеки един от тези два класа има само по 1 метод изпълняващ само 1 задача, а именно добавяне и премахване на продукт/и в/от склада. Методите са доста дълги и вероятно трябваше да ги разделя на под-методи, но тогава логиката щеше да се “размие” и последователността на действията да стане по-неясна.

В пакетът “**xml**” се съдържат основните класове, които определят логиката на работа с “.xml”-файлове за цялата програма

-**XmlAccess –** клас предназначен за работа с “.xml”-файлове. Класът има два метода, един за запис(marshal) и един за четене(unmarshal) в/от файл с разширение “**.xml**”. Както вече споменах използвам **JAXB**(Java Architecture for XML Binding) за тази цел.

//marshal (write to) XML  
public void saveToFile(Warehouse warehouse, String fileName) throws JAXBException  
{  
 JAXBContext context = JAXBContext.*newInstance*(Warehouse.class);  
 Marshaller mar = context.createMarshaller();  
 mar.setProperty(Marshaller.*JAXB\_FORMATTED\_OUTPUT*, true);  
 mar.marshal(warehouse, new File(fileName));  
}  
  
//unmarshal (read from) XML  
public Warehouse loadFromFile(String fileName) throws JAXBException, IOException  
{  
 JAXBContext context = JAXBContext.*newInstance*(Warehouse.class);  
 return (Warehouse) context.createUnmarshaller().unmarshal(new FileReader(fileName));  
}

Източник: <https://www.baeldung.com/jaxb>

-**LocalDateAdapter** **–** клас, който както вече споменах в началото използвам, за да осигуря на JAXB необходимия “mapping” на данните за време/дата, тъй като използвам библиотека “joda-time-2.10.14.jar”, която e допълнително добавена към проекта от мен както вече споменах.

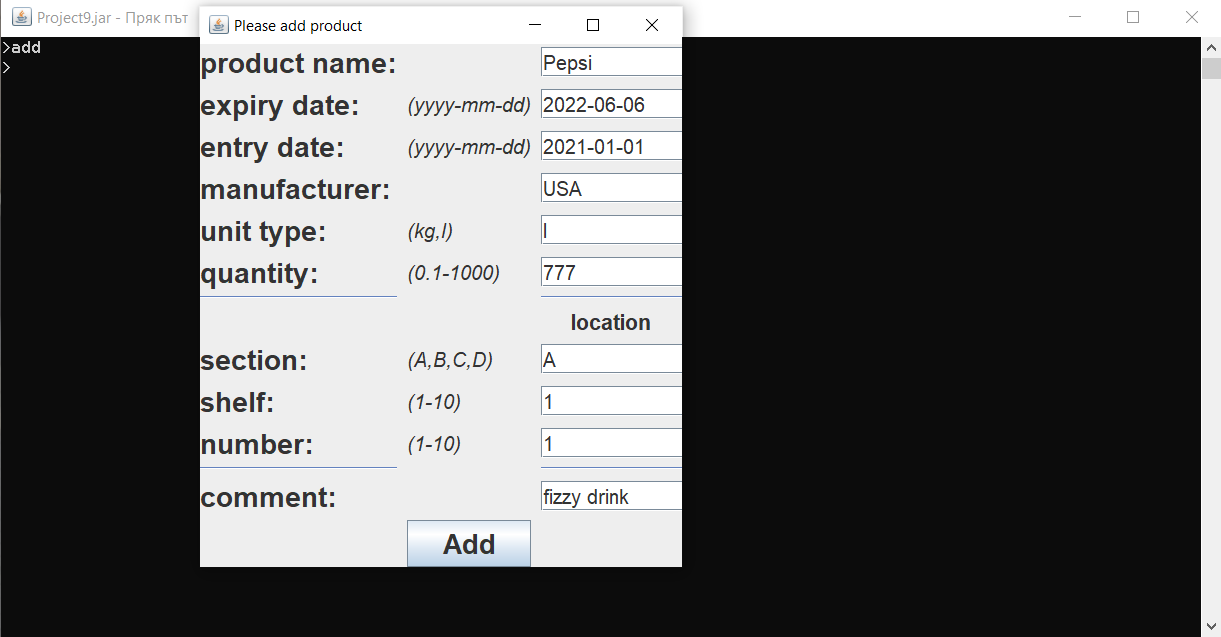
import org.joda.time.LocalDate;  
  
import javax.xml.bind.annotation.adapters.XmlAdapter;  
  
public class LocalDateAdapter extends XmlAdapter<String, LocalDate> {  
  
 public LocalDate unmarshal(String v) throws Exception {  
 return LocalDate.*parse*(v);  
 }  
  
 public String marshal(LocalDate v) throws Exception {  
 return v.toString();  
 }  
}

- **package-info.java** **–** клас, който съществува в два от пакетите на проекта (“*xml”*,”*bg.tu\_varna.sit”*). Както споменах още в началото, “package-info.java” има за цел да регистрира на пакетно ниво споменатия по-горе клас “**LocalDateAdapter**”, за да може атрибути/полета/методи, които са от тип принадлежащ на допълнително добавената библиотека(“**joda-time-2.10.14.jar**”) да бъдат разпознати и приложени успешно.

@XmlJavaTypeAdapters({  
  
 @XmlJavaTypeAdapter(type= LocalDate.class, value=LocalDateAdapter.class)  
  
})  
  
package xml;  
  
import org.joda.time.LocalDate;  
  
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;  
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapters;

**Тестване на опциите на програмата**

**add**

****

**print**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**remove**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**clean**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**log <from> <to>**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**open**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**save**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**saveas**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**close**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**help**

**Картина, която съдържа текст

Описанието е генерирано автоматично**

**exit-** програмата прекратява работа

**5.Заключение**

Програмата изпълнява всички операции зададени в условието на проекта. Стартира се при зареждане на файла “Project9.jar - Пряк път”, който се намира в папката на проекта.(За всеки случай в папка “LaunchProgram” съм поставил и файл “Launch Project9.bat” ако програмата не тръгне при избор на файла “Project9.jar”) Вероятно е възможна и по-добра реализация на списъка от продукти в склада, чрез колекция различна от “Map”, просто аз реших да използвам съответната колекция. Също така, съм се опитал максимално да оптимизирам работата на цялата програма като както вече споменах съм приложил **Singleton Pattern**, **S.O.L.I.D.** принципите, **Java Swing**, **JAXB** и външната библиотека “**joda-time-2.10.14.jar**”. Структурирал съм пакетите и класовете на програмата по такъв начин, че да бъде възможно безпроблемно добавяне на нови функционалности и подробности. За бъдещо подобрение на програмата бих предложил усъвършенстване на метода “log” и класа “StorageHistory” касаещи воденето на справки за всички промени в наличността на склада в период зададен от потребителя, включително зареждания и извеждания на стоки.

Връзка към хранилището ми в **GitHub**:

<https://github.com/yordanov0502/OOP1-project>

Източници:

<https://www.youtube.com/watch?v=aIdIXsi1qTU&list=LL&index=29&t=45s>

<https://www.baeldung.com/jaxb>

<https://stackoverflow.com/questions/19966394/how-to-open-run-jar-file-double-click-not-working>

<https://stackoverflow.com/questions/4157972/how-to-update-a-value-given-a-key-in-a-hashmap>

<https://stackoverflow.com/questions/11430022/why-is-a-default-constructor-required-for-objects-exported-by-a-jax-ws>

<https://www.oracle.com/technical-resources/articles/javase/jaxb.html>

<https://beginnersbook.com/2017/10/java-localdate-compareto-method-example/>

<https://howtodoinjava.com/java/sort/java-sort-map-by-key/>

<https://www.youtube.com/watch?v=ffVY6HGtxH8&list=LL&index=5>

<https://howtodoinjava.com/jaxb/jaxb-example-marshalling-and-unmarshalling-hashmap-in-java/>

<https://www.javatpoint.com/uml-relationship>

<https://dzone.com/articles/jaxb-and-joda-time-dates-and>

накратко небезизвестните сайтове и форуми :

● Stack Overflow

● java T point

● Geeks for Geeks

● HowToDoInJava

● Baeldung

● BeginnersBook

● DZone