

СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ”
Факултет по математика и информатика



Обектно – ориентирано програмиране
Курсов проект на тема:
„Склад“

Изготвила:	Весела Илиянова Стоянова
Ф.Н.	71949
I курс	Бакалавър
Специалност:	Информационни системи
Ръководител:	Калин Георгиев

07.06.2020 г.

Съдържание:

1. Увод:	3
1.1. Описание и идея на проекта:	3
1.2. Цел и задачи на разработката:	3
1.3. Структура на документацията:	3
2. Преглед на предметната област:	3
2.1. Основни дефиниции, концепции и алгоритми, които ще бъдат използвани:	3
2.2. Дефиниране на проблеми и сложност на поставената задача:	4
2.3. Подходи , методи (евентуално модели и стандарти) за решаване на поставените проблеми:	4
2.4. Потребителски (функционални) изисквания (права, роли, статуси, диаграми) и качествени (нефункционални) изисквания (скалируемост, поддръжка):	4
3. Проектиране:	5
3.1. Общата архитектура на проекта е представена на фиг. 3:	5
3.2. Диаграми (на структура и поведение – по обекти, слоеве с най – важните извадки от кода):	6
4. Реализация, тестване:	8
4.1. Реализация на класове:	8
4.2. Планиране, описание и създаване на тестови сценарии:	12
5. Заключение:	12
5.1. Обобщение на изпълнението на началните цели:	12
5.2. Насоки за бъдещо развитие и усъвършенстване:	12
Използвана литература:	12
Връзка към хранилище в <i>Github</i> :	12

Таблицы и графики:

Фиг.1: Снимка на екрана при стартиране на програмата	4
Фиг.2: Снимка на екрана при въвеждане на различни команди	5
Фиг.3: Реализираните класове заедно с техните член-данни и методи	5
Фиг.4: Команда <i>add</i> в класа <i>Store</i>	6
Фиг.5: Команда <i>remove</i> в клас <i>Store</i>	7
Фиг.6: Команда <i>log<from><to></i> в клас <i>Store</i>	7
Фиг.7: <i>Header</i> файловете и <i>Source</i> файловете в проекта	8
Фиг.8: Методи и член-данни на класа <i>AuditStatement</i>	9
Фиг.9: Методи и член-данни на класа <i>ISODate</i>	9
Фиг.10: Методи и член-данни на класа <i>Location</i>	10
Фиг.11: Методи и член-данни на класа <i>Product</i>	10
Фиг.12: Методи и член-данни на класа <i>Space</i>	11
Фиг.13: Методи и член-данни на класа <i>Store</i>	11

1. Увод:

1.1. Описание и идея на проекта:

В рамките на зададения проект трябва да се реализира програма, реализираща информационна система, обслужваща склад. Програмата съхранява и обработва данните за наличността в склада във файл.

1.2. Цел и задачи на разработката:

Основната цел е програмата да бъде достатъчно лесна и удобна за използване. За целта в проекта е реализиран *CommandsExecutor*, който съдържа метод *showStartMenu*. В *main()* функцията на програмата ще можем да стартираме програмата само с извикването на този метод. Програмата ще изисква от потребителя да въведе някоя от следните функции: *open*, *help* или *exit*, с помощта на която да определи коя функция да бъде изпълнена. При въвеждането на невалидни данни (данни, различни от *open*, *help*, *exit*) на стандартния изход ще бъдат изведени отново възможните функции. Всички промени, които се правят, трябва да бъдат записвани във файл. Когато бъде записан нов файл, изпълнението на програмата продължава и се извежда съобщение за успешното записване на данните във файл. Програмата завършва, когато бъде въведена функцията *exit*.

1.3. Структура на документацията:

Структурата на документацията се състои в няколко опорни точки. Първоначално се представят належащите проблеми, както и търсенето на достатъчно ефективни решения. След това се преминава към проектиране на проекта и накрая се преминава към реализация и тестване на проекта. Накрая се представят насоки за бъдещо развитие и усъвършенстване.

2. Преглед на предметната област:

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

- Използвана е определена организация на пространството в склада. Той е разделен на 2 части като първата половина е за продукти, чиято мерна единица е килограм, а другата половина е за продукти, чиято мерна единица е литър. Създадени са константите

```
const int SECTIONS_COUNT = 4;  
const int SHELVES_COUNT = 6;  
const int NUMBERS_COUNT = 15;  
const int PRODUCTS_NUMBER = 30;
```

като всяко местоположение на продукт се характеризира със секция, рафт и номер. В склада има 4 секции като на всяка секция и 6 рафта, на които има по 15 номера като на всеки номер могат да се поберат по 30 продукта.

- Проектът работи с дати, които са конструирани в *ISO 8601* формат, който представлява международен стандарт за обмен на дати и данни за време. Стойностите за дата и време се подреждат от най-значителната към най-незначителната: година, месец, ден, час, минута, секунда и част от секундата. В проекта датите са представени само с година, месец и ден. В класът *ISODate* е представен метод *constructDate*, който конструира дата в *ISO 8601* формат от *string*. За повече информация https://bg.wikipedia.org/wiki/ISO_8601.

- Някои от функциите, дефинирани в *CommandsExecutor* са разделени на две части в зависимост от това кога се извикват. Такива функции са *help()* и *showMenu()*. Функцията *help* е разделена на *showHelp()* и *showAdvancedHelp()* като *showHelp()* се извиква, ако при стартиране на програмата, потребителят иска да види кратка информация за командите, а *showAdvancedHelp()* се извиква, когато вече е зареден един файл и потребителят иска да провери информация за командите. Функцията *showMenu()* е разделена на *showStartMenu()* и *showAdvancedMenu()*, като *showMenu()* се извиква за първоначалното меню, а *showAdvancedMenu()* се извиква, когато вече е зареден файл.

2.2. Дефиниране на проблеми и сложност на поставената задача:

Има множество проблеми, които трябва да бъдат преодоляни с цел успешната реализация на проекта. Те включват гарантирането на удобен и лесен за използване интерфейс. Друг проблем е, че въведените от потребителя данни, трябва да бъдат валидирани. Също така, функционалността на проекта се явява проблем. Въведените от потребителя данни, освен че трябва да бъдат валидирани, то трябва да бъдат записвани и във файл, който да бъде достъпен и след приключване на програмата. Също така проектът трябва да работи с дати, които трябва да бъдат валидирани в специален формат.

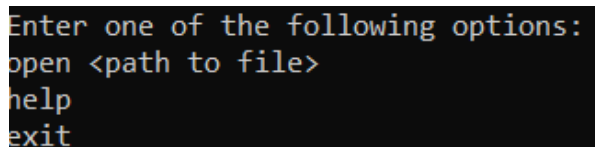
2.3. Подходи , методи (евентуално модели и стандарти) за решаване на поставените проблеми:

Начините за разрешаване на поставените проблеми включват използване на различни методи. Всеки метод ще се грижи за коректността на изпълнението, както и изхода на програмата.

2.4. Потребителски (функционални) изисквания (права, роли, статуси, диаграми) и качествени (нефункционални) изисквания (скалируемост, поддръжка):

Функционалността започва от *CommandsExecutor*, където се въвежда команда от потребителя, тя се обработва, разпознава коя е командата и я изпълнява.

При отваряне на проекта, потребителят има 3 възможности – да отвори файл, да види кратка информация за възможните потребителски команди и да излезе от програмата.



```
Enter one of the following options:
open <path to file>
help
exit
```

Фиг.1: Снимка на екрана при стартиране на програмата

При отваряне на вече съществуващ файл или започване на нов, потребителят има възможност да:

- затвори файла(*close*);
- да го запази със същото име(*save*);
- да го запази с ново име(*save as*);
- да види кратка информация за възможните потребителски команди(*help*);

- да излезне от програмата(*exit*);
- да види информация за наличните продукти в склада(*print*);
- да добави нов продукт(*add*);
- да премахне продукт(*remove*);
- да получи справка за всички промени в наличността в даден период(*log<from> <to>*);
- да разчисти склада от всички стоки, на които е изтекъл срокът на годност(*clean*).

```

Enter one of the following options:
close
save
save as
help
exit
print
add
remove
log <from> <to>
clean

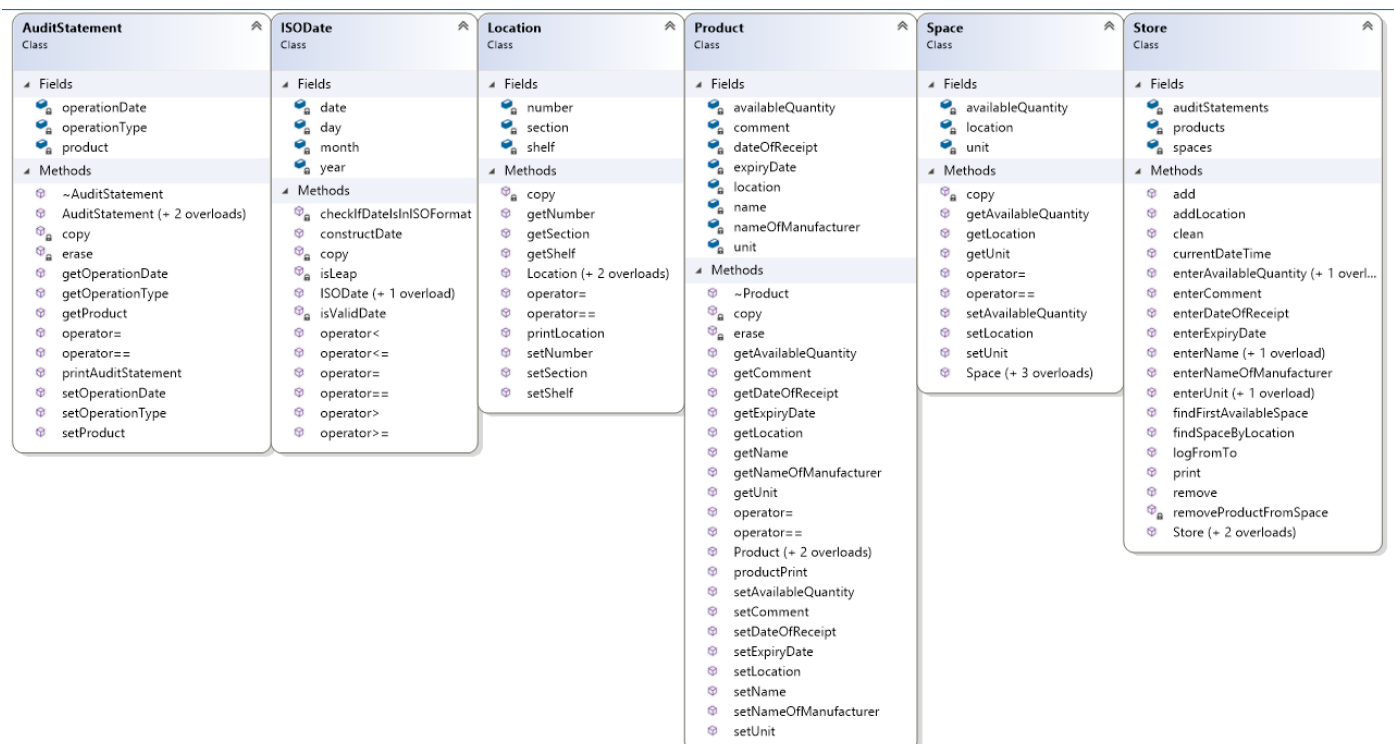
```

Фиг.2: Снимка на екрана при въвеждане на различни команди

3. Проектиране:

3.1. Обща архитектура

Общата архитектура на проекта е представена на фиг. 3.



Фиг.3: Реализираните класове заедно с техните член-данни и методи

Обектно – ориентираният дизайн се състои в реализацията на файла *CommandsExecutor*, където се намират основните операции за реализацията на този проект. Това са операциите *print*, *add*, *remove*, *log <from> <to>* и *clean*, в допълнение на общите операции *open*, *close*, *save*, *save as*, *help*, *exit*.

3.2. Диаграми (на структура и поведение – по обекти, слоеве с най – важните извадки от кода):

Най-важните части на програмата са свързани с основните команди, които са представени на фигури 4, 5 и 6.

```
114 //Функция add, която добавя нов продукт в склада.
115 void Store::add()
116 {
117     Product product;
118     enterName(product);
119     enterExpiryDate(product);
120     enterDateOfReceipt(product);
121     enterNameOfManufacturer(product);
122     enterUnit(product);
123     enterAvailableQuantity(product);
124     enterComment(product);
125     bool isSuitableLocationFound = addLocation(product);
126
127     //Ако е намерено място за продукта, то го добавяме във вектора от продукти.
128     //Добавяме му датата, на която е извършена операцията.
129     //За дата на извършване на операцията приемаме датата, на която сме добавили продукта в склада.
130     //Датата на извършване на операцията използваме във функцията log<from><to>.
131     if (isSuitableLocationFound)
132     {
133         this->products.push_back(product);
134         string currentDateString = currentDateTime();
135         ISODate currentDate;
136         currentDate.constructDate(currentDateString);
137         AuditStatement auditStatement = AuditStatement("add", product, currentDate);
138         this->auditStatements.push_back(auditStatement);
139     }
140
141     //Ако не е намерено място за продукта в склада, то извеждаме съобщение за грешка.
142     else
143     {
144         cout << "No suitable location was found for the product: " << endl << product << endl;
145     }
146 }
```

Фиг.4: Команда *add* в класа *Store*

```

148 //Функция remove, която извежда продукт от склада.
149 void Store::remove()
150 {
151     string name = enterName();
152     string unit = enterUnit();
153     double availableQuantity = enterAvailableQuantity();
154
155     Product* oldestProductMatchingCriteria = nullptr;
156     int indexOfOldestProduct;
157
158     //Обхождаме вектора от продукти.
159     for (int i = 0; i < products.size(); i++)
160     {
161         //Ако името на продукта, мерната единица и количеството съвпадат с въведените от потребителя, то премахваме този продукт.
162         if (products[i].getName().compare(name) == 0 && products[i].getUnit().compare(unit) == 0 &&
163             products[i].getAvailableQuantity() == availableQuantity)
164         {
165             if (oldestProductMatchingCriteria == nullptr || products[i].getExpiryDate() < oldestProductMatchingCriteria->getExpiryDate())
166             {
167                 oldestProductMatchingCriteria = &products[i];
168                 indexOfOldestProduct = i;
169             }
170         }
171     }
172
173     //Ако не е намерен такъв продукт, то извеждаме съобщение за грешка.
174     if (oldestProductMatchingCriteria == nullptr)
175     {
176         cout << "No product matching criteria is found." << endl;
177     }
178
179     //Ако открием такъв продукт, то го добавяме и във вектора от auditStatement-и.
180     else
181     {
182         Product productCopy = *oldestProductMatchingCriteria;
183         removeProductFromSpace(*oldestProductMatchingCriteria);
184
185         string currentDateString = currentDate();
186         ISODate currentDate;
187         currentDate.constructDate(currentDateString);
188         AuditStatement auditStatement = AuditStatement("remove", *oldestProductMatchingCriteria, currentDate);
189         this->auditStatements.push_back(auditStatement);
190         products.erase(products.begin() + indexOfOldestProduct);
191         cout << "The product was removed." << endl;
192         cout << productCopy << endl;
193     }
194 }

```

Фиг.5: Команда *remove* в клас *Store*

```

196 //Функция log <from><to>, която извежда справка за всички промени в наличността в даден период.
197 void Store::logFromTo(ISODate fromDate, ISODate toDate, Store& store)
198 {
199     int counter = 0;
200     //Проверяваме за коректност на въведените дати.
201     if (fromDate <= toDate)
202     {
203         //Обхождаме вектора от auditStatement-и.
204         for (int i = 0; i < auditStatements.size(); i++)
205         {
206             //Проверяваме дали датата на извършване на операцията е в интервала между началната дата и крайната дата.
207             //Ако е в този интервал, то принтираме всички добавяния и премахвания на продукти.
208             //Чрез counter броим колко добавяния и премахвания на продукти има.
209             if (auditStatements[i].getOperationDate() >= fromDate && auditStatements[i].getOperationDate() <= toDate)
210             {
211                 store.auditStatements[i].printAuditStatement();
212                 counter++;
213             }
214         }
215
216         //Ако нямаме нито едно добавяне или премахване на продукт, то извеждаме съобщение за грешка.
217         if (counter == 0)
218         {
219             cout << "There is no operation." << endl;
220         }
221     }
222
223     //Ако <from> е по-голям от <to>, то въведените дати са некоректни и извеждаме съобщение за грешка.
224     else if (fromDate > toDate)
225     {
226         cout << "Error, bad input!" << endl;
227     }
228 }

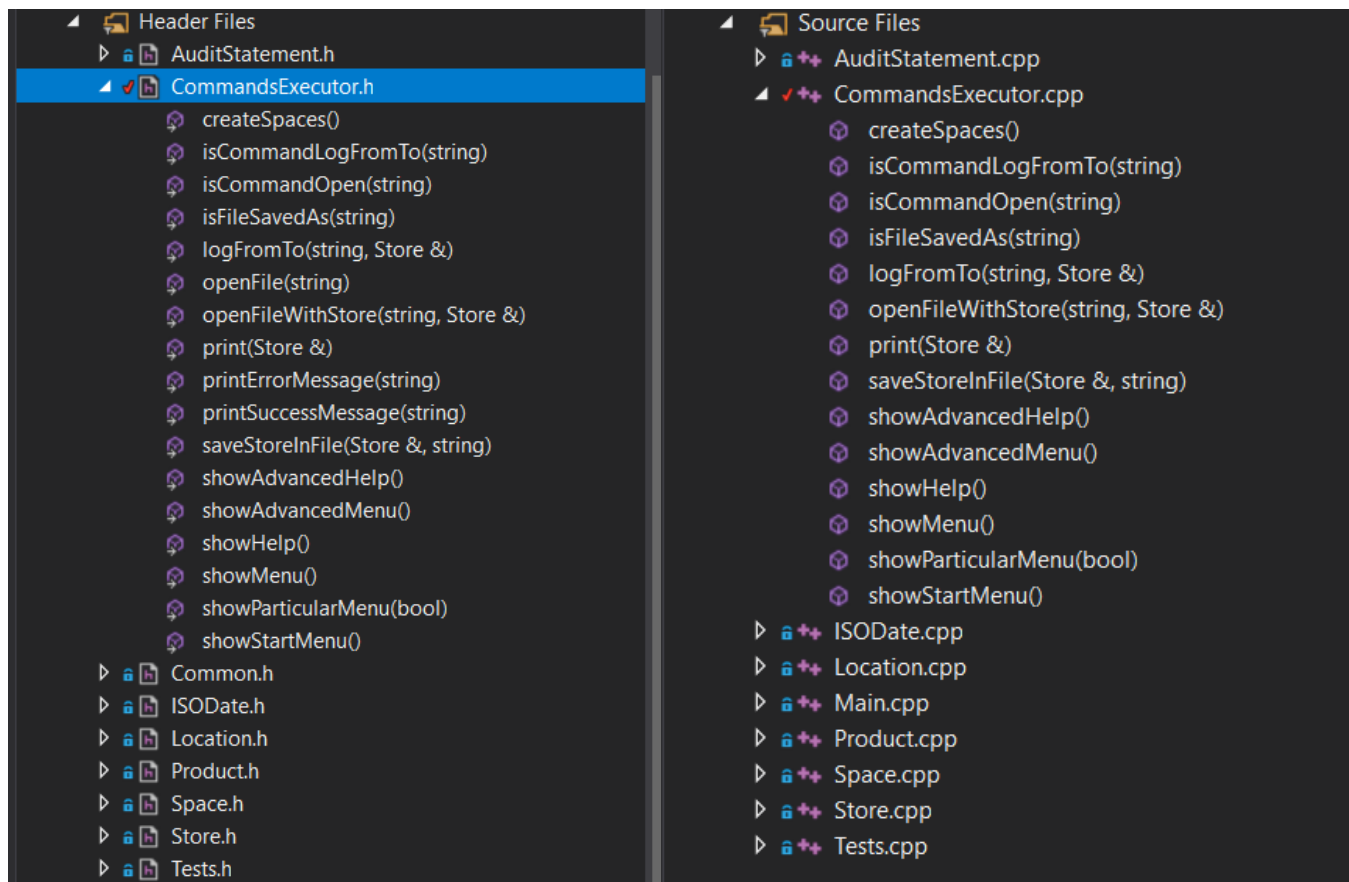
```

Фиг.6: Команда *log<from><to>* в клас *Store*

4. Реализация, тестване:

4.1. Реализация на класове:

Използвано е разделно компилиране за по – добра организация и четимост – Фиг. 7.



Фиг.7: *Header* файловете и *Source* файловете в проекта

В проекта са реализирани множество от класове, които са представени чрез голямата четворка или в т.н. канонична форма. Също така са реализирани помощни класове и функции за оптимизация и по-добра четимост. Използвани са възможно най-малко глобални променливи и е наблегнато на локалните променливи. Осъществена е връзка между отделните класове.

Функцията *open<path>* създава нов файл, ако не съществува файл с име *<path>* или отваря вече съществуващ такъв. При отваряне на програмата и въвеждане на функцията *help* се извиква функцията *showHelp()*, която дава информация за функциите *open*, *help* и *exit*. При въвеждане на функцията *exit* се излиза от програмата. При отваряне на файл потребителят има няколко възможности – *close*, *save*, *save as*, *help*, *exit*, *print*, *add*, *remove*, *log <from> <to>* и *clean*.

Класът *AuditStatement* е създаден с цел в него да се записват данните при добавяне и премахване на продукт. Този клас се използва главно за функцията *log<from><to>*, тъй като за нея трябва да бъдат записани данните на всички продукти, които са добавени или премахнати. Този клас съдържа методите и член-данните от фигура 8.

Name	Type	Modifier	Summary
Methods			
~AuditStatement		public	
AuditStatement		public	
AuditStatement		public	
AuditStatement		public	
copy	void	private	
erase	void	private	
getOperationDate	ISODate	public	Getters
getOperationType	string	public	
getProduct	Product	public	
operator=	AuditStatement&	public	
operator==	bool	public	
printAuditStatement	void	public	
setOperationDate	void	public	Setters
setOperationType	void	public	
setProduct	void	public	
Fields			
operationDate	ISODate	private	
operationType	string	private	
product	Product	private	

Фиг.8: Методи и член-данни на класа *AuditStatement*

Класът *ISODate* е създаден с цел конструиране на дати в *ISO 8601* формат. В него е реализиран метод *constructDate*, който е представен като булева функция, а не като конструктор, тъй като проверява дали въведената дата е коректна. Този клас съдържа методите и член-данните от фигура 9.

Name	Type	Modifier	Summary
Methods			
checkIfDatelsInISOFormat	bool	private	
constructDate	bool	public	
copy	void	private	
isLeap	bool	private	
ISODate		public	
ISODate		public	
isValidDate	bool	private	Проверяваме дали датата е валидна
operator<	bool	public	
operator<=	bool	public	
operator=	ISODate&	public	
operator==	bool	public	
operator>	bool	public	
operator>=	bool	public	
Fields			
date	string	private	
day	int	private	
month	int	private	
year	int	private	

Фиг.9: Методи и член-данни на класа *ISODate*

Класът *Location* е създаден с цел конструиране на местоположение на продукта. Този клас съдържа член-данните и методите от фигура 10.

Name	Type	Modifier	Summary
Methods			
copy	void	private	
getNumber	int	public	Getters
getSection	int	public	
getShelf	int	public	
Location		public	
Location		public	
Location		public	
operator=	Location&	public	
operator==	bool	public	
printLocation	void	public	Принтираме мястото на продукта.
setNumber	void	public	Setters
setSection	void	public	
setShelf	void	public	
Fields			
number	int	private	
section	int	private	
shelf	int	private	

Фиг.10: Методи и член-данни на класа *Location*

Класът *Product* е създаден с цел да съдържа всички характеристики на продукта. Неговите член-данни и методи са описани във фигура 11.

Name	Type	Modifier	Summary
Methods			
~Product		public	
copy	void	private	
erase	void	private	
getAvailableQuantity	double	public	
getComment	string	public	
getDateOfReceipt	ISODate	public	
getExpiryDate	ISODate	public	
getLocation	Location	public	
getName	string	public	Getters
getNameOfManufacturer	string	public	
getUnit	string	public	
operator=	Product&	public	
operator==	bool	public	
Product		public	
Product		public	
Product		public	
productPrint	void	public	
setAvailableQuantity	void	public	
setComment	void	public	
setDateOfReceipt	void	public	
setExpiryDate	void	public	
setLocation	void	public	
setName	void	public	Setters
setNameOfManufacturer	void	public	
setUnit	void	public	
Fields			
availableQuantity	double	private	
comment	string	private	
dateOfReceipt	ISODate	private	
expiryDate	ISODate	private	

Фиг.11: Методи и член-данни на класа *Product*

Класът *Space* е създаден с цел в него да се пази информация за мястото на продукта. Той използва местоположението, мерната единица и количеството на продукта, за да продуктите да бъдат подредени в склада. На фигура 12 са показани методите и член-данните на този клас.

Name	Type	Modifier	Summary
Methods			
copy	void	private	
getAvailableQuantity	int	public	
getLocation	Location*	public	Getters
getUnit	string	public	
operator=	Space&	public	
operator==	bool	public	
setAvailableQuantity	void	public	
setLocation	void	public	Setters
setUnit	void	public	
Space		public	
Space		public	
Space		public	
Space		public	
Fields			
availableQuantity	int	private	
location	Location*	private	
unit	string	private	

Фиг.12: Методи и член-данни на класа *Space*

Класът *Store* е основен за този проект. В него са реализирани функциите *print*, *add*, *remove*, *log<from><to>*, *clean*. Той използва голяма част от методите и член-данните на останалите класове. На фигура 13 са изобразени методите и член-данните на класа *Store*.

Name	Type	Modifier	Summary	Hide
Methods				
add	void	public	Функция add, която добавя нов продукт в склада.	
addLocation	bool	public	Добавяме място на продукта.	
clean	void	public	Функция clean, която разчиства склада от всички стоки, на които е изтекъл срокът на годност или от тези	
currentDateTime	const string	public	Използвано от https://stackoverflow.com/questions/997946/how-to-get-current-time-and-date-in-c	
enterAvailableQuantity	double	public	Функция, чрез която въвеждаме количеството на продукта.	
enterAvailableQuantity	void	public		
enterComment	void	public	Функция, чрез която въвеждаме коментар за продукта.	
enterDateOfReceipt	void	public	Функция, чрез която въвеждаме датата на постъпване в склада.	
enterExpiryDate	void	public	Функция, чрез която въвеждаме срокът на годност на продукта.	
enterName	string	public	Функция, чрез която въвеждаме име на продукта.	
enterName	void	public		
enterNameOfManufacturer	void	public	Функция, чрез която въвеждаме името на доставчика на продукта.	
enterUnit	string	public	Функция, чрез която въвеждаме мерната единица на продукта.	
enterUnit	void	public		
findFirstAvailableSpace	Space*	public	Намираме първото свободно място.	
findSpaceByLocation	Space	public		
logFromTo	void	public	Функция log <from> <to>, която извежда справка за всички промени в наличността в даден период.	
print	void	public		
remove	void	public	Функция remove, която извежда продукт от склада.	
removeProductFromSpace	void	private		
Store		public	Създаваме вектор от space-ове.	
Store		public	Създаваме вектор от auditStatement-и.	
Store		public		
Fields				
auditStatements	vector<AuditStatement>	private		
products	vector<Product>	private		
spaces	vector<Space>	private		

Фиг.13: Методи и член-данни на класа *Store*

4.2. Планиране, описание и създаване на тестови сценарии:

Първите стъпки от тестването на проекта включват тестване на конструкторите на всички класове. След това бяха тествани общите команди – *open*, *save*, *save as*, *close*, *help*, *exit*. Следващите стъпки от тестването са свързани със създаване на склад, който съдържа множество продукти, върху които се извършват различни операции. Първоначално, чрез функцията *add*, която добавя нов продукт, бяха добавени няколко продукта с различно име, срок на годност и мерна единица, както и няколко продукта с еднакви характеристики. Бяха въведени продукти, както с изтекъл, така и с неизтекъл срок на годност. След това беше извикана функцията *remove*, която изважда продукт от склада. Беше въведен продукт, който съществува в склада и такъв, който не съществува, за да бъде проверено поведението на програмата в различни ситуации. След това беше извикана функцията *log*<*from*><*to*>, която извежда справка за всички промени в наличността за определен период от време. Бяха въведени валидни и невалидни периоди – например когато датата <*to*> е по-малко от датата <*from*>. Например *log 2019-02-02 2018-05-05*. След това бяха въведени валидни периоди като бяха въвеждани периоди, в които няма нито една промяна в наличността, както и такива с няколко промени. След това беше извикана функцията *clean*, която разчиства склада от всички стоки с изтекъл срок на годност. В склада имаше продукти с изтекъл срок, които бяха премахнати след извикването на тази функция. Бяха създадени няколко броя склада, като на всеки бяха тествани всички функции.

5. Заключение:

5.1. Обобщение на изпълнението на началните цели:

Проектът е завършен успешно. Интерфейсът е достатъчно функционален и същевременно лесен за използване и удобен.

5.2. Насоки за бъдещо развитие и усъвършенстване:

Част от бъдещите планове за развитие на този проект включват подобряване на функционалността на проекта, което ще бъде свързано с добавяне на нови характеристики на продуктите – например цена. Също така валидиране на входните данни - за всички продукти с мерна единица литър да не може да бъде въведена мерна единица килограм. Друга идея за бъдещо развитие на този проект включва изчисляване на загубите от продуктите, чийто срок на годност е изтекъл. Също така може да бъдат въведени и мерни единици грам и милилитър.

Използвана литература:

1. https://en.wikipedia.org/wiki/ISO_8601

Връзка към хранилище в *Github*:

<https://github.com/VeselaStoyanova>