Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра Програмної інженерії

КУРСОВА РОБОТА

ПОЯСНЮВАЛЬНА ЗАПИСКА

з дисципліни “ Об’єктно - орієнтоване програмування”

“Склад”

Керівник , проф. Бондарєв В.М.

Студент гр. ПІ - 15 – 1 Торбієвський О.М

Комісія:

проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дудар З.В.

<посада> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_<прізвище, ініціали >

<посада> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_<прізвище, ініціали >

Харків 2016

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ**

Кафедра: ***Програмної інженерії***

Дисципліна: ***Об’єктно-орієнтоване програмування***

Спеціальність: ***Програмна інженерія***

Курс 1 .Група *ПІ-15-1* . Семестр *2* .

***ЗАВДАННЯ***

***на курсовий проект студента***

*Торбієвського Олександра Михайловича*

1. Тема проекту: «Склад»
2. Термін здачі студентом закінченого проекту: ***“25” - травня - 2016 р***.
3. Вихідні дані до проекту: база товарів, які зберігаються на складі: назва, кількість, одиниці виміру. Реєстрація завозу, видачі товарів. Формування накладної, яка містить інформацію про всі товари, які зареєстровані на складі.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |
| --- | --- | --- |
| *№* | *Назва етапу* | *Термін виконання* |
| 1 | Видача теми, узгодження і затвердження теми | 1-03-2016 р. |
| 2 | Формулювання вимог до програми | 1-03-2016 – 6-03-2016 р. |
| 3 | Розробка підсистеми зберігання та пошуку товарів. | 7-03-2016 – 20-03-2016 р. |
| 4 | Розробка підсистеми реєстрації прибуття та відгрузки товару. | 21-03-2016 – 3-04-2016 р. |
| 5 | Розробка функцій зберігання та завантаження даних | 4-04-2016 – 9-04-2016 р. |
| 6 | Тестування і доопрацювання розробленої програмної системи. | 10-04-2016 – 29-04-2016 р. |
| 7 | Оформлення пояснювальної записки, додатків, графічного матеріалу | 1-05-2016 – 24-05-2016 р. |
| 8 | Захист | 25-05-2016 р. |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Торбієвський О.М.

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бондарєв В. М.

« 1 »\_березня\_\_\_\_\_\_\_\_\_\_ 2016 р.

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 56 с., 32 рис., 1 додаток, 5 джерел.

Метою роботи є розробка програми “Склад” на засадах об'єктно-орієнтованого програмування.

Методи розробки базуються на використанні середи розробки Microsoft Visual Studio 2015 Enterprise, Windows Forms, платформи .NET Framework 4.5, мови програмування C#.

Профіль на github: https://github.com/TheFlaved/Warehouse\_OOP

В результаті отримана програма під назвою “Склад”, яка дозволяє наглядати за складом, бачити які товари є у наявності, їх кількіть, одиниці виміру. Товар має такі характеристики: назва, одиниці виміру, ціна за одиницю та кількість товару в упаковці, остання зміна у кількості та її дата. Є можливість реєструвати прибуття та видачу товарів. Також програма має функцію формування відомості у HTML файл про товари, які є на складі.

ОБ’ЕКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ПРОГРАМА, КЛАС, C#, ПЛАТФОРМА .NET, ТОВАР, СКЛАД, HTML.

ЗМІСТ

Вступ 6

1 Специфікація програми 7

1.1 Функції програми 8

1.2 Інтерфейс користувача 8

2 Проектна специфікація. 14

2.1 Архітектурний стиль 14

2.2. Об’єктна структура програми 15

2.3 Формат даних 16

3 Інструкція користувача 17

ВИСНОВКИ 22

ПЕРЕЛІК ПОСИЛАНЬ.. 23

Додаток А – Код програми.. 24

ВСТУП

Керувати складським приміщенням – дуже складна робота, яка вимагає від наглядача максимальної концентрації та неймовірно хорошої пам’яті.

Дослідивши негативні сторони ведення паперової документації, ми можемо заявити, що це важкий та трудомісткий процес, що віднімає багато сил.

Таким чином можно сказати, що переведення такого виду документації у електронний вигляд дуже доцільне та може сильно полегшити життя наглядачів.

Програма цілком заміняє усю паперову тяганину, яка замінюється автоматичною роботою програми. Таким чином модлювання неповноцінно функціонуючої складської системи стає набагато складнішим, бо автоматизація складських приміщень дає небачені переваги.

1 СПЕЦИФІКАЦІЯ ПРОГРАМИ

1.1 Функції програми

Программа повинна повністю імітувати усю паперову роботу складського наглядача

Програма, що буде розроблена, повинна мати наступні функції:

1. відображення бази товарів;
2. додавання нового товару;
3. реєстрацію змін, прибуття або видачі товару;
4. видалення товару;
5. пошук товару за назвою;
6. сортування товару за кількістю, загальною сумою, алфавітним порядком;
7. друк відомості про складське приміщення.

1.2 Інтерфейс користувача

Після запуску програми користувач побачить головну форму, вона ж форма управління базою товарів (рисунок 1.1).

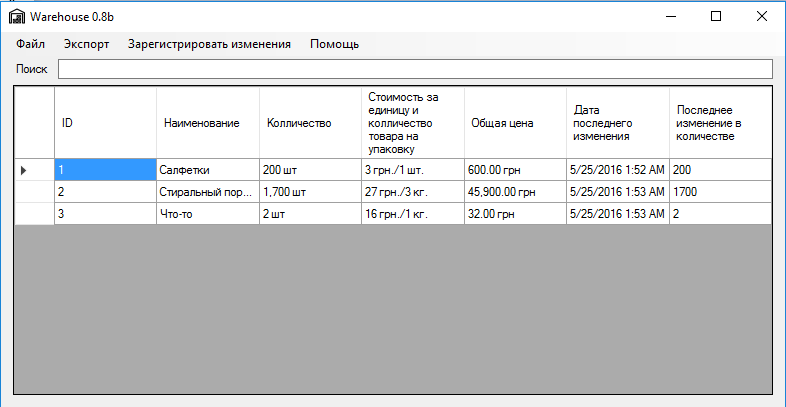


Рисунок 1.1 – Головна форма

На формі зображена база товарів. Таким чином користувач може переглянути всі характеристики бажаного товару, а саме: назву, кількість, ціну за одиницу та кількість товару у упаковці.

При натисканні пункту меню «Зарегистрировать изменения». Ми бачимо наступну форму. (Рисунок 1.2)

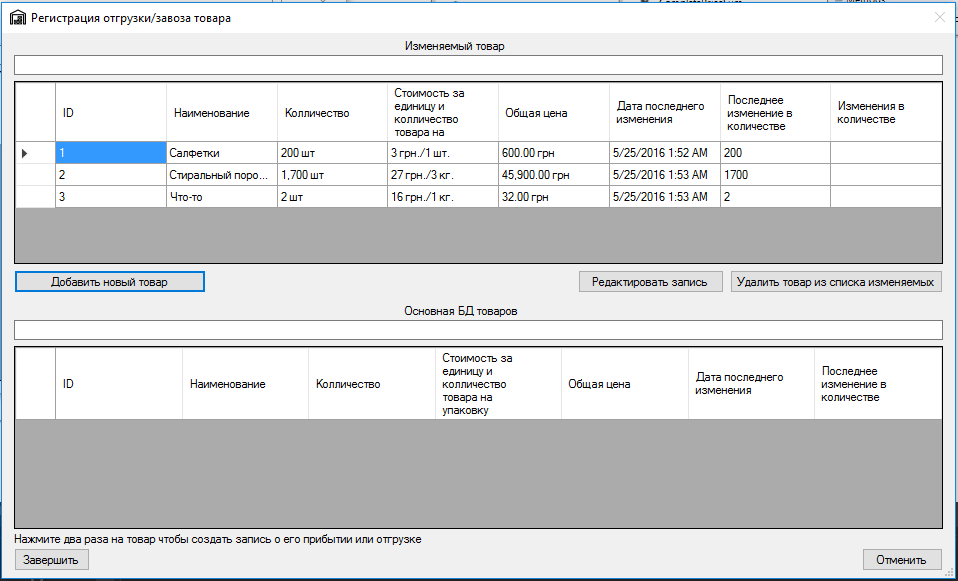


Рисунок 1.2 – Форма реєстрації прибуття або відгрузки товару

У формі ми бачимо дві таблиці, одна з яких відображає товари які є у данний момент на складі (знизу), а зверху ми бачимо товари додані або змінені у момент реєстрації. Колонка «Изменения в количестве», може реагувати двома способами. Коли товар вже був на складі, ви можете змінити його кількість у позитивну або негативну сторону, кількість товару не може бути нижче 0 (Рисунок 1.3), тому якщо не виконати цю умову, ви побачите вікно з помилкою. Щоб додати існуючий товар достатньо два рази натиснути на товар із таблиці з основною базою даних.

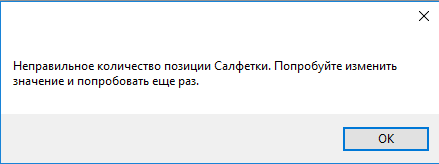


Рисунок 1.3 – Форма «Помилка»

При виділенні товару у таблиці «Изменяемый товар» та натисканні кнопки «Редактировать запись».(Рисунок 1.4) Ми побачимо вікно у якому ми можемо змінити назву та іншні характеристики товару, але не можемо змінити кількість, бо це порушує цілісність системи зміни кількості товару, яка змінюється беспосередьно при реєстрації змін, форма додавання товару аналогічна, викликається кнопкою «Добавить новий товар».

.

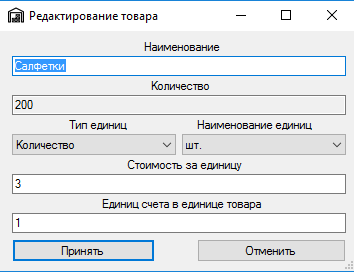


Рисунок 1.4 – Форма редагування товару при реєстрації

Поля форми заповнені даними з бази. На формі знаходяться дві кнопки: «Принять» і «Отменить». «Отменить» закриває форму, виділений товар залишається незмінним і користувач повертається до головної форми. «Принять» також закривє форму, але зберігає зміни у товарі, не змінюючи унікальний ідентифікатор. Перевірка на заповненість полів також присутня. На головній формі у базі користувач побачить зроблені ним зміни у даному препараті.

При натисканні «Удалить товар из списка изменяемых» відобразиться дочірня форма (рисунок 1.5) для підтвердження видалення виділеного у базі товару. При цьому товар у основный базі залишається незмінним.

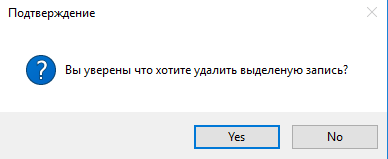


Рисунок 1.5 – Форма підтвердження видалення товару з редагуємих

Функція пошуку дозволяє знайти товар за його назвою, або її частиною. Такий пошук реалізований у всіх таблицях даної програми. (Рисунок 1.6)

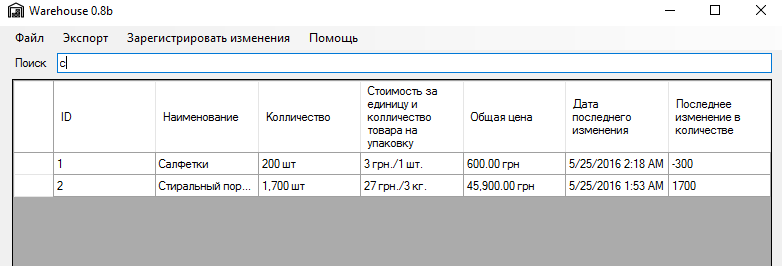


Рисунок 1.6 – Реалізація пошуку

Також знаходячись у головній форму можна зовсім видалити товар із бази складу.(Рисунок 1.7)

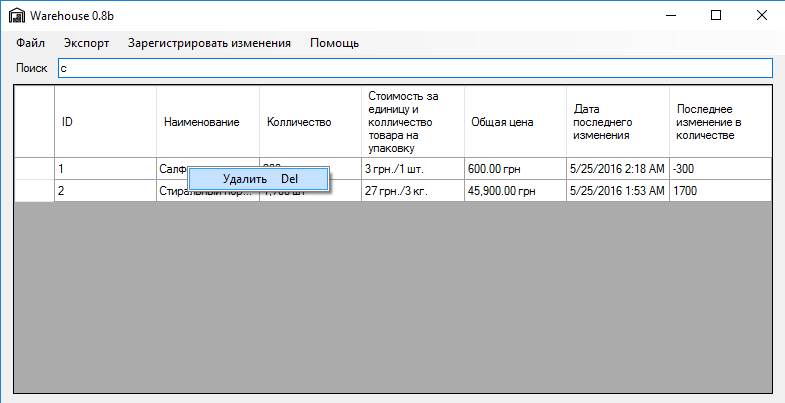


Рисунок 1.7 – Контекстне меню видалення

При натисканні цього пункту меню користувачу пропонується підтвердити дію.(Рисунок 1.8)

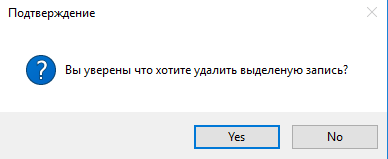


Рисунок 1.8 – Форма підтвердження

Також у програмі присутня функція виведення відомості про склад у файл. Викликається вона натисканням кнопки «Экспорт» та її підменю «В файл». (Рисунок 1.10) Потім користувач вибирає місце для збереження файла.

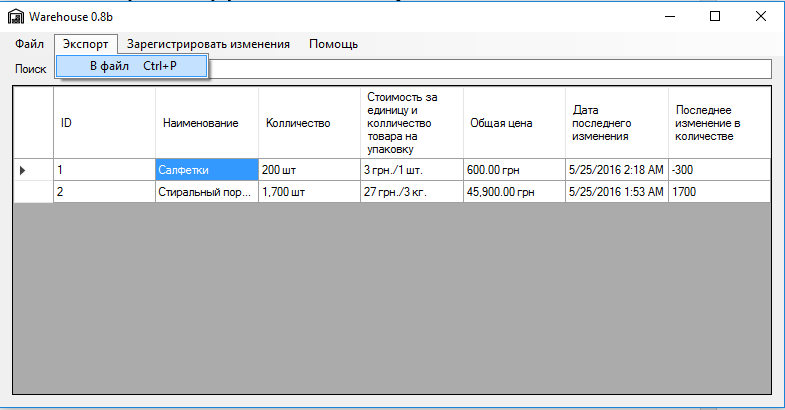


Рисунок 1.10 – Метод вивдення відомості у HTML файл.

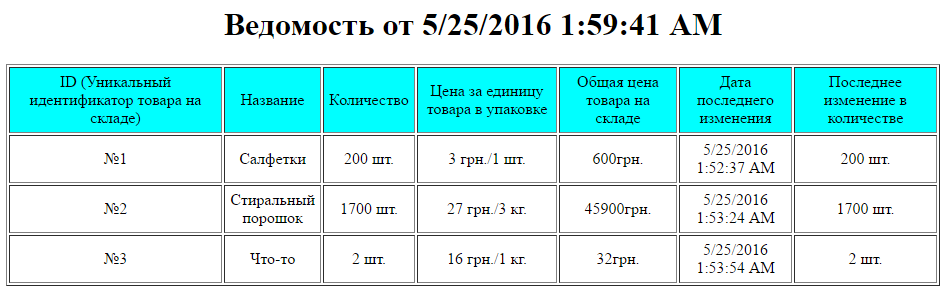


Рисунок 1.11 – Вигляд відомості

2 ПРОЕКТНА СПЕЦИФІКАЦІЯ

2.1 Архітектурний стиль

Додаток Windows Forms є подієво-орієнтований додатком, який підтримується додотаком Microsoft .NET Framework. На відміну від пакетних програм, велика частина часу витрачається на очікування від користувача будь-яких дій, як, наприклад, введення тексту в текстове поле або кліка мишкою по кнопці.

Була частково реалізована ідея MVC, бо її реалізація за допомогою MVC є дуже важким і непотрібним процесом, тому у процесі розробки було вирішено об’єднати контролер і зображення – Windows.Forms. Це ніяк не порушує ідею MVC. Головна задача – зробити модель повністю незалежною від інших частин програми, що дозволить робити зміни у контролері або зображенні, не впливаючи на модель.

Зараз проект працює з файлами, але у майбутньому може виникнути необхідність підключити базу данних. Дуже часто також виникає необхідність змінити зображення, не змінюючи інші частини програми. Саме через ці причини була вибрана архітектура MVC.

2.2. Обєктна структура програми

Програма представлена набором класів, що взаємодіють між собою (рисунок 2.1).

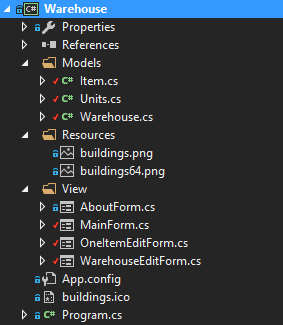


Рисунок 2.1. – Архітектура проекту

У програмі реалізовані такі класи: Item, Units та його наслідники, Warehouse. А також класи форм, які містять обробники дій Діаграма класів представлена на рисунку 2.2.

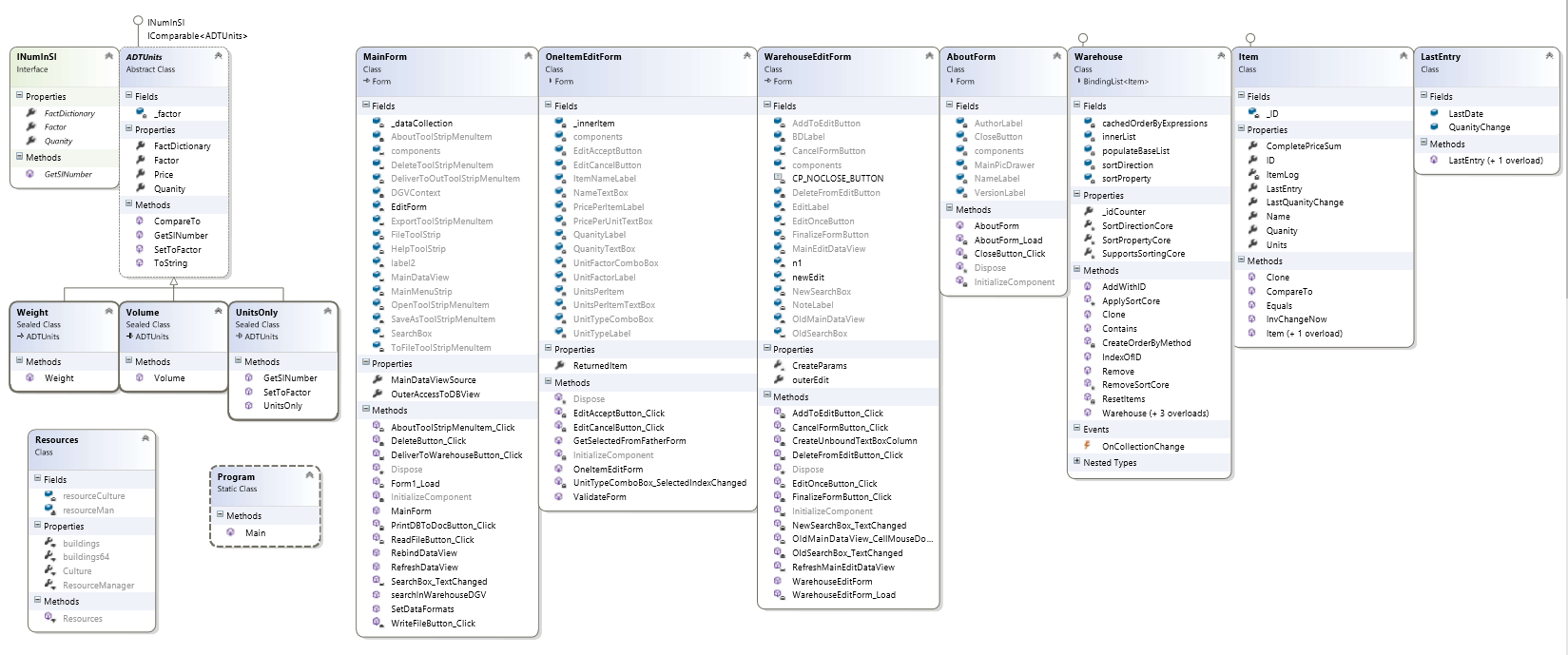


Рисунок 2.2 – Діаграма класів

2.3. Формат даних

Для зберігання інформації використовується механізм серіалізації. Серіалізація – це процес перетворення об’єкта у потік байтів. Після перетворення цей потік байтів можна записати на диск або тимчасово зберегти у пам’яті. За необхідністю можливий зворотній процес – десереалізація, тобто отримання із потоку байтів попередньо збереженого об’єкту.

У програмі всі дані зберігаються у .bin файлі, це двоїчна серіалізація, де дані зберігаються у двоїчний код.

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

Встановлення програми:

* 1. Запускаємо «Мій комп'ютер».
  2. Заходимо на носій з програмним продуктом.
  3. Розпаковуємо архів у бажану папку на комп'ютері.
  4. Можна запускати «.exe» файл і починати роботу з програмою.

Робота з програмою

Запускаємо програму. Головне вікно зображене на рисунку 3.1. За змовчуванням відкривається головна форма.

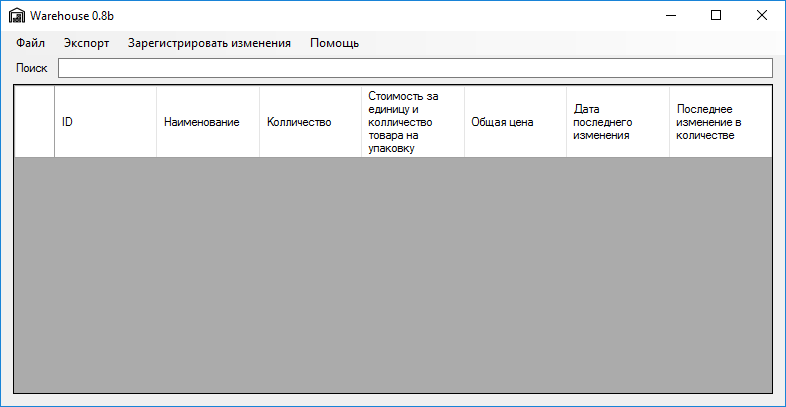


Рисунок 3.1 – Головна форма

Видалення товару:

А) настиснути правою кнопкою миші на товар

Б) натиснути на кнопку видалити

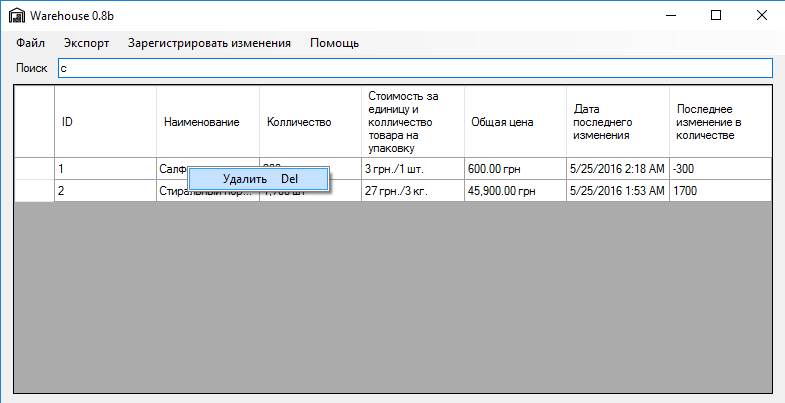


Рисунок 3.2 – Видалення товару

Реєстрація відгрузки та прибутитя товару:

А) натиснути «Зарегистрировать изменения»

Б) натиснути «Добавить товар»

В) додати товар

Г) повторювати Б-В стільки разів скільки потрібно

Д) у разі необхідності додати подвійним кліком товари, які вже існували у базі даних

Е) Змінити кількість товарів існуючих у базі даних змінюючи дані у колонці «Изменения в количестве»

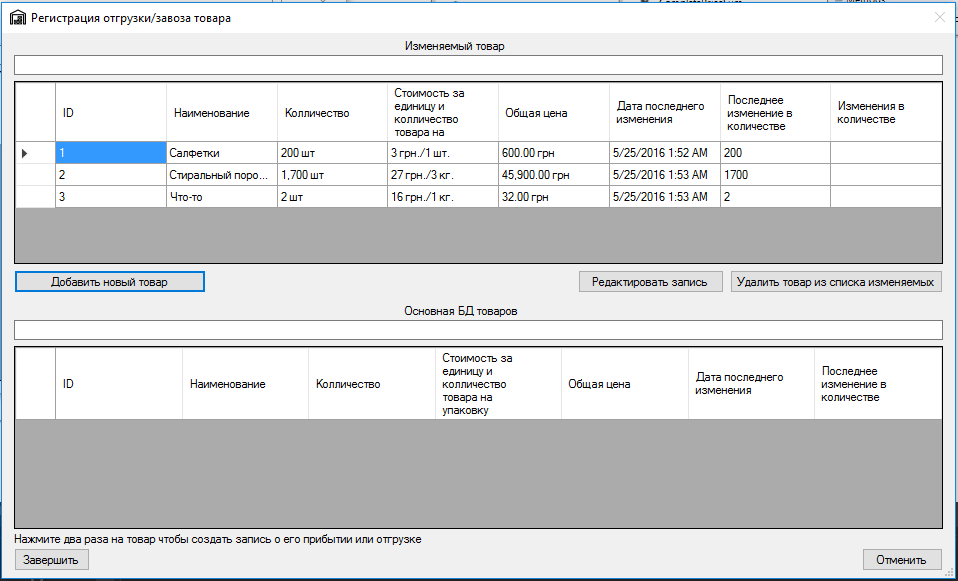


Рисунок 3.3 – Реєстрація прибуття або відгрузки товару

Додавання товару:

А) Викликати форму кнопкою

Б) Увести правильні дані у поля

В) Настиснути «Добавить»

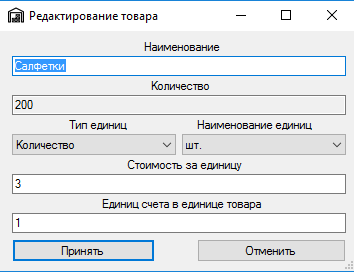


Рисунок 3.4 – Додавання чи редагування товару

Редагування товару:

А) Викликати форму кнопкою

Б) Увести правильні дані у поля

В) Настиснути «Добавить»

Друк відомості у файл:

А) Натиснути у головній формі кнопку «Экспорт» та вибрати у її підменю пункт «В файл»

Б) Вибрати місце для збереження файлу

В) Натиснути зберегти.

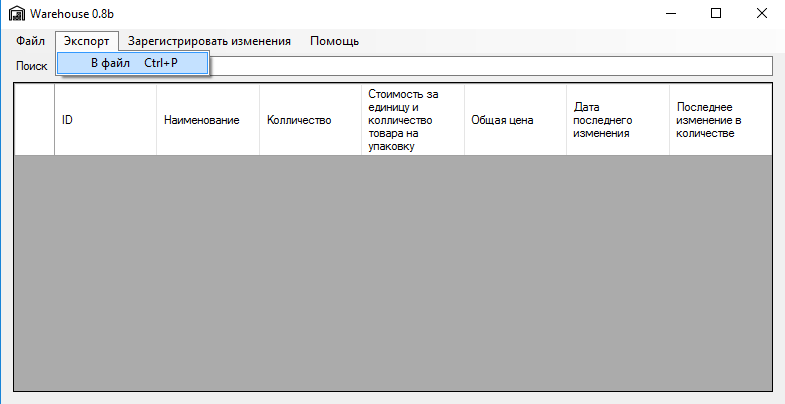


Рисунок 3.5 – Друк відомості у файл

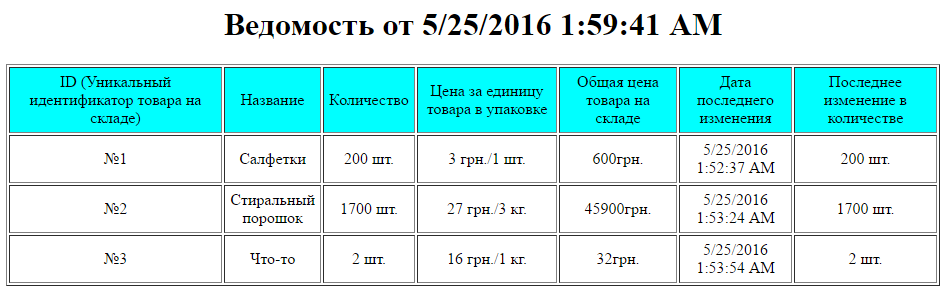


Рисунок 3.6 - Результат збереження

ВИСНОВКИ

Під час виконнання курсового проекту та супроводження програмного продукту мною були пройдені всі шляхи розробки даної програми, починаючи з постановки задачі до тестування і налагодження програми.

У результаті розробки проекту була створена прикладна програма «Склад» на платформі .Net з використанням мови C#.

Програма дозволяє зберігати, редагувати та вести облік товарів на складському приміщені.

Мною були докладно розглянуті такі аспекти .Net: основні принципи об’єктно-орієнтованого програмування, LINQ запити, двоїчна серіалізація об’єктів, делегати, події, ітератори та індексатори. Був отриманий досвід у проектуванні, реалізації, документуванні повноцінної програми.

У майбутньому планується виконання деяких пунктів для покращення програми, а саме: переробка дизайну і використання бази даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. Язык программирования C#. Классика Computers Science. 4-е издание= C# Programming Language (Covering C# 4.0), 4th Ed. – СПб.: «Питер», 2012. – 784 с.
2. Бондарев В. М. «Объектно-ориентированное программирование на С#» [Текст]: учеб. пособ / В. М. Бондарев. – Х.: Смит, 2009. – 224 с.;
3. Дейт, К. Дж. Введення у системи баз даних [Текст] / К. Дж. Дейт.- 7-е вид. – М. : Вид. дім «Вільямс», 2001. – 846 с.
4. Троелсен: «Мова програмування С# 2005 та платформа .NET 2.0» [Текст]: учеб. пособ, 3е видання. : Пер. з англ. – Москва. : ООО «И.Д. Вільямс», 2007. – 1168с.
5. Ульман, Дж. Д. Основи систем баз даних [Текст] / Дж. Д. Ульман - М. : Фінанси і статистика, 1983. – 334 с.

Додаток А – Код програми

//Главная форма приложения

**public** **partial** **class** **MainForm** : Form

{

//Внешняя форма для регистрации отгрузки/поступления товара

**private** WarehouseEditForm EditForm;

**public** DataGridView OuterAccessToDBView => MainDataView;

**public** **object** MainDataViewSource

{

**get** { **return** MainDataView.DataSource; }

**set** { MainDataView.DataSource = **value**; }

}

**private** Warehouse \_dataCollection;

**public** **MainForm**()

{

InitializeComponent();

}

//Действия выполняемые при вызове формы

**private** **void** **Form1\_Load**(**object** sender, EventArgs e)

{

Warehouse n = **new** Warehouse();

\_dataCollection = n;

MainDataViewSource = \_dataCollection;

SetDataFormats(MainDataView);

\_dataCollection.OnCollectionChange += () => RebindDataView(MainDataView,\_dataCollection);

}

//Обработчки нажатия на кнопку удаления из базы данных склада

**private** **void** **DeleteButton\_Click**(**object** sender, EventArgs e)

{

**try**

{

**object** toDelete = MainDataView.CurrentRow.DataBoundItem; //Can cause exception

**if** (MessageBox.Show("Вы уверены что хотите удалить выделеную запись?", "Подтверждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Question,

MessageBoxDefaultButton.Button1) == DialogResult.Yes)

{

\_dataCollection.Remove((Item)toDelete);

}

}

**catch** (NullReferenceException)

{

MessageBox.Show("База данных не загружена!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1);

}

}

//Вывод ведомости в HTML файл

**private** **void** **PrintDBToDocButton\_Click**(**object** sender, EventArgs e)

{

SaveFileDialog saveFile = **new** SaveFileDialog

{

FileName = @"report.htm",

Filter = "All files (\*.htm)|\*.htm"

};

**if** (saveFile.ShowDialog() == DialogResult.OK)

{

**object** path = saveFile.FileName;

**using** (FileStream fs = **new** FileStream(path.ToString(), FileMode.Create))

{

**using** (StreamWriter writer = **new** StreamWriter(fs, Encoding.UTF8))

{

writer.WriteLine("<h1 style=\"text-align: center\">Ведомость от " + DateTime.Now + "</h1>");

writer.Write("<table cellspacing=\"2\" border=\"1\" cellpadding=\"5\" style=\"text-align: center\">");

writer.WriteLine("<tr style=\"background-color: cyan\"><td>ID (Уникальный идентификатор товара на складе)</td>" +

"<td>Название</td>" +

"<td>Количество</td>" +

"<td>Цена за единицу товара в упаковке</td>" +

"<td>Общая цена товара на складе</td>" +

"<td>Дата последнего изменения</td>" +

"<td>Последнее изменение в количестве</td></tr>");

**foreach** (Item item **in** \_dataCollection)

{

writer.WriteLine("<tr>" +

"<td>№"+ item.ID + "</td>" +

"<td>" + item.Name + "</td>" +

"<td>" + item.Quanity + " шт. </td>" +

"<td>" + item.Units.ToString() + "</td>" +

"<td>" + item.CompletePriceSum + "грн. </td>" +

"<td>" + item.LastEntry + "</td>" +

"<td>" + item.LastQuanityChange + " шт. </td>" +

"</tr>");

}

writer.Write("</table>");

}

}

MessageBox.Show(@"Ведомость создана");

}

}

//Обработчик поля поиска, вызывается при изменении текста

**private** **void** **SearchBox\_TextChanged**(**object** sender, EventArgs e) => searchInWarehouseDGV(MainDataView,SearchBox);

//Открытие файлы (Десериализация)

**private** **void** **ReadFileButton\_Click**(**object** sender, EventArgs e)

{

IFormatter formatter = **new** BinaryFormatter();

OpenFileDialog openFile = **new** OpenFileDialog

{

FileName = "DB.bin",

Filter = "Binary DB (\*.bin)|\*.bin|All files (\*.\*)|\*.\*"

};

**if** (openFile.ShowDialog() == DialogResult.OK)

{

Stream stream = **new** FileStream(openFile.FileName, FileMode.Open, FileAccess.Read, FileShare.Read);

\_dataCollection = (Warehouse)formatter.Deserialize(stream);

stream.Close();

RebindDataView(MainDataView, \_dataCollection);

\_dataCollection.OnCollectionChange += () => RebindDataView(MainDataView, \_dataCollection);

}

}

//Сохранение файла (Сериализация)

**private** **void** **WriteFileButton\_Click**(**object** sender, EventArgs e)

{

IFormatter formatter = **new** BinaryFormatter();

SaveFileDialog saveFile = **new** SaveFileDialog

{

FileName = "United.bin",

Filter = "Binary DB (\*.bin)|\*.bin|All files (\*.\*)|\*.\*"

};

**if** (saveFile.ShowDialog() == DialogResult.OK)

{

**using** (

FileStream stream = **new** FileStream(saveFile.FileName, FileMode.Create, FileAccess.Write, FileShare.None)

)

{

formatter.Serialize(stream, **new** Warehouse(\_dataCollection, \_dataCollection.\_idCounter));

stream.Close();

}

}

}

//Вызов формы отгрузки/прибытия товара и дальнейшее добавление/замена товаров в главной коллекции

**private** **void** **DeliverToWarehouseButton\_Click**(**object** sender, EventArgs e)

{

EditForm = **new** WarehouseEditForm();

EditForm.ShowDialog(**this**);

**if**(EditForm.DialogResult == DialogResult.OK)

**foreach** (Item item **in** (Warehouse) EditForm.outerEdit)

{

\_dataCollection.AddWithID(item, **true**);

}

}

//Обновления таблицы отображения

**public** **void** **RefreshDataView**(DataGridView view, **object** data)

{

CurrencyManager cm = (CurrencyManager)(view.BindingContext[data]);

**if** (cm != **null**)

{

cm.Refresh();

}

SetDataFormats(view);

}

//Аналогично предыдущему, но другим способом

**public** **void** **RebindDataView**(DataGridView view, **object** data)

{

view.DataSource = **null**;

view.DataSource = data;

SetDataFormats(view);

}

//Установка форматов данных для отображения

**public** **void** **SetDataFormats**(DataGridView DGV)

{

**try**

{

DGV.Columns["CompletePriceSum"].DefaultCellStyle.Format = ("#,###.00 грн.");

DGV.Columns["Quanity"].DefaultCellStyle.Format = ("#,## шт.;(#,## шт.)");

//DGV.Columns["LastQuanityChange"].DefaultCellStyle.Format = ("#,## шт.;(#,## шт.)");

}

**catch** (NullReferenceException)

{

**throw** **new** **ArgumentException**("Bad DGV");

}

}

//Основной метод поиска в таблице отображения

**public** **void** **searchInWarehouseDGV**(DataGridView view, TextBox box)

{

**foreach** (**object** row **in** view.Rows)

{

**if**(((DataGridViewRow)row).Index == -**1**) **continue**;

((DataGridViewRow)row).Visible = **true**;

}

**if** (box.Text == "") **return**;

view.CurrentCell = **null**;

**for** (**int** i = **0**; i < view.RowCount; i++)

{

**if** (!((Item)view.Rows[i].DataBoundItem).Name.ToLower().Contains(box.Text.ToLower()))

view.Rows[i].Visible = **false**;

}

}

//Вызов окна "о программе"

**private** **void** **AboutToolStripMenuItem\_Click**(**object** sender, EventArgs e)

{

AboutForm about = **new** AboutForm();

about.ShowDialog();

}

}

**public** **partial** **class** **WarehouseEditForm** : Form

{

//Ссылка на основную форму

**private** MainForm n1;

//Коллекция которая содержит изменяемые элементы

**private** Warehouse newEdit;

//Реализация внешнего доступа к изменяемой коллекции

**public** **object** outerEdit => newEdit;

//Отключение кнопки "Закрыть"

**private** **const** **int** CP\_NOCLOSE\_BUTTON = **0**x200;

**protected** **override** CreateParams CreateParams

{

**get**

{

CreateParams myCp = **base**.CreateParams;

myCp.ClassStyle = myCp.ClassStyle | CP\_NOCLOSE\_BUTTON;

**return** myCp;

}

}

**public** **WarehouseEditForm**()

{

InitializeComponent();

}

//Действия которые необходимо совершить для загрузки и полноразмерной функциональности

**private** **void** **WarehouseEditForm\_Load**(**object** sender, EventArgs e)

{

newEdit = **new** Warehouse(((Warehouse)((MainForm)Owner).MainDataViewSource).Count);

MainEditDataView.DataSource = newEdit;

**foreach** (DataGridViewColumn column **in** MainEditDataView.Columns)

{

column.SortMode = DataGridViewColumnSortMode.NotSortable;

}

n1 = **new** MainForm();

OldMainDataView.DataSource = ((MainForm)Owner).MainDataViewSource;

**foreach** (DataGridViewColumn column **in** MainEditDataView.Columns) column.ReadOnly = **true**;

CreateUnboundTextBoxColumn();

n1.SetDataFormats(OldMainDataView);

newEdit.OnCollectionChange += () => RefreshMainEditDataView(newEdit);

}

//Создание не привязанной к данным колонки

**private** **void** **CreateUnboundTextBoxColumn**()

{

DataGridViewTextBoxColumn bc = **new** DataGridViewTextBoxColumn();

bc.Name = "QChange";

bc.HeaderText = "Изменения в количестве";

bc.ReadOnly = **false**;

MainEditDataView.Columns.Insert(MainEditDataView.ColumnCount, bc);

}

//Кнопка отмены редактирования

**private** **void** **CancelFormButton\_Click**(**object** sender, EventArgs e)

{

**if** (

MessageBox.Show("Вы уверенны что хотите отменить все изменения?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button1) ==

DialogResult.Yes)

{

DialogResult = DialogResult.Cancel;

Close();

}

}

//Добавление в коллекцию редактирования из основной коллекции

**private** **void** **OldMainDataView\_CellMouseDoubleClick**(**object** sender, DataGridViewCellMouseEventArgs e)

{

**if** (OldMainDataView.SelectedCells != **null** && OldMainDataView.SelectedCells.Count != **0**)

{

**int** rowIndex = OldMainDataView.SelectedCells[**0**].RowIndex;

**if** (rowIndex == -**1**) **return**;

Item workingItem = (Item) OldMainDataView.Rows[rowIndex].DataBoundItem;

**if** (newEdit.Contains(workingItem))

{

MessageBox.Show("Товар уже есть в списке изменяемых!");

}

**else**

{

newEdit.AddWithID((Item) OldMainDataView.Rows[rowIndex].DataBoundItem);

RefreshMainEditDataView(newEdit);

}

}

}

//Обновление основной таблицы редактирования

**private** **void** **RefreshMainEditDataView**(**object** source)

{

List<**object**> temp = **new** List<**object**>();

List<**bool**> temp2 = **new** List<**bool**>();

**for** (**int** i = **0**; i < MainEditDataView.RowCount; i++)

{

**var** **value** = MainEditDataView["QChange", i].Value;

**if** (**value** != **null** && **value**.ToString() != "")

temp.Add(Convert.ToInt32(MainEditDataView["QChange", i].Value));

**else** temp.Add("");

temp2.Add(Convert.ToBoolean(MainEditDataView["QChange", i].ReadOnly));

}

n1.RefreshDataView(MainEditDataView, source);

**for** (**int** i = **0**; i < temp.Count; i++)

{

MainEditDataView["QChange", i].Value = temp[i];

MainEditDataView["QChange", i].ReadOnly = temp2[i];

}

}

//Кнопка удаления из окна редактирования

**private** **void** **DeleteFromEditButton\_Click**(**object** sender, EventArgs e)

{

**try**

{

**object** toDelete = MainEditDataView.CurrentRow.DataBoundItem; //Can cause exception

**if** (MessageBox.Show("Вы уверены что хотите удалить выделеную запись?", "Подтверждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Question,

MessageBoxDefaultButton.Button1) == DialogResult.Yes)

{

newEdit.Remove((Item)toDelete);

n1.RefreshDataView(MainEditDataView, newEdit);

}

}

**catch** (NullReferenceException)

{

//TODO: Edit this ;D

MessageBox.Show("Ничто не выбрано, все тлен", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1);

}

}

**private** **void** **AddToEditButton\_Click**(**object** sender, EventArgs e)

{

OneItemEditForm editForm = **new** OneItemEditForm();

**if** (editForm.ShowDialog() == DialogResult.OK)

{

newEdit.AddWithID((Item)editForm.ReturnedItem);

RefreshMainEditDataView(newEdit);

MainEditDataView["QChange", MainEditDataView.RowCount - **1**].ReadOnly = **true**;

}

}

//Обработчики событий изменения текста в строках поиска

**private** **void** **NewSearchBox\_TextChanged**(**object** sender, EventArgs e) => n1.searchInWarehouseDGV(MainEditDataView, NewSearchBox);

**private** **void** **OldSearchBox\_TextChanged**(**object** sender, EventArgs e) => n1.searchInWarehouseDGV(OldMainDataView,OldSearchBox);

**private** **void** **FinalizeFormButton\_Click**(**object** sender, EventArgs e)

{

**foreach** (Item item **in** newEdit)

{

**var** **value** = MainEditDataView["QChange", MainEditDataView.RowCount - **1**].Value;

**if** (**value** != **null** && **value**.ToString() != "")

**try**

{

item.InvChangeNow(Convert.ToInt32(MainEditDataView["QChange", MainEditDataView.RowCount - **1**].Value));

}

**catch** (InvalidDataException)

{

MessageBox.Show("Неправильное количество позиции " + item.Name +

". Попробуйте изменить значение и попробовать еще раз.");

**return**;

}

}

DialogResult = DialogResult.OK;

Close();

}

//Редактирование одного элемента

**private** **void** **EditOnceButton\_Click**(**object** sender, EventArgs e)

{

OneItemEditForm dialog = **new** OneItemEditForm();

**try**

{

dialog.GetSelectedFromFatherForm(MainEditDataView);

}

**catch** (NoNullAllowedException)

{

**return**;

}

**if** (dialog.ShowDialog() == DialogResult.OK)

{

**int?** tmp = ((Item) MainEditDataView.CurrentRow.DataBoundItem).ID;

((Item) dialog.ReturnedItem).ID = tmp;

newEdit.AddWithID((Item)dialog.ReturnedItem,**true**);

}

}

**private** **void** **MainEditDataView\_CellEndEdit**(**object** sender, DataGridViewCellEventArgs e)

{

**if** (!(**new** Regex(@"^\d+$")).IsMatch(MainEditDataView[e.ColumnIndex, e.RowIndex].Value.ToString()))

{

MainEditDataView[e.ColumnIndex, e.RowIndex].Value = "";

MessageBox.Show("Вы ввели неверное значение, повторите попытку");

}

}

}

//Форма редактирования единичного товара

**public** **partial** **class** **OneItemEditForm** : Form

{

// Свойство для доступа к возвращаемому товару из вне

**public** **object** ReturnedItem => \_innerItem;

// Возвращаемый товар

**private** Item \_innerItem;

**public** **OneItemEditForm**()

{

InitializeComponent();

}

// Обработчик кнопки отмены

**private** **void** **EditCancelButton\_Click**(**object** sender, EventArgs e)

{

**if** (

MessageBox.Show("Вы действительно хотите отменить изменения?", "Предупреждение", MessageBoxButtons.YesNo,

MessageBoxIcon.Question, MessageBoxDefaultButton.Button1) == DialogResult.Yes)

{

**this**.DialogResult = DialogResult.Cancel;

Close();

}

}

// Проверка формы

**public** **bool** **ValidateForm**()

{

Regex numsInt = **new** Regex(@"^\d+$");

Regex numsFloat = **new** Regex(@"\-?\d+(\.\d{0,})?");

**bool** expressionResult = (numsInt.IsMatch(QuanityTextBox.Text) &&

numsFloat.IsMatch(PricePerUnitTextBox.Text) &&

numsFloat.IsMatch(UnitsPerItemTextBox.Text));

**return** expressionResult;

}

// Принятие изменений

**private** **void** **EditAcceptButton\_Click**(**object** sender, EventArgs e)

{

**try**

{

**if** (!ValidateForm()) **throw** **new** DataException();

ADTUnits type;

**switch** (UnitTypeComboBox.Text)

{

**case** "Вес":

type = **new** Weight(Convert.ToDouble(PricePerUnitTextBox.Text),

Convert.ToDouble(UnitsPerItemTextBox.Text), UnitFactorComboBox.Text);

**break**;

**case** "Объем":

type = **new** Volume(Convert.ToDouble(PricePerUnitTextBox.Text),

Convert.ToDouble(UnitsPerItemTextBox.Text), UnitFactorComboBox.Text);

**break**;

**case** "Количество":

type = **new** UnitsOnly(Convert.ToDouble(PricePerUnitTextBox.Text),

Convert.ToDouble(UnitsPerItemTextBox.Text));

**break**;

**default**:

type = **new** UnitsOnly(Convert.ToDouble(PricePerUnitTextBox.Text),

Convert.ToDouble(UnitsPerItemTextBox.Text));

**break**;

}

\_innerItem = **new** Item(NameTextBox.Text, Convert.ToDouble(QuanityTextBox.Text), type);

**this**.DialogResult = DialogResult.OK;

**this**.Close();

}

**catch** (DataException)

{

MessageBox.Show("Проверьте правильность ввода полей", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Hand, MessageBoxDefaultButton.Button1);

}

}

// События по изменению индекса

**private** **void** **UnitTypeComboBox\_SelectedIndexChanged**(**object** sender, EventArgs e)

{

UnitFactorComboBox.Items.Clear();

**switch** (UnitTypeComboBox.Text)

{

**case** "Вес":

**foreach** (**var** item **in** **new** Weight(**1**, **1**, "кг.").FactDictionary.Keys)

{

UnitFactorComboBox.Items.Add(item);

UnitFactorComboBox.Text = "кг.";

}

**break**;

**case** "Объем":

**foreach** (**var** item **in** **new** Volume(**1**, **1**, "л.").FactDictionary.Keys)

{

UnitFactorComboBox.Items.Add(item);

UnitFactorComboBox.Text = "л.";

}

**break**;

**case** "Количество":

UnitFactorComboBox.Items.Add("шт.");

UnitFactorComboBox.Text = "шт.";

**break**;

}

}

// Метод для перехвата выбраной в таблице строки

**public** **void** **GetSelectedFromFatherForm**(DataGridView DGV)

{

**if** (DGV.CurrentRow == **null**)

{

MessageBox.Show("Ничего не выбрано");

**throw** **new** **NoNullAllowedException**();

}

Item worker = (Item) DGV.CurrentRow.DataBoundItem;

NameTextBox.Text = worker.Name;

UnitsPerItemTextBox.Text = worker.Units.Quanity.ToString();

UnitFactorComboBox.Text = worker.Units.Factor;

PricePerUnitTextBox.Text = worker.Units.Price.ToString();

QuanityTextBox.Text = worker.Quanity.ToString();

QuanityTextBox.ReadOnly = **true**;

Type unitType = worker.Units.GetType();

**switch** (unitType.Name)

{

**case** "Volume":

UnitTypeComboBox.Text = "Объем";

**break**;

**case** "Weight":

UnitTypeComboBox.Text = "Вес";

**break**;

**default**:

UnitTypeComboBox.Text = "Количество";

**break**;

}

UnitFactorComboBox.Text = worker.Units.Factor;

}

}

**public** **partial** **class** **AboutForm** : Form

{

**public** **AboutForm**()

{

InitializeComponent();

}

**private** **void** **AboutForm\_Load**(**object** sender, EventArgs e)

{

MainPicDrawer.Image = Resources.buildings64;

}

**private** **void** **CloseButton\_Click**(**object** sender, EventArgs e)

{

Close();

}

}

[Serializable]

**class** **Warehouse** : BindingList<Item>, ICloneable

{

//Счетчик для уникальных идентификаторов в коллекции

**public** **int** \_idCounter { **get**; **private** **set**; }

//Оригинальный (не сортированый)

**private** List<Item> innerList;

//Направление сортировки

**private** ListSortDirection sortDirection;

//Событие изменения коллекции

**public** **delegate** **void** **CollectionChanged**();

**public** **event** CollectionChanged OnCollectionChange;

//Поле, которое описывает название поля по которому в данный момент осуществляется сортировка

[NonSerialized] **private** PropertyDescriptor sortProperty;

//Сброс режима сортировки

Action<Warehouse, List<Item>>

populateBaseList = (a, b) => a.ResetItems(b);

//Словарь методов для сортировки списка, при отсутствии в нем методя для подходящего свойства создается новый и сохраняется в этом словаре

**private** **static** Dictionary<**string**, Func<List<Item>, IEnumerable<Item>>>

cachedOrderByExpressions = **new** Dictionary<**string**, Func<List<Item>, IEnumerable<Item>>>();

**public** **Warehouse**()

{

innerList = **new** List<Item>();

\_idCounter = **0**;

}

**public** **Warehouse**(**int** idCounter) : **this**()

{

\_idCounter = idCounter;

}

**public** **Warehouse**(IEnumerable<Item> listToClone)

{

innerList = listToClone.ToList();

populateBaseList(**this**, innerList);

\_idCounter = **0**;

}

**public** **Warehouse**(IEnumerable<Item> listToClone, **int** idCounter) : **this**(listToClone)

{

\_idCounter = idCounter;

}

//Имплементация методов для сортируемого списка типа BindingList

**protected** **override** **void** **ApplySortCore**(PropertyDescriptor prop,

ListSortDirection direction)

{

sortProperty = prop;

**var** orderByMethodName = sortDirection ==

ListSortDirection.Ascending

? "OrderBy"

: "OrderByDescending";

**var** cacheKey = **typeof**(Item).GUID + prop.Name + orderByMethodName;

**if** (!cachedOrderByExpressions.ContainsKey(cacheKey))

{

CreateOrderByMethod(prop, orderByMethodName, cacheKey);

}

ResetItems(cachedOrderByExpressions[cacheKey](innerList).ToList());

ResetBindings();

sortDirection = sortDirection == ListSortDirection.Ascending

? ListSortDirection.Descending

: ListSortDirection.Ascending;

}

//Создание метода сортировки для определенного поля

**private** **void** **CreateOrderByMethod**(PropertyDescriptor prop,

**string** orderByMethodName, **string** cacheKey)

{

**var** sourceParameter = Expression.Parameter(**typeof**(List<Item>), "source");

**var** lambdaParameter = Expression.Parameter(**typeof**(Item), "lambdaParameter");

**var** accesedMember = **typeof**(Item).GetProperty(prop.Name);

**var** propertySelectorLambda =

Expression.Lambda(Expression.MakeMemberAccess(lambdaParameter,

accesedMember), lambdaParameter);

**var** orderByMethod = **typeof**(Enumerable).GetMethods()

.Where(a => a.Name == orderByMethodName &&

a.GetParameters().Length == **2**)

.Single()

.MakeGenericMethod(**typeof**(Item), prop.PropertyType);

**var** orderByExpression = Expression.Lambda<Func<List<Item>, IEnumerable<Item>>>(

Expression.Call(orderByMethod,

**new** Expression[]

{

sourceParameter,

propertySelectorLambda

}),

sourceParameter);

cachedOrderByExpressions.Add(cacheKey, orderByExpression.Compile());

}

//Сброс сортировки

**protected** **override** **void** **RemoveSortCore**()

{

ResetItems(innerList);

}

//Сброс сортировки с заменой

**private** **void** **ResetItems**(List<Item> items)

{

**base**.ClearItems();

**for** (**int** i = **0**; i < items.Count; i++)

{

**base**.InsertItem(i, items[i]);

}

}

//Сигнализация того, что список поддерживает сортировку

**protected** **override** **bool** SupportsSortingCore

{

**get** { **return** **true**; }

}

//Направление сортировки

**protected** **override** ListSortDirection SortDirectionCore

{

**get** { **return** sortDirection; }

}

//Свойство по которому в данный момент осуществляется сортировка

**protected** **override** PropertyDescriptor SortPropertyCore

{

**get** { **return** sortProperty; }

}

//Добавление с заменой или без по уникальному идентификатору

**public** **void** **AddWithID**(Item item, **bool** replaceFlag = **false**)

{

**if** (item.ID == **null**)

{

item.ID = ++\_idCounter;

Add(item);

}

**else**

{

**if** (Contains(item) && replaceFlag) **this**[IndexOfID(item)] = item;

**else**

{

//item.ID = ++\_idCounter;

Add(item);

}

}

OnCollectionChange?.Invoke();

}

//Глубокое копирование списка

**public** **object** **Clone**()

{

Warehouse result = **new** Warehouse();

**foreach** (Item item **in** **this**)

{

result.Add(item);

}

**return** result;

}

//Метод для проверки наличия определенного элемента в списке

**public** **new** **bool** **Contains**(Item item)

{

**foreach** (Item itemCol **in** **this**)

{

**if** (item.ID == itemCol.ID) **return** **true**;

}

**return** **false**;

}

//Метод для поиска индекса определнного элемента в списке

**public** **int** **IndexOfID**(Item item)

{

**for** (**int** i = **0**; i < Count; i++)

{

**if** (**this**[i].ID == item.ID) **return** i;

}

**return** -**1**;

}

//Метод удаления из списка и дальнейшая сигнализация о изменении в списке

**public** **new** **void** **Remove**(Item item)

{

**base**.Remove(item);

OnCollectionChange?.Invoke();

}

}

//Класс содержащий название,колличество и т.п. характеристики товара на складе

[Serializable]

**class** **Item** : IComparable<Item>, ICloneable

{

**private** **int?** \_ID;

[DisplayName("ID")]

**public** **int?** ID

{

**get** { **return** \_ID; }

**set** { \_ID = **value**; }

}

[DisplayName("Наименование")]

**public** **string** Name { **get**; **set**; }

[DisplayName("Колличество")]

**public** **double** Quanity { **get**; **set**; }

[DisplayName("Стоимость за единицу и колличество товара на упаковку")]

**public** ADTUnits Units { **get**; **set**; }

[DisplayName("Общая цена")]

**public** **double** CompletePriceSum => Quanity \* Units.Price;

[DisplayName("Дата последнего изменения")]

**public** DateTime LastEntry => ItemLog.Last().LastDate;

[DisplayName("Последнее изменение в количестве")]

**public** **double** LastQuanityChange => ItemLog.Last().QuanityChange;

**private** List<LastEntry> ItemLog { **get**; }

**public** **Item**(**string** name, **double** quanity, ADTUnits units) : **this**(name,quanity,units,**new** LastEntry(quanity)) { }

**public** **Item**(**string** name, **double** quanity, ADTUnits units, LastEntry le)

{

ItemLog = **new** List<LastEntry>();

Name = name;

Quanity = quanity;

Units = units;

ItemLog.Add(le);

}

//Изменение колличества товара с пометкой по времени в данный момент

**public** **void** **InvChangeNow**(**double** qChange)

{

ItemLog.Add(**new** LastEntry(qChange,DateTime.Now));

**if**(Quanity + qChange < **0**) **throw** **new** InvalidDataException("Bad quanity");

Quanity += qChange;

}

//Если идентификатор товара не равен идентификатору другого товара, то объекты не равны

**public** **override** **bool** **Equals**(**object** obj)

{

**if** (obj.GetType() == **this**.GetType())

{

**return** (((Item) obj).ID == ID);

}

**return** **false**;

}

//Сравнение по колличеству

**public** **int** **CompareTo**(Item other) => Quanity.CompareTo(other.Quanity);

//Конструктор клонирования объекта

**public** **object** **Clone**()

{

Item result = **new** Item(Name, Quanity, Units);

result.ItemLog.AddRange(ItemLog.AsEnumerable());

result.ID = ID;

**return** result;

}

}

// Класс который содержит информацию о последнем изменении товара

[Serializable]

**class** **LastEntry**

{

**public** DateTime LastDate;

**public** **double** QuanityChange;

**public** **LastEntry**(**double** qChange) : **this**(qChange, DateTime.Now) { }

**public** **LastEntry**(**double** qChange, DateTime time)

{

QuanityChange = qChange;

LastDate = time;

}

}

**public** **interface** INumInSI

{

//Колличество

**double** Quanity { **get**; **set**; }

//Приставка которая сигнализует о размерности

**string** Factor { **get**; **set**; }

//Словарь приставок и их множителей

Dictionary<**string**, **double**> FactDictionary { **get**; **set**; }

//Метод возвращающий номер в единцах измерения СИ

**double** **GetSINumber**();

}

[Serializable]

**abstract** **class** **ADTUnits** : INumInSI, IComparable<ADTUnits>

{

**public** **double** Quanity { **get**; **set**; }

**public** **string** Factor

{

**get** { **return** \_factor; }

**set**

{

**if**(FactDictionary.ContainsKey(**value**)) **throw** **new** ArgumentException("Invalid factor");

\_factor = **value**;

}

}

**private** **string** \_factor;

**public** Dictionary<**string**, **double**> FactDictionary { **get**; **set**; }

**public** **double** Price { **get**; **set**; }

**public** **virtual** **double** **GetSINumber**()

{

**return** Quanity \* FactDictionary[Factor];

}

**public** **virtual** **void** **SetToFactor**(**string** newFact)

{

Quanity = GetSINumber() / FactDictionary[newFact];

Factor = newFact;

}

**public** **override** **string** **ToString**()

{

**return** (Price + " грн./" + Quanity + " " + Factor);

}

**public** **virtual** **int** **CompareTo**(ADTUnits other)

{

**return** Price.CompareTo(other.Price);

}

}

//Вес

[Serializable]

**sealed** **class** **Weight** : ADTUnits

{

**public** **Weight**(**double** price, **double** quanity, **string** factor)

{

FactDictionary = **new** Dictionary<**string**, **double**>();

Quanity = quanity;

Factor = factor;

Price = price;

FactDictionary.Add("т.", **1000**); //Тонна

FactDictionary.Add("ц.", **100**); //Центнер

FactDictionary.Add("кг.", **1**); //Килограмм

FactDictionary.Add("г.", **0.001**); //Грамм

FactDictionary.Add("мг.", **0.000001**); //Миллиграмм

**if** (!FactDictionary.ContainsKey(Factor)) **throw** **new** ArgumentException("Invalid factor");

}

}

//Объем

[Serializable]

**sealed** **class** **Volume** : ADTUnits

{

**public** **Volume**(**double** price, **double** quanity, **string** factor)

{

FactDictionary = **new** Dictionary<**string**, **double**>();

Quanity = quanity;

Factor = factor;

Price = price;

FactDictionary.Add("л.", **1**); //Литр

FactDictionary.Add("мл.", **0.001**); //Грамм

FactDictionary.Add("мг.", **0.000001**); //Миллиграмм

**if** (!FactDictionary.ContainsKey(Factor)) **throw** **new** ArgumentException("Invalid factor");

}

}

//Количество

[Serializable]

**sealed** **class** **UnitsOnly** : ADTUnits

{

**public** **UnitsOnly**(**double** price, **double** quanity)

{

FactDictionary = **new** Dictionary<**string**, **double**>();

Quanity = quanity;

Factor = "шт.";

Price = price;

}

**public** **override** **double** **GetSINumber**() => Quanity;

**public** **override** **void** **SetToFactor**(**string** newFact)

{

**throw** **new** **InvalidOperationException**();

}

}

[A](http://hilite.me/api)