Міністерство освіти і науки України

Запорізька державна інженерна академія

Факультет інформаційних та електронних технологій

Кафедра програмного забезпечення автоматизованих систем

**КУРСОВА РОБОТА**

**з курсу Об’єктно-орієнтоване програмування**

тема: Програмне забезпечення вбудованого процесора холодильника

Спеціальність „Програмна інженерія”

Виконав студент

групи СП-15-1

Кириченко Олександр

Керівник роботи

Доцент

Попівщий В.І.

Оцінка

Члени комісії:

доц. Попівщий В.І.

зав. каф Вербицький В.Г.

доц. Безверхий А.І.

Запоріжжя @ 2016

**Розділ 1.Вступ**

Потрібно розробити модель програмного забезпечення вбудованого процесора холодильника.

Холодильник складається з кількох холодильних камер для зберігання продуктів. У кожній холодильній камері є регулятор температури, мотор, термометр, індикатор, таймер, датчик відкриття дверей камери і пристрій для подачі звукових сигналів.

За допомогою терморегулятора встановлюється максимально припустима температура самій камері. Мотор призначений для підтримки низької температури. Термометр постійно вимірює температуру всередині камери, а індикатор температури, розташований на дверцятах, постійно висвічує її значення. При підвищенні температури вище межі, що визначається поточним положенням регулятора, включається мотор. При зниженні температури нижче деякого іншого значення, пов'язаного з першим, мотор відключається.

Доступ в камеру здійснюється через дверцята. Якщо двері холодильної камери відкрита протягом занадто довгого часу, подається звуковий сигнал. Звуковий сигнал також подається в будь-яких позаштатних ситуаціях (наприклад, при поломці мотора).

Холодильник веде журнал, в якому зазначаються всі події, що відбуваються:

• зміна положення терморегулятора камери;

• включення і відключення мотора;

• доступ в камеру;

• позаштатні ситуації.

Вміст журналу може бути передано в комп'ютер, приєднаний до спеціального гнізда на корпусі холодильника.

**Розділ 1.1 Глосарій**

**Лямбда-вираз** (в програмуванні) - це спеціальний синтаксис для оголошення анонімних функцій за місцем їх використання. Використовуючи лямбда-вирази, можна оголошувати функції в будь-якому місці коду. Зазвичай лямбда-вираз допускає замикання на лексичний контекст, в якому цей вираз використано.

Лямбда-вирази приймають дві форми. Форма, яка найбільш прямо замінює анонімний метод, являє собою блок коду, укладений у фігурні дужки це - пряма заміна анонімних методів. Лямбда-вирази, з іншого боку, надають ще більш скорочений спосіб оголошувати анонімний метод не вимагає ні коду у фігурних дужках, ні оператора return. Обидва типи лямбда-виразів можуть бути перетворені в делегати.

У всіх лямбда-виразах використовується лямбда-оператор =>, який читається як «переходить в» (у мовах F # і PascalABC.NET використовується оператор ->). Ліва частина лямбда-оператора визначає параметри введення (якщо такі є), а права частина містить вираз або блок оператора.

**Делегат** - це тип, який представляє собою посилання на методи з певним списком параметрів і повертаним типом. При створенні екземпляра делегата цей екземпляр можна пов'язати з будь-яким методом з сумісною сигнатурою і повертаним типом. Метод можна викликати (активувати) за допомогою примірника делегата.

Делегати використовуються для передачі методів в якості аргументів до інших методів. Обробники подій - це нe що інше, як методи, що викликаються за допомогою делегатів. Ви створюєте свій метод, а клас, такий як елемент управління Windows, може викликати ваш метод при виникненні певної події.

**Подія** - це іменований делегат, при виклику якого, будуть запущені всі підписалися на момент виклику події методи заданої сигнатури.

**StreamWriter** - розроблений для виведення символів в певній кодуванні, тоді як класи, похідні від Stream, розроблені для введення і виведення байтів.

Якщо не вказано інакше, StreamWriter за замовчуванням використовує екземпляр UTF8Encoding. Цей екземпляр UTF8Encoding побудований без мітки порядку проходження байтів, тому використання методу GetPreamble повертає порожній масив байтів. Значення за замовчуванням - UTF-8 коду для даного конструктора створює виняток неприпустимі в байтах. Це розширення функціональності відрізняється від розширення функціональності, надане об'єктом кодування у властивості Encoding.UTF8. Щоб вказати специфікацію і визначити виникнення виключення на неприпустимі байти, використовуйте конструктор, який приймає як параметр об'єкт кодування, наприклад StreamWriter (String, Boolean, Encoding) або StreamWriter.

**Розділ 1.2 Опис предметної області**

Иан Грэхем. Объектно-ориентированные методы. Принципы и практика— 3-е изд. — М.: «Вильямс», 2004.

Антони Синтес. Освой самостоятельно объектно-ориентированное программирование за 21 день— М.: «Вильямс», 2002.

У будь-якому місці починаючи від офісу, закінчуючи будинком є ​​люди, у яких є природне бажання продовжити життя продуктам, які вони споживають. Тому існує така річ, як холодильник. Це пристрій, що підтримує низьку температуру в теплоізольованої камері. Застосовується зазвичай для зберігання їжі або предметів, що вимагають зберігання в прохолодному місці. У розвинених країнах побутовий холодильник є майже в кожній родині. Робота холодильника заснована на використанні холодильної машини, що переносить тепло з робочої камери холодильника назовні, де воно розсіюється в зовнішнє середовище. Існують також комерційні холодильники з більшою холодопродуктивністю, які використовуються на підприємствах громадського харчування і в магазинах і промислові холодильники, обсяг робочої камери яких може досягати десятків і сотень кубометрів, вони використовуються, наприклад, на м'ясокомбінатах, промислових виробництвах.

Холодильники можуть підрозділятися на два види: середньо температурні камери для зберігання продуктів і низькотемпературні морозильники.

**Морозильник** - окремий прилад або складова частина холодильника, призначений для заморожування і зберігання продуктів харчування.

**Розділ 1.3 Неформальна постановка задачі**

Програма призначена для забезпечення роботи вбудованого процесора холодильника. Інтерфейс повинен мати кнопку "Cold Store" для зміни морозильної камери, "Regime of freeze" для встановлення максимально допустимої температури в даній камері, кнопки для відкриття і закриття дверки холодильника, дисплей відображення температури в камері, повідомлень користувачеві, звукові індикатори про те, що дверцята холодильника відкрита досить довго і про те, що сталася позаштатна ситуація, а також можливість переглянути журнал подій або зберегти даний журнал на зовнішній пристрій.

**Розділ 1.4 Огляд існуючих методів рішення**

Для порівняння я взяв холодильник, який стоїть у кімнаті гуртожитку .В ньому той же функціонал. З переваг в моєму продукті можливість перегляді всіх подій які відбуваються в холодильнику, перегляд поточної температури в кожному відсіку морозильної камери, а також звуковий датчик який подається у разі якщо двері холодильної камери відкрита протягом занадто довгого часу. Звуковий сигнал також подається в будь-яких позаштатних ситуаціях (наприклад, при поломці мотора).

До розробки свого продукту я аналізував холодильники - аналоги. Проаналізувавши всі їх плюси і мінуси, я склав план реалізації своєї версії ПЗ для холодильника.

**Розділ 1.5 План робіт**

01.07.15 Було розроблено ТЗ

02.07.15 Складено план роботи, початок розробки

05.07.15 Перша alpha - версія програми

03.10.2016 Було розроблено Технiчне завдання

04.10.2016 План реалiзації проекту

11.10.2016 Перша alpha - версія програми

12.10.2016 Тестування alpha - версії

01.11.2016 Перехід до beta - версії

01.11.2016 Тестування beta - версії програми

25.11.2016 Остання версія програми

08.12.2016 Створення додатку

09.12.2016 Презентація застосунку

**Розділ 2. Вимоги до оточення**

Незважаючи на всі запевнення в тому, що холодильник наглухо ізольований від зовнішніх впливів, розміщувати його в будинку поблизу нагрівальних пристроїв строго не рекомендується. Краще всього поставити холодильне обладнання в вентильоване прохолодне місце. Поряд не повинно бути батарей опалення, обігрівачів, газових і електричних кухонних плит і великого вікна, через яке на холодильник будуть більшу частину дня потрапляти прямі сонячні промені. Найкраще буде поставити холодильник на протязі.

Якщо на вашій кухні встановлені підлоги з підігрівом, то холодильник не повинен бути встановлений на активних опалювальних елементах, що слід врахувати при установці холодильника або при облаштуванні підлог. Якщо в зоні холодильника будуть трубки обігрівальної системи, і підлога буде безперервно обігріватися, холодильний компресор працюватиме в постійному режимі, оскільки тепле повітря буде постійно оточувати пристрій. Безперервна робота компресора холодильника призведе до того, що техніка швидко вийде з ладу. Як мінімум, холодильника потрібно дорогий ремонт. Як максимум, доведеться йти за новим холодильником.

**Розділ 2.1 Вимоги до апаратного забезпечення**

Мінімальні вимоги до апаратного забезпечення:

**Система** Windows XP Sр3

**Процесор** з тактовою частотою не менше 300 MHz.

**Оперативна пам’ять** ємністю 128 Мб.

**Жорсткий диск** об’ємом 100Мб

**Звукова карта** або звукові виходи.

**Монітор** з розподільною здатністю 800х600.

Рекомендовані вимоги до апаратного забезпечення:

**Система** Windows 10

**Процесор** з тактовою частотою не менше 2 ГГц.

**Оперативна пам’ять** ємністю 1024Мб.

**Жорсткий диск** об’ємом 20Гб

**Графічний процесор** 128 Мб пам'яті (адаптер з підтримкою DirectX 9), драйвери WDDM версії 1.0 і старше.

**Звукова карта** або звукові виходи.

**Монітор** з розподільною здатністю 800х600.

**Розділ 2.2 Вимоги до програмного забезпечення**

Операційна Windows – подібна система**,** а також встановлений .Net Framework версії не меньшої за 4**.**

**Розділ 2.3 Вимоги до користувача**

Користувач повинен пройти базовий курс користування операційною системою Windows.

В цьому випадку програмне забезпечення не потребує спеціальних навичок користування.

.

**Розділ 3. Архітектура системи**

Проаналізувавши постановку завдання можна виділити наступні класи:

* Вікно журналу подій;
* Вікно для збереження файлу з подіями;
* TemperatureControl;.(відповідає за зміну температури терморегулятора)
* Engine;(контролює температуру відповідно до терморегулятора)
* Freezer;(виміряє температуру та встановлює коли увімкнути мотор )
* Refrigerator;(створює три морозильні камери та вмикає їх)
* ListHandlerEventArgs;(спадкує системний клас подій)
* JournalEntry;(клас обгортка для збереження події)
* TeamsJourna;(клас який реагуватиме на події)
* Вікно для перегляду журналу подій

А також інтерфейси:

-IMessageEvent;(подія для журналу)

- IThermometer;(подія для термометру)

**Розділ 3.1. Діаграма класів**





**Розділ 3.2. Основні атрибути і методи. Їх короткий опис**

**Клас TemperatureControl**

Атрибути:

double max Celsius;-максимальна межа температури

double min Celsius; -мінімальна межа температури

double currentTemperature; -поточна температура

string freezeName; -ім'я морозильної камери

Методи:

public void CelsiusValueChange (RegimeOfFreeze f\_regime) -зміна режиму морозильної камери

**Клас Engine**

Атрибути:

Temperature Control temperature; -терморегулятор, контрольована температура

public event RefregiratorEvents message; -подія виникає при включенні, виключенні і поломці мотора bool statusOfTheEngine; -Статус мотора true-включений, false-відключений

Методи:

public void EngineOn (); - включення мотора, температура знижується

public void EngineOff (); - вимкнення мотора, температура підвищується

public void CrushMotor (); - зламати мотор, температура підвищується, мотор перестає працювати

**Класс Freezer**

Атрибути:

Temperature Control idea Temperature; -терморегулятор

System.Windows.Threading.DispatcherTimer timer; -таймер для дверцята холодильника

Engine mazda; -Мотор

public event RefregiratorEvents message; -подія холодильника

public event TempuratureEvent show; -індикатор температури

Методи:

Temperature Control idea Temperature; -терморегулятор

System.Windows.Threading.DispatcherTimer timer; -таймер для дверцята холодильника

Engine mazda; -Мотор

public event RefregiratorEvents message; -подія холодильника

public event TempuratureEvent show; -індикатор температури

**Класс Refrigerator**

Атрибути:

Freezer upper Shelf; - перша морозильна камера

Freezer middle Shelf; - друга морозильна камера

Freezer lowerShelf; -третя морозильна камера

Методи:

public void ON();-увімкнути холодильник

**Класс ListHandlerEventArgs**

Атрибути:

public string FreezeName {get; set; } -ім'я Морозильної камери

public DateTime Time {get; set; } - Час події

public string Changes {get; set; } -подія яке відбулося

**Класс JournalEntry**

Атрибути:

public string FreezeName {get; set; } -ім'я морозильної камери

public DateTime Time {get; set; } - Час події

string EventName {get; set; } -подія яке відбулося

Методи:

string Information()-сформувати рядок який містить інформаційне поле класу

**Класс TeamsJournal**

Атрибути:

List<JournalEntry> events;-журнал подій

Методи:

public void NewEvents(object sender, ListHandlerEventArgs f\_info)-обробник подій

public IEnumerator GetEnumerator()-ітератор для формування рядка який містить усі події

public string Information()-повертає інформаційне поле класу

**Розділ 3.3. Відношення між класами**

У класі Temperature Control створюються поля які контролюють максимальне і мінімальне значення температури.

У класі Engine створюється екземпляр TemperatureControl, подія RefregiratorEvents (яке повідомляє коли вмикається і вимикається мотор). Клас призначений для контролю температури усередині морозильної камери, завдяки методам які включають і вимикають мотор.

Клас Freezer створює екземпляр Engine, TemperatureControl бере свідчення поточної температури в камері і вирішує включити або відключити мотор. Також клас дає можливість відкриття і закриття відсіку морозильної камери

Клас Refrigerator створює три екземпляри Freezer. Запускає мотори даних примірників.

Клас ListHandlerEventArgs успадковується від System.EventArgs для можливості створення нових полів у події.

Клас JournalEntry грає роль інформаційного поля для подій.

Клас TeamsJournal клас передплатник співпадає за сигнатурі події класу видавця.

**Розділ 3.4.Варіанти користування**

На діаграмі показана модель співвідношення ролей розробника і користувача до модулів та функціоналу програми. Діаграма максимальна проста і не розкриває засоби реалізації потокових дій, а розкриває лише напрямок розподілу ролей на класи і їхні можливості.



**Розділ 4. Проект програмної системі**

**Розділ 4.1. Засобі реалізації**

Модель програмного забезпечення була реалізована засобами мови С #. Середа розробки: Microsoft Visual Studio 2013.

**Розділ «4.2 Проект інтерфейсу»**

Вигляд програми:



**Розділ 4.3. Реалізація та тестування**

Фізичні характеристики даної системи:

1) Об’єм коду в строках– 500.

2) Об’єм коду в КБ– 400.

3) Кількість форм – 3.

4) Витрати оперативної пам'яті МБ ~8.

5) Затрати часу на обробку основних операцій~ 0.4 сек.

**Висновки**

* У процесі виконання курсової роботи було розроблено модель програмного забезпечення вбудованого процесора холодильника засобами мови С #
* Було спроектовано об'єктно-орієнтовану модель програмного забезпечення: виділено основні класи, встановлені залежності між ними і побудовано діаграму класів. Засвоєні принципи об'єктно - орієнтованого програмування (успадкування, інкапсуляція, поліморфізм), залежність між класами.
* Отримано досвід по роботі зі схемами UML, зокрема побудова діаграми варіантів використання.

**Література**

1.Веселов А.І., Попівщий В.І. Методичні вказівки до виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування», Запоріжжя. ЗДІА, 2010.

2. Оформлення і захист курсових, дипломних та кваліфікаційних робіт Методичні вказівки для студ. ЗДІА спец. 080403 "ПЗАС" ден. та заоч. форм навчання / А. І. Безверхий, Н. П. Полякова, В. І. Попівщий, І. А. Скрипник ; ЗДІА. - Запоріжжя : ЗДІА, 2007.

3. Гайсарян С.С. Объектно-ориентированное программирование. М.: ЦИТ, 2002.

4.Проценко В.С., Чаленко П.Й., Ставровський А.Б. Техніка програмування мовою Сі. – К.: Либідь, 1993.

**Додаток**

**Додаток 1**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Media;

using System.Collections;

namespace refrigerator

{

enum RegimeOfFreeze { min = 1, middle, hight }//перечисленние для режимов морозильной каммеры

class TemperatureControl

{

double maxCelsius;//максимальный предел режима

double minCelsius;//минимальный предел режима

double currentTemperature = 7;//текущая температура в помещении

string freezeName;//имя морозильной камеры для события

public string FreezeName { get { return freezeName; } }//доступ к полю данных

public double MaxCelsius { get { return maxCelsius; } }//доступ к полю данных

public double MinCelsius { get { return minCelsius; } }//доступ к полю данных

public double CurrentTemperature { get { return currentTemperature; } set { currentTemperature = value; } }//доступ к полю данных

public TemperatureControl()//конструктор по умолчанию

{

maxCelsius = minCelsius = currentTemperature = 0;

}

public void CelsiusValueChange(RegimeOfFreeze f\_regime)

{

switch (f\_regime){

case RegimeOfFreeze.min://vegetables

minCelsius = 3;

maxCelsius = 6;

break;

case RegimeOfFreeze.middle://half-finished product

minCelsius = 2;

maxCelsius = 4;

break;

case RegimeOfFreeze.hight://fresh zone

minCelsius = 0;

maxCelsius = 1;

break;

default: break;}}

public TemperatureControl(RegimeOfFreeze f\_regime, string f\_name)//смена режима морозильной камеры

{

freezeName = f\_name;

CelsiusValueChange(f\_regime);

}

}

delegate void TempuratureEvent();//делегат для будущего события

class Engine : IMessageEvent

{

TemperatureControl temperature;//экземпляр Терморегулятора, поскольку мотор управляет температурой

public event RefregiratorEvents message;//событие основаное на сигнатуре делегата для журнала логов

bool statusOfTheEngine;//текущее состояние мотора true-работает false-отключён

bool crashStatus;//статус для указания поломки мотора

public TemperatureControl getTc { get { return temperature; } }//доступ к терморегулятору

public string GetFreezeName { get { return temperature.FreezeName; } }///доступ к имени полки

public double GetCurrentTemperature { get { return temperature.CurrentTemperature; } }///доступ к текущей температуре в камере

public bool StatusOfTheEngine { get { return statusOfTheEngine; } }//доступ к статусу мотора

public bool CrashStatus { get { return crashStatus; } }///доступ к статусу работоспособности мотора

public Engine()

{

temperature = null;

}

public Engine(TemperatureControl f\_value)

{

temperature = f\_value;

}

public void EngineOn()//включить мотор

{

statusOfTheEngine = true;//мотор работает

//записать в журнал логов

if (message != null) { message(this, new ListHandlerEventArgs(DateTime.Now, "Engine on", temperature.FreezeName)); }

//таймер для управления мотором

System.Windows.Threading.DispatcherTimer time = new System.Windows.Threading.DispatcherTimer();

time.Interval = new TimeSpan(0, 0, 1);

time.Tick += (Object myObject, EventArgs myEventArgs) =>

{

//неизвесное число для поломки мотора

Random s = new Random();

int num = s.Next(1, 10000);

if (num == 543)

{

/\*если число совпало

\* остановить мотор

\* изменить статус мотора

\*/

time.Stop();

statusOfTheEngine = false;

CrushMotor();//сломать мотор

}

temperature.CurrentTemperature -= 0.125;//понижаем температуру

//если понизили до минимального предела, остановить, следующее действие через булевскую переменную

if (temperature.CurrentTemperature <= temperature.MinCelsius) { statusOfTheEngine = false; time.Stop(); }

};

time.Start();

}

public void EngineOff()

{

statusOfTheEngine = true;//статус-мотор отключён

//событие для журнала логов

if (message != null) { message(this, new ListHandlerEventArgs(DateTime.Now, "Engine off", temperature.FreezeName)); }

//таймер для повышения температуры

System.Windows.Threading.DispatcherTimer time = new System.Windows.Threading.DispatcherTimer();

time.Interval = new TimeSpan(0, 0, 1);

time.Tick += (Object myObject, EventArgs myEventArgs) =>

{

temperature.CurrentTemperature += 0.125;

//если достигнут предел, освобождаем ресурс таймера ,следующее действие через булевскую переменную

if (temperature.CurrentTemperature >= temperature.MaxCelsius) { statusOfTheEngine = false; time.Stop(); }

};

time.Start();

}

public void CrushMotor()

{

//событие для журнала логов

if (message != null) { message(this, new ListHandlerEventArgs(DateTime.Now, "Engine crushed", temperature.FreezeName)); }

//показываем что мотор сломан

crashStatus = true;

//подаем звуковой сигнал

System.Media.SystemSounds.Hand.Play();

//на последок поднимаем температуру в камере

EngineOff();

}

}

interface IThermometer

{

double TakeTemperature { get; }//играет роль термометра

}

interface IMessageEvent

{

event RefregiratorEvents message;//событие для журнала логов

}

delegate void RefregiratorEvents(object source, ListHandlerEventArgs args);//делегат на основе которого строится событие

class Freezer : IThermometer, IMessageEvent

{

System.Windows.Threading.DispatcherTimer time;//таймер для дверцы холодильника

Engine mazda;//морозильная камера управляет мотором

public event RefregiratorEvents message;

public event TempuratureEvent show;//индикатор температуры

public Freezer()

{

time = new System.Windows.Threading.DispatcherTimer();

}

public Freezer(TemperatureControl f\_choice)

{

mazda = new Engine(f\_choice);

time = new System.Windows.Threading.DispatcherTimer();

}

public RegimeOfFreeze IdeaTemperature//меняем положение терморегулятора

{

set

{

if (message != null)//событие терморегулятора

{

message(this, new ListHandlerEventArgs(DateTime.Now, "Regime of freeze changed", mazda.GetFreezeName));

}

mazda.getTc.CelsiusValueChange(value);

}

}

public double TakeTemperature { get { return mazda.GetCurrentTemperature; } }//играет роль термометра

public string ShowTemperature()//значение термометра в строку

{

return TakeTemperature.ToString("0.00" + " °C");

}

public void MesuareTemperature()

{

//таймер для измерения температуры

System.Windows.Threading.DispatcherTimer time = new System.Windows.Threading.DispatcherTimer();

time.Interval = new TimeSpan(0, 0, 1);

time.Tick += (Object myObject, EventArgs myEventArgs) =>

{

if (show != null) show();//событие которое показывает температуру

if (!mazda.StatusOfTheEngine)//если мотор отключён

{

//если максимально допустимая температура включить мотор

if (mazda.GetCurrentTemperature >= mazda.getTc.MaxCelsius && !mazda.CrashStatus) mazda.EngineOn();

//отключить мотор

if (mazda.GetCurrentTemperature <= mazda.getTc.MinCelsius && !mazda.CrashStatus) mazda.EngineOff();

}

};

time.Start();

}

public void DoorOpen()

{

//дверца открыта

if (message != null) { message(this, new ListHandlerEventArgs(DateTime.Now, "Door open", mazda.GetFreezeName)); }

//Первоначальный интервал, после которого будет подаватся звуковой сигнал

time = new System.Windows.Threading.DispatcherTimer();

time.Interval = new TimeSpan(0, 0, 8);

time.Start();

time.Tick += (Object myObject, EventArgs myEventArgs) =>

{

//интервал прошёл, подать звуковой сигнал

time.Interval = new TimeSpan(0, 0, 1);

System.Media.SystemSounds.Hand.Play();

};

}

public void DoorClose()

{

//дверца закрыта

if (message != null) { message(this, new ListHandlerEventArgs(DateTime.Now, "Door close", mazda.GetFreezeName)); }

//остановить таймер для дверцы холодильника

time.Stop();

}

public Engine GetEngine { get { return mazda; } }//получить доступ к мотору

public TemperatureControl GetTC { get { return mazda.getTc; } }//получить доступ к терморугулятор

}

class Refrigerator

{

Freezer upperShelf;//создаем

Freezer middleShelf;//морозильные

Freezer lowerShelf;//камеры

public Refrigerator()

{

upperShelf = new Freezer();

middleShelf = new Freezer();

lowerShelf = new Freezer();

}

public Refrigerator(Freezer f\_first, Freezer f\_second, Freezer f\_third)

{

upperShelf = f\_first;

middleShelf = f\_second;

lowerShelf = f\_third;

}

public Freezer UpperShelf { get { return upperShelf; } }

public Freezer LowerShelf { get { return lowerShelf; } }

public Freezer MiddleShelf { get { return middleShelf; } }

public void ON()

{

upperShelf.MesuareTemperature();//включаем

middleShelf.MesuareTemperature();//термоментры

lowerShelf.MesuareTemperature();//камер, которые запустят мотор

}

}

class ListHandlerEventArgs : System.EventArgs

{

//добавляем поля к событию в виде авто.свойств

public string FreezeName { get; set; }

public DateTime Time { get; set; }

public string Changes { get; set; }

public ListHandlerEventArgs()

{

FreezeName = Changes = "";

Time = DateTime.MinValue;

}

public ListHandlerEventArgs(DateTime f\_time, string f\_changes, string f\_name)

{

FreezeName = f\_name;

Time = f\_time;

Changes = f\_changes;

}

}

class JournalEntry

{

//класс который будет хранить события

public string FreezeName { get; set; }

public DateTime Time { get; set; }

string EventName { get; set; }

public JournalEntry(DateTime f\_time, string f\_event, string f\_name)

{

FreezeName = f\_name;

Time = f\_time;

EventName = f\_event;

}

string Information()

{

string info = "Name->" + FreezeName + ";Time>" + Time + ";Changes->" + EventName + "\n";

return info;

}

public override string ToString()

{

return Information();

}

}

class TeamsJournal

{

//класс подписчик

List<JournalEntry> events = new List<JournalEntry>();//список событий

public void NewEvents(object sender, ListHandlerEventArgs f\_info)//метод котоый реагирует на события

{

//событие произошло, формируем узел списка

JournalEntry teamEvent = new JournalEntry(f\_info.Time, f\_info.Changes, f\_info.FreezeName);

//добавляем узел

events.Add(teamEvent);

}

public IEnumerator GetEnumerator()//итератор для перебора событий

{

foreach (var item in events)

{

yield return item;

}

}

}

}

**Класс JournalForm**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace WpfRefrigerator

{

/// <summary>

/// Interaction logic for Journal.xaml

/// </summary>

public partial class Journal : Window

{

public Journal()

{

InitializeComponent();

}

private void Read\_File()

{

MainWindow s = this.Owner as MainWindow;

foreach (var item in s.RfEvents)//через итератов класса получаем информацию об событиях

{

JournalInfo.Text += item.ToString() + '\n';

}

}

private void Load(object sender, RoutedEventArgs e)

{

Read\_File();

}

}

**}**

**Класс MainWindow**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.IO;

using refrigerator;

namespace WpfRefrigerator

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

string regime = "";//режим по умолчанию

Refrigerator rf;//создаеи холодильник

internal Refrigerator Rf { get { return rf; } set { rf = value; } }//доступ к холодильнику

TeamsJournal rfEvents;//класс подписчик

internal TeamsJournal RfEvents { get { return rfEvents; } }//доступ к классу подписчику

private void TemperatureEvent(Freezer f\_freezer, TextBox f\_tbox)

{

f\_freezer.show += () => f\_tbox.Text = f\_freezer.ShowTemperature();//подписуемся на событие которое показувает температуру

}

private void MessageEvent(Freezer f\_freezer, TeamsJournal f\_journal)

{

//подписуемся на события холодильника

f\_freezer.message += f\_journal.NewEvents;

f\_freezer.GetEngine.message += f\_journal.NewEvents;

}

private void Load(object sender, RoutedEventArgs e)

{

//иннициализация холодильника и его событий

rfEvents = new TeamsJournal();

Freezer firstCam = new Freezer(new TemperatureControl(RegimeOfFreeze.min, "First Freezer"));

Freezer secondCam = new Freezer(new TemperatureControl(RegimeOfFreeze.min, "SecondFreezer"));

Freezer thirdCam = new Freezer(new TemperatureControl(RegimeOfFreeze.min, "ThirdFreezer"));

TemperatureEvent(firstCam, firstFreezeCelsius);

TemperatureEvent(secondCam, secondFreezeCelsius);

TemperatureEvent(thirdCam, thirdFreezeCelsius);

MessageEvent(firstCam, rfEvents);

MessageEvent(secondCam, rfEvents);

MessageEvent(thirdCam, rfEvents);

Rf = new Refrigerator(firstCam, secondCam, thirdCam);

Rf.ON();

}

private void ControlDoor(Freezer f\_choice, Button f\_current)

{

LinearGradientBrush gradientBrush = new LinearGradientBrush(Color.FromRgb(209, 227, 250), Color.FromRgb(170, 199, 238), new Point(0.5, 0), new Point(0.5, 1));

Background = gradientBrush;

string ont = f\_current.Content.ToString();

if (ont == "close")

{

f\_current.Content = "open";

f\_current.Background = Brushes.PaleGreen;

f\_choice.DoorOpen();

Info.Text = f\_choice.GetTC.FreezeName + " door open";

}

else

{

f\_current.Content = "close";

f\_current.Background = Brushes.Tomato;

f\_choice.DoorClose();

Info.Text = f\_choice.GetTC.FreezeName + " door close";

}

}

private void firstDoor\_Click(object sender, RoutedEventArgs e)

{

ControlDoor(Rf.UpperShelf, firstDoor);

}

private void secondDoor\_Click(object sender, RoutedEventArgs e)

{

ControlDoor(Rf.MiddleShelf, secondDoor);

}

private void thirdDoor\_Click(object sender, RoutedEventArgs e)

{

ControlDoor(Rf.LowerShelf, thirdDoor);

}

private void ChangeRegimeOfFreeze(Freezer f\_freezerCam)

{

double saveTemp = f\_freezerCam.TakeTemperature;

switch (regime)

{

case "Minimum":

f\_freezerCam.IdeaTemperature = RegimeOfFreeze.min;

break;

case "Middle":

f\_freezerCam.IdeaTemperature = RegimeOfFreeze.middle;

break;

case "Maximum":

f\_freezerCam.IdeaTemperature = RegimeOfFreeze.hight;

break;

}

f\_freezerCam.GetTC.CurrentTemperature = saveTemp;

Info.Text = f\_freezerCam.GetTC.FreezeName + " thermostat positions changed";

}

private void ChangeCam()

{

int choice = Convert.ToInt32(FreezerBar.Value);

switch (choice)

{

case 1:

ChangeRegimeOfFreeze(Rf.UpperShelf);

break;

case 2:

ChangeRegimeOfFreeze(Rf.MiddleShelf);

break;

case 3:

ChangeRegimeOfFreeze(Rf.LowerShelf);

break;

}

}

private void RegimeChanged(object sender, RoutedEventArgs e)

{

//новый режим морозильной каммеры

RadioButton userChoice = (RadioButton)sender;

regime = userChoice.Content.ToString();

ChangeCam();

}

private void ViewJournal(object sender, RoutedEventArgs e)

{

//просмотр событий в новом окне

Journal currentEvents = new Journal();

currentEvents.Owner = this;

currentEvents.ShowDialog(); }

private void MenuItem\_Click(object sender, RoutedEventArgs e)

{

//системное окно сохраненния файла, формата txt

Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();

dlg.FileName = "Document"; // Default file name

dlg.DefaultExt = ".text"; // Default file extension

dlg.Filter = "Text documents (.txt)|\*.txt"; // Filter files by extension

// Show save file dialog box

Nullable<bool> result = dlg.ShowDialog();

// Process save file dialog box results

if (result == true)

{

// Save document

string filename = dlg.FileName;

using (var news = new StreamWriter(filename, false))

{

foreach (var item in rfEvents)

{

news.WriteLine(item);

}

}

}

}

}

}