Министерство науки и высшего образования Российской Федерации   
Федеральное государственное бюджетное образовательное учреждение высшего образования «Петрозаводский государственный университет»  
 Физико-технический институт   
Кафедра информационно-измерительных систем и физической электроники

**Отчет по лабораторной работе**

«КОМАНДНОЕ ЗАДАНИЕ ПО КУРСУ ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ»

Преподаватель:

канд. физ.-мат. наук, доцент

Бульба Артём Владимирович

«\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022 г

Петрозаводск 2022

Над проектом работали студенты 21316  
1. Урванцев Кирилл Дмитриевич - Team leader  
2. Бочарников Семен Александрович   
3. Лумпов Иван Алексеевич

Оглавление

[Цель работы 4](#_Toc123160415)

[Кратко о программной реализации 4](#_Toc123160416)

[1.Краткое словесное описание сюжета 4](#_Toc123160417)

[2.Материалы заказчика 5](#_Toc123160418)

[3. Действующие субъекты: 5](#_Toc123160419)

[4. Список вариантов использования 5](#_Toc123160420)

[5. Описание каждого вариант использования 6](#_Toc123160421)

[6. Диаграммы вариантов использования 6](#_Toc123160422)

[7.Список предварительный список существительных 7](#_Toc123160423)

[8.Уточненный список классов 7](#_Toc123160424)

[9.Список атрибутов классов 8](#_Toc123160425)

[10.Сообщения 8](#_Toc123160426)

[11. Диаграмма классов 9](#_Toc123160427)

[12.Диаграммы последовательностей 9](#_Toc123160428)

[13.Код заголовочных файлов 11](#_Toc123160429)

[14.Код исходных файлов 13](#_Toc123160430)

[15. Руководство пользователя 26](#_Toc123160431)

[Ссылка на репозиторий GitHub 27](#_Toc123160432)

[Графическая интерпретация истории проекта на GitHub с графического подсмотрщик истории “gitk” 27](#_Toc123160433)

[Мой вклад в проект 27](#_Toc123160434)

[Заключение 35](#_Toc123160435)

# Цель работы

Смоделировать ситуацию выполнения заказа на разработку программного обеспечения пекарни

# Кратко о программной реализации

Среда разработки Qt Creator (Qt 5.4.2)

Язык программирования C++

Созданные заголовочные и единицы компиляции:

bekker.h – заголовочный файл, содержащий описания классов годового отчёта, затрат и главного меню

products.h – заголовочный файл, содержащий описания классов продуктов, заказов и доходов

maininterface.cpp – реализация главного меню

prodconstruct.cpp – реализация конструкторов продуктов и заказов

main.cpp – основной файл запускающий главное меню

eexpenserecord.cpp – содержит реализацию конструктора затрат, ввода затрат и вывода годового отчёта

addproduct.cpp – содержит реализацию добавления продуктов, вывода их списка, добавления заказов, пересчёта годовой выручки и конструктор доходов

# 1.Краткое словесное описание сюжета

В городе, близком к границе с Россией, открылась новая пекарня. Для нормального функционирования ей требуется программное обеспечение, реализующее учет и продажи. У заказчика была таблица с данными, включающая в себя цены, наименования товара и вес. Но так как это молодой бизнес у предпринимателя не было много средств, так готовое программное обеспечение было дороже, чем наш программный продукт. Так же у него не было навыков для создания такой программы. Проблема, порученная нашему коллективу заказчиком, было написание эффективной программы для оформления заказов в местной пекарне. В ожиданиях заказчика было, что программа будет проста в использовании и нетребовательна в компьютерных навыках для оператора. Так же было необходимо, чтобы управление программой мог вести как продавец, оформляющий заказы, так и управляющий персонал, например бухгалтер или менеджер. Исходя из поставленного технического задания и рекомендаций, которые мы получили от заказчика, была сформирована консольная программа на языке программирования C++ в среде разработки IDE Qt Creator (Qt 5.4.2), простая в освоении и использовании. Поэтапно были разработаны каждая из частей требуемой программы: меню кассира и меню управляющего персонала. Для кассира меню позволяет сформировать заказ из имеющихся в списке товаров, поделенных на категории. В свою очередь управляющий персонал может посмотреть годовую отчетность, представленную в таблице, а также список заказов за все время с указанием товара и общей стоимости заказа. С помощью консольного вывода программы был реализован графический интерфейс с возможностью выбора нужной подпрограммы. В результате была сформирована программа, позволяющая анализировать финансовые параметры, такие как прибыль, доход и расход, а также оформлять заказы.

# 2.Материалы заказчика

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Вес, гр | Себестоимость продукта, руб | Цена на прилавке, руб |
| Торты | | | |
| Honey cake | 350 | 220 | 300 |
| Napoleon | 300 | 220 | 350 |
| Cherry cake | 450 | 300 | 400 |
| Хлеб | | | |
| White | 200 | 21 | 38 |
| Wheat | 150 | 74 | 120 |
| Marble | 210 | 50 | 100 |
| Кондитерские изделия | | | |
| Hot dog | 100 | 50 | 100 |

Таблица 1 – Материалы заказчика

# 3. Действующие субъекты:

1.Кассир   
2. Менеджер  
  
Для работы с программой предполагается два разных человека. Один работает кассиром, второй является представителем управляющего персонала (менеджер, директор, бухгалтер). В обязанности кассира входит оформление заказа. Управляющий персонал может оформлять таблицу расходов за год и добавлять новые позиции в список товаров и смотреть годовую отчетность.

# 4. Список вариантов использования



Рисунок 1 – Диаграмма вариантов использования

# 5. Описание каждого вариант использования

Приходит клиент и заказывает 2 хлеба. Кассир запускает программу, выбирает меню «Cashier». В появившемся меню кассир выбирает функцию «Add order». Из категорий выпечки выбирается «Bread». Затем выбирается необходимый хлеб, например белый, введя цифру, обозначающую этот товар. После этого, кассир вводит необходимое количество продукта и вводит букву d в программе для завершения заказа.

Бухгалтер хочет посмотреть годовую отчетность. Для этого он запускает программу и выбирает меню «Manager». В появившемся меню он выбирает «Annual report», чтобы посмотреть годовую отчетность. Необходимо ввести год, за который необходимо посмотреть отчетность. В консоль выводится таблица с доходом, расходом и прибылью по месяцам.

Бухгалтер хочет ввести дополнительные расходы. Для этого он запускает программу и выбирает меню «Manager». Необходимо ввести год, затем ввести все расходы с января с пробелами между ними. После этого все введенные расходы будут отражены в годовой отчетности.

Менеджер хочет добавить торт «Антон». Для этого он запускает программу и выбирает меню «Manager». Далее выбирается необходимая категория продукта, в данном случае Cake. Вводится имя торта, его вес, себестоимость и его цена. Для выхода из меню создания необходимо ввести букву e.

Менеджер хочет ознакомиться со списком всей продукции. Для этого он запускает программу и выбирает меню «Manager». В появившемся меню он выбирает «Product list», чтобы посмотреть все продукты. В консоль выводится весь список товаров, разбитых на категории.

# 6. Диаграммы вариантов использования

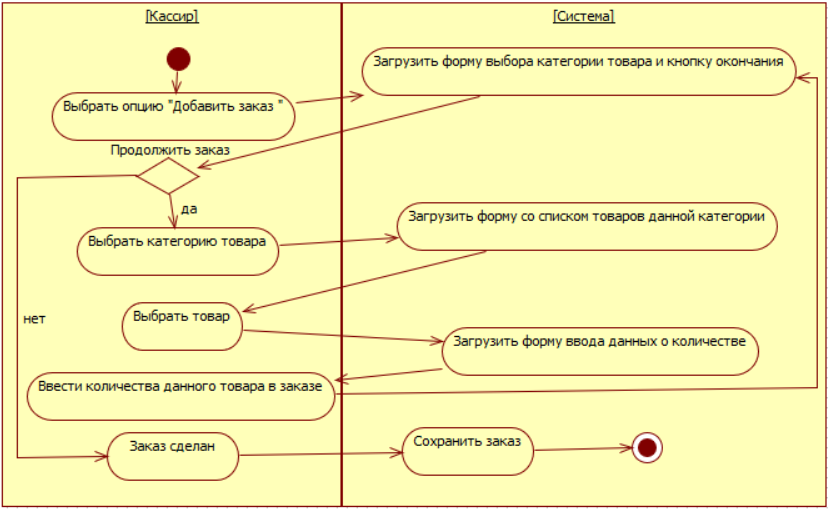


Рисунок 2.1 – Диаграмма действия “Добавить заказ”

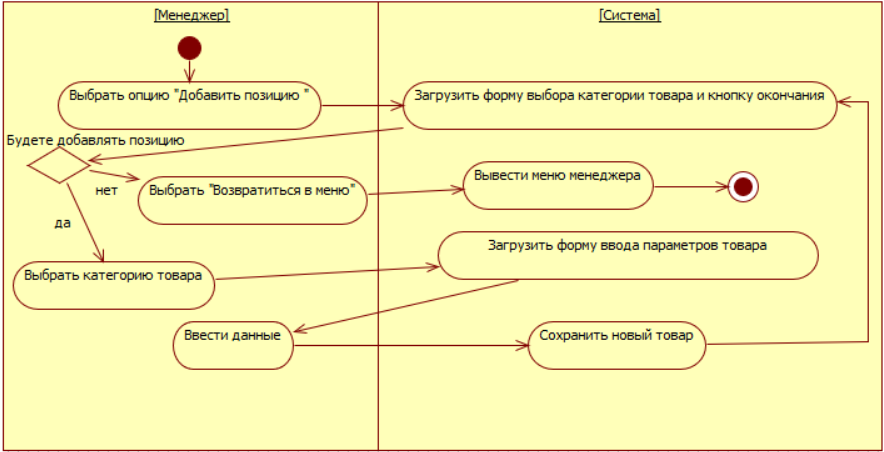


Рисунок 2.2 - Диаграмма действия “Добавить позицию”

# 7.Список предварительный список существительных

|  |  |  |  |
| --- | --- | --- | --- |
| клиент | товар | меню «Manager» | торт |
| кассир | количество | консоль | имя |
| хлеб | продукт | таблица | вес |
| программа | буква | доход | себестоимость |
| меню «Cashier» | заказ | расход | цена |
| функция | бухгалтер | прибыль | выход |
| категория | отчетность | месяц | создание |
| выпечка | год | январь | список |
| цифра |  | пробел | продукция |

# 8.Уточненный список классов

|  |  |
| --- | --- |
| Кассир | Годовая отчетность |
| Программа | Меню «Manager» |
| Меню «Cashier» | Консоль |
| Категория | Таблица |
| Товар | Менеджер |
| Количество | Список |
| Бухгалтер | Продукт |

# 9.Список атрибутов классов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Базовый класс **Product** обладает следующими атрибутами: | |  | | |  |  |
| 1. Name - имя |  | |  |  |  |  |
| 2. Weight -вес |  | |  |  |  |  |
| 3. CostPrice - себестоимость | | |  |  |  |  |
| 4. Price – рыночная стоимость | | |  |  |  |  |
| Производные классы Cake, Pastry и Bread от базового класса Product наследуют его атрибуты | | | | | |  |
|  |  | |  |  |  |  |
| Производный класс **Order** от базового класса Product наследует его атрибуты и  имеет собственные атрибуты: | | | | | |  |
|  | 1. Year - год | |  |  |  |  |
|  | 2. Month - месяц | |  |  |  |  |
|  | 3. Sum - сумма | |  |  |  |  |
|  | 4. Profit – прибыль | |  |  |  |  |
| Класс **ProfitRecord** обладает следующими атрибутами: | | Класс **AnnualReport**  обладает следующими атрибутами: | | | | |
|  | 1. Year - год | |  | 1. Year - год |  |  |
|  | 2. Jan - январь | |  | 2. Jan - январь |  |  |
|  | 3. Feb - февраль | |  | 3. Feb - февраль |  |  |
|  | 4. Mar - март | |  | 4. Mar - март |  |  |
|  | 5. Apr - апрель | |  | 5. Apr - апрель |  |  |
|  | 6. May - май | |  | 6. May - май |  |  |
|  | 7. Jun - июнь | |  | 7. Jun - июнь |  |  |
|  | 8. Jul - июль | |  | 8. Jul - июль |  |  |
|  | 9. Aug - август | |  | 9. Aug - август |  |  |
|  | 10. Sep - сентябрь | |  | 10. Sep - сентябрь |  |  |
|  | 11. Oct - октябрь | |  | 11. Oct - октябрь |  |  |
|  | 12. Nov - ноябрь | |  | 12. Nov - ноябрь |  |  |
|  | 13. Dec – декабрь | |  | 13. Dec – декабрь |  |  |

# 10.Сообщения

Оформи заказ – Создание заказа

Выведи годовой отчёт – Расчет прибыли, расходов, доходов

Заполни расходы – Заполнение таблицы расходов

Добавь позицию – Создание новых товаров в списке

Выведи список позиций – Вывод списка доступных товаров

# 11. Диаграмма классов

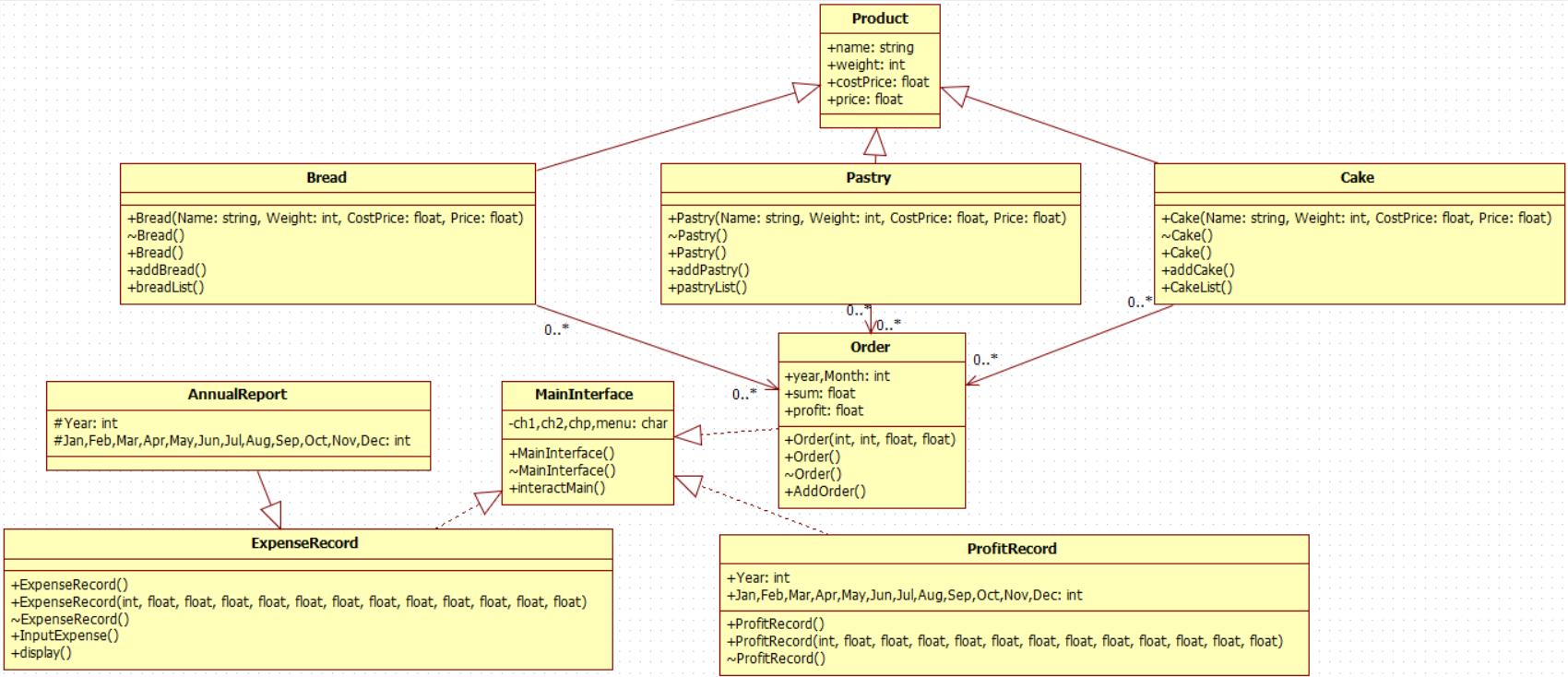


Рисунок 3 – Диаграмма классов проекта TeamProjectBekker

# 12.Диаграммы последовательностей

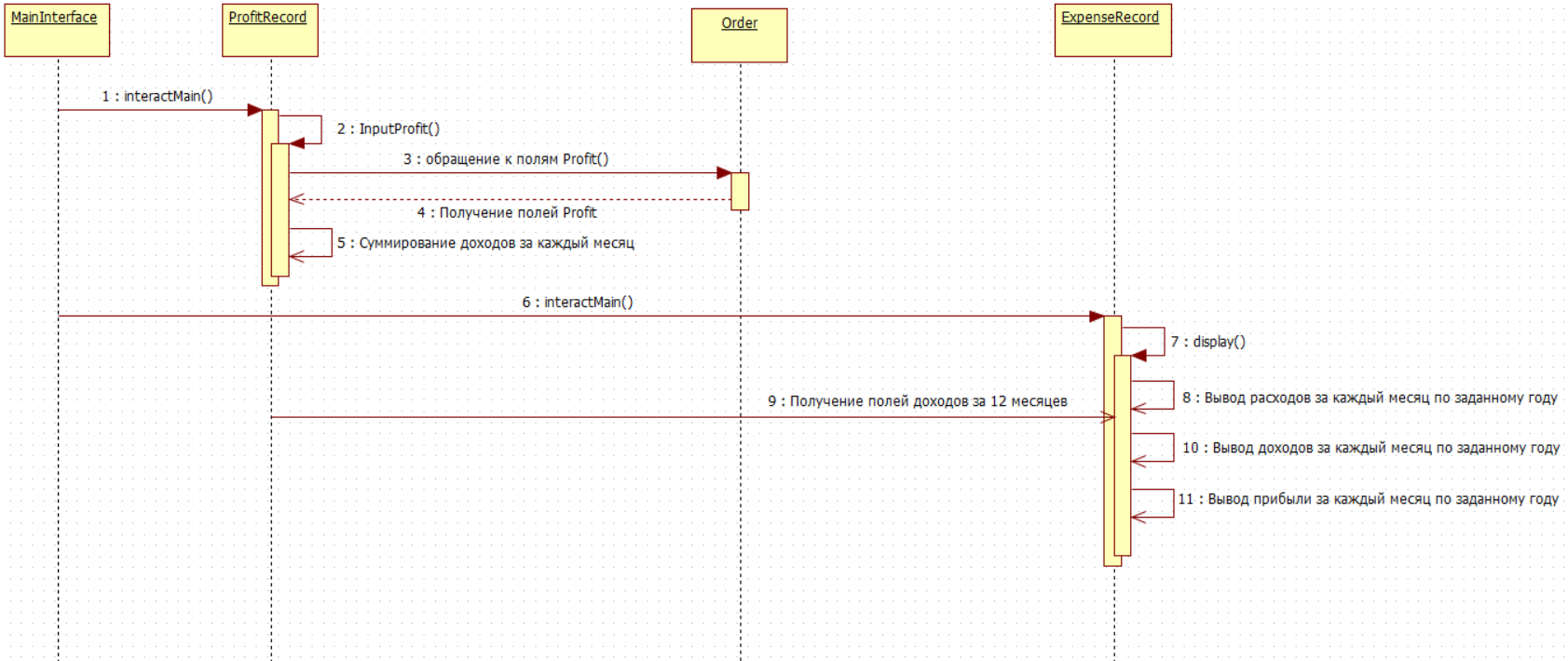


Рисунок 4.1 – Диаграмма последовательности “Просмотр годового отчета”

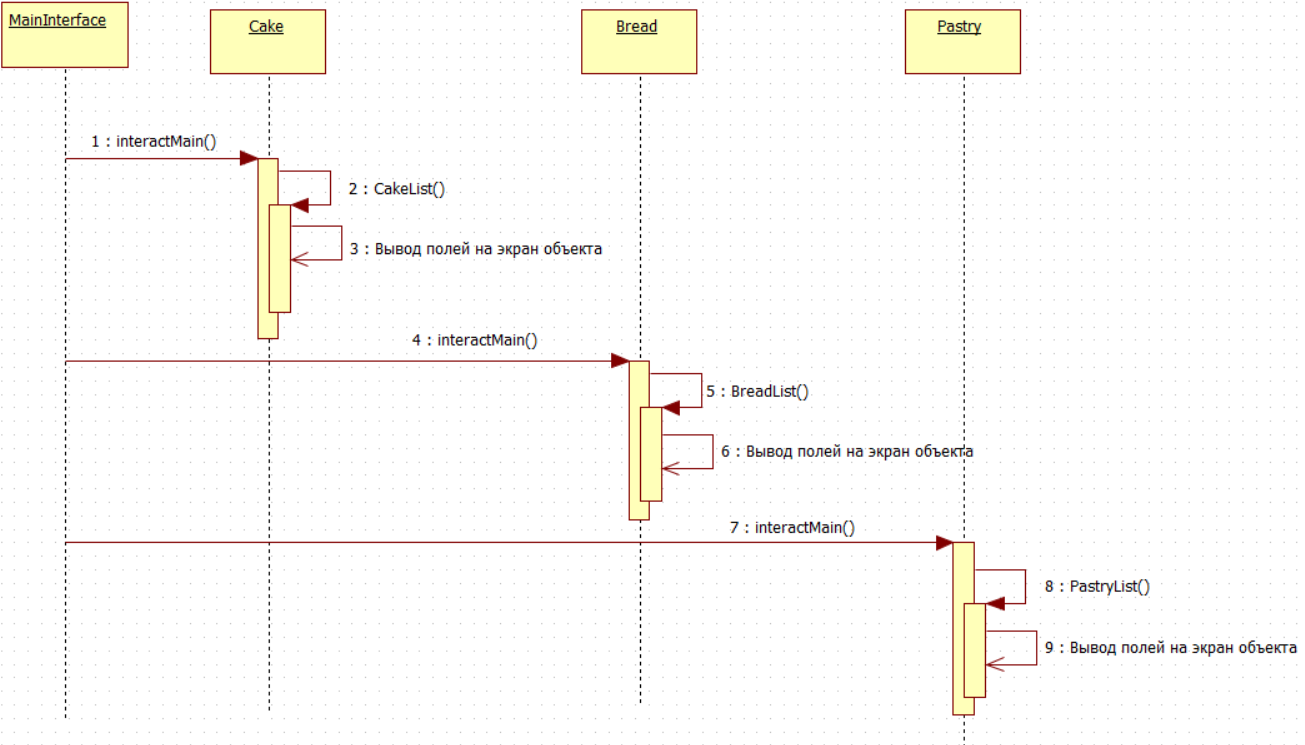


Рисунок 4.2 – Диаграмма последовательности “Просмотр списка позиций”

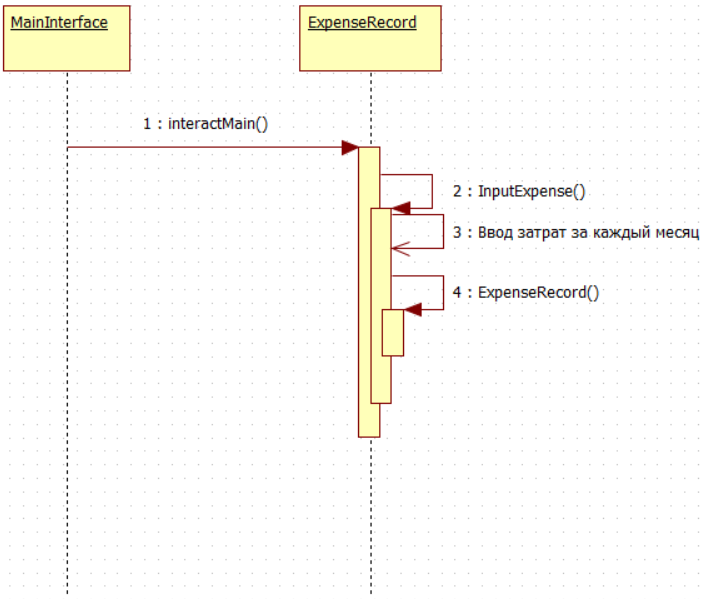


Рисунок 4.3 – Диаграмма последовательности “Заполнение расходов”

# 13.Код заголовочных файлов

#ifndef BEKKER\_H

#define BEKKER\_H

#include <iostream>

#include <string>

using namespace std;

class AnnualReport

{

protected:

int Year;

int Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec;

};

class ExpenseRecord : public AnnualReport//месяц должен быть

{

public:

ExpenseRecord(){}

ExpenseRecord(int ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float );

~ExpenseRecord(){}

void InputExpense();

void display();

};

class MainInterface

{

private: //Поле

char ch,ch2,chp,menu;

public: //Методы

MainInterface(){}

~MainInterface() {}

void interactMain();

};

#endif // BEKKER\_H

Products.h

#ifndef PRODUCTS

#define PRODUCTS

#include <iomanip> //для setw()

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

using namespace std;

void ProdInit(); //начальный список продуктов

void ListClear(); //Очищение всех листов продуктов Cakelist,Breadlist,PastryList

void ClearExpense(); //Очищение вектора расходов v

//Базовый класс Product

class Product

{

public:

string Name;

int Weight; // вес

float CostPrice; //себестоимость

float Price; //рыночная стоимость(на полке на магазина)

};

//производный класс Cake от базового Product

class Cake : public Product

{

public:

Cake(){}

Cake(string ,int ,float ,float );

~Cake(){}//broken da boken

void addCake();

void CakeList();

};

//производный класс Pastry от базового Product

class Pastry : public Product

{

public:

Pastry(){}

Pastry(string ,int ,float ,float );

~Pastry(){}

void addPastry();

void PastryList();

};

//производный класс Bread от базового Product

class Bread : public Product

{

public:

Bread(){}

Bread(string ,int ,float ,float );

~Bread(){}

void addBread();

void BreadList();

};

//производный класс Order от базового Product(чтобы иметь доступ к полям класса Product)

//Добавляет возможность заказа из списка продуктов

class Order : public Product

{

public:

int Year,Month;

float Sum;

float Profit;

Order(int Year,int Month,float Sum ,float Profit);

Order(){}

~Order(){}

void AddOrder();

};

//Класс ProfitRecord для расчетов доходов с заказов Order за каждый месяц

class ProfitRecord

{

public:

int Year;

int Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec;

ProfitRecord(){}

ProfitRecord(int ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float );

~ProfitRecord(){}

void InputProfit();

};

extern vector <ProfitRecord> vProf;

#endif // PRODUCTS

# 14.Код исходных файлов

addproduct.cpp

#include "products.h"

vector <Cake> vCake;

vector <Pastry> vPastry;

vector <Bread> vBread;

vector <Order> vOrd;

vector <ProfitRecord> vProf;

void Cake::addCake() //добавление новых позиций

{

string \*CName = new string;

int \*CWeight = new int;

float \*CCostPrice = new float;

float \*CPrice = new float;

cout<<"Enter Cake name:\n";

cin>>\*CName;

cout<<"\nEnter Cake weight:\n";

cin>>\*CWeight;

cout<<"\nEnter Cake cost price:\n";

cin>>\*CCostPrice;

cout<<"\nEnter Cake price:\n";

cin>>\*CPrice;

vCake.push\_back(Cake(\*CName,\*CWeight,\*CCostPrice,\*CPrice));

delete CName;

delete CWeight;

delete CCostPrice;

delete CPrice;

}

void Bread::addBread()

{

string \*CName = new string;

int \*CWeight = new int;

float \*CCostPrice = new float;

float \*CPrice = new float;

cout<<"Enter Bread name:\n";

cin>>\*CName;

cout<<"\nEnter Bread weight:\n";

cin>>\*CWeight;

cout<<"\nEnter Bread cost price:\n";

cin>>\*CCostPrice;

cout<<"\nEnter Bread price:\n";

cin>>\*CPrice;

vBread.push\_back(Bread(\*CName,\*CWeight,\*CCostPrice,\*CPrice));

delete CName;

delete CWeight;

delete CCostPrice;

delete CPrice;

}

void Pastry::addPastry()

{

string \*CName = new string;

int \*CWeight = new int;

float \*CCostPrice = new float;

float \*CPrice = new float;

cout<<"Enter Pastry name:\n";

cin>>\*CName;

cout<<"\nEnter Pastry weight:\n";

cin>>\*CWeight;

cout<<"\nEnter Pastry cost price:\n";

cin>>\*CCostPrice;

cout<<"\nEnter Pastry price:\n";

cin>>\*CPrice;

vPastry.push\_back(Pastry(\*CName,\*CWeight,\*CCostPrice,\*CPrice));

delete CName;

delete CWeight;

delete CCostPrice;

delete CPrice;

}

void ProdInit() //создания базового списка продуктов

{

string \*CName = new string;

//Торты

\*CName = "Honey cake";

vCake.push\_back(Cake(\*CName,350,200,300)); //название/вес/себестоимость/ЦенаНаПолке

\*CName = "Napoleon";

vCake.push\_back(Cake(\*CName,300,220,350));

\*CName = "Cherry Cake";

vCake.push\_back(Cake(\*CName,450,300,400));

//Хлеб

\*CName = "White";

vBread.push\_back(Bread(\*CName,200,21,38));

\*CName = "Wheat";

vBread.push\_back(Bread(\*CName,150,74,120));

\*CName = "Marble";

vBread.push\_back(Bread(\*CName,210,50,100));

//Выпечка

\*CName = "Hot dog";

vPastry.push\_back(Pastry(\*CName,100,50,100));

\*CName = "Donut";

vPastry.push\_back(Pastry(\*CName,100,50,100));

\*CName = "Croissant";

vPastry.push\_back(Pastry(\*CName,100,50,100));

\*CName = "Swiss roll";

vPastry.push\_back(Pastry(\*CName,100,50,100));

\*CName = "Begel";

vPastry.push\_back(Pastry(\*CName,100,50,100));

delete CName;

}

void Cake::CakeList()

{

cout << setiosflags(ios::left);//выравнивание по левой части

cout<< setw(15) <<"-----------------Cakes-----------------"<<endl;

// setw() устанавливает ширину следующего ввода-вывода равной переданному интегральному аргументу

for (unsigned int j = 0; j<vCake.size(); j++)

{

cout<< setw(3) << j << setw(15) << vCake[j].Name << setw(15) << vCake[j].Weight << setw(5) << vCake[j].Price <<endl;

}

}

void Bread::BreadList()

{

cout << setiosflags(ios::left);//выравнивание по левой части

cout<< setw(15) <<"\n-----------------Bread-----------------"<<endl;

for (unsigned int j = 0; j<vBread.size(); j++)

{

cout<< setw(3) << j << setw(15) << vBread[j].Name << setw(15) << vBread[j].Weight << setw(5) << vBread[j].Price <<endl;

}

}

void Pastry::PastryList()

{

cout << setiosflags(ios::left);//выравнивание по левой части

cout<< setw(15) <<"\n-----------------Pastry-----------------"<<endl;

for (unsigned int j = 0; j<vPastry.size(); j++)

{

cout << setw(3) << j << setw(15) << vPastry[j].Name << setw(15) << vPastry[j].Weight << setw(5) << vPastry[j].Price <<endl;

}

}

void Order::AddOrder() //метод добавления заказа класса "заказ"

{

int Year,Month,no,match,kol; //параметры заказа год/месяц/номер/флаг match/количество

float Sum = 0; //обнуляем сумму

float Profit = 0; // обнуляем прибыль

char memnu;

do{

system("cls");

cout<<" Cake 'c'\n"

<<" Bread 'b'\n"

<<" Pastry 'p'\n"

<<" Done 'd'\n";

cin>>memnu;

switch(memnu)

{

case 'c': //добавление в зазаз Cake

{

system("cls");

Cake \*Tempc = new Cake;

Tempc->CakeList();

delete Tempc;

cin>>no;

cout<<"\nQuantity: ";

cin>>kol;

Sum += vCake[no].Price \* kol; //считаем сумму позиции Cake сумма=цена товара\*количество

Profit += (vCake[no].Price-vCake[no].CostPrice) \* kol;//Прибыль=(ЦенаНаПолке-Себестоимость)\*количество

}

break;

case 'b': //добавление в заказ Bread

{

system("cls");

Bread \*Tempb = new Bread;

Tempb->BreadList();

delete Tempb;

cin>>no;

cout<<"\nQuantity: ";

cin>>kol;

Sum += vBread[no].Price \* kol; //считаем сумму позиции Bread сумма=цена товара\*количество

Profit += (vBread[no].Price-vBread[no].CostPrice) \* kol; //Прибыль=(ЦенаНаПолке-Себестоимость)\*количество

}

break;

case 'p': //добавление в заказ Pastry

{

system("cls");

Pastry \*Tempp = new Pastry;

Tempp->PastryList();

delete Tempp;

cin>>no;

cout<<"\nQuantity: ";

cin>>kol;

Sum += vPastry[no].Price \* kol;//считаем сумму позиции Pastry сумма=цена товара\*количество

Profit += (vPastry[no].Price-vPastry[no].CostPrice) \* kol;//Прибыль=(ЦенаНаПолке-Себестоимость)\*количество

}

break;

case 'd':

{

time\_t now = time(0); //начинает счёт времени с 0 для правильного отражения года

tm \*ltm = localtime(&now);//время вроде в секундах

Year = 1900 + ltm->tm\_year; //tm\_year выдаёт год с 1900 те 122 поэтому и +1900

Month = ltm->tm\_mon+1;

}

break;

}//конец switch(memnu)

}while(memnu != 'd');

match = 0;

for (unsigned int j = 0; j<vOrd.size(); j++) //проходим по вектору заказов

{

if (vOrd[j].Year == Year) //ищем год в векторе заказов с тем что в системе ПК

{

if (vOrd[j].Month == Month) //ищем месяц в векторе заказов с тем что в системе ПК

{

vOrd[j].Sum += Sum;

vOrd[j].Profit += Profit;

match = 1;

}

}

}

if (match == 0)

{

vOrd.push\_back(Order(Year,Month,Sum,Profit));//записываем в вектор заказов заказ,вызывая конструктор с параметрами заказа

}

system("cls"); // стираем консоль

cout<<"That will be "<<Sum<<endl; // сумма=цена товара\*количество

//cout<<Profit<<endl; сумма прибыли=(ЦенаНаПолке-Себестоимость)\*количество

system("pause");

}

ProfitRecord::ProfitRecord(int Year,float Jan,float Feb,float Mar,float Apr,float May,float Jun,float Jul,float Aug,float Sep,float Oct,float Nov,float Dec){

this->Year = Year; //по году будет происходить поиск нужного объекта

this->Jan = Jan;// расходы за Январь

this->Feb = Feb;// расходы за Февраль

this->Mar = Mar;// расходы за Март

this->Apr = Apr;// расходы за Апрель

this->May = May;// расходы за Май

this->Jun = Jun;// расходы за Июнь

this->Jul = Jul;// расходы за Июнь

this->Aug = Aug;// расходы за Август

this->Sep = Sep;// расходы за Сентабрь

this->Oct = Oct;// расходы за Октябрь

this->Nov = Nov;// расходы за Ноябрь

this->Dec = Dec;// расходы за Декабрь

}

void ProfitRecord::InputProfit()

{

int match;

vProf.clear();

for (unsigned int j = 0; j<vOrd.size() ; j++ ) //проходим по вектору указателей на обьекты заказов(сколько заказов столько и выполнится for)

{

match = 0;

for(unsigned int i = 0; i<vProf.size() ; i++ ) //проходим по вектору указателей на обьекты прибыли(сколько записей прибыли столько и выполнится)

{

if(vProf[i].Year == vOrd[j].Year) //ищем в векторе прибыли обьект с таким же годом,как в векторе заказов

{

match = 1; // если существует вектора прибыли с таким годом,то записываем в него данные

switch(vOrd[j].Month)

{

case 1://суммируем в конкретный месяц вектора прибыли <- поле прибыли с вектора заказов

vProf[i].Jan += vOrd[j].Profit;

break;

case 2:

vProf[i].Feb += vOrd[j].Profit;

break;

case 3:

vProf[i].Mar += vOrd[j].Profit;

break;

case 4:

vProf[i].Apr += vOrd[j].Profit;

break;

case 5:

vProf[i].May += vOrd[j].Profit;

break;

case 6:

vProf[i].Jun += vOrd[j].Profit;

break;

case 7:

vProf[i].Jul += vOrd[j].Profit;

break;

case 8:

vProf[i].Aug += vOrd[j].Profit;

break;

case 9:

vProf[i].Sep += vOrd[j].Profit;

break;

case 10:

vProf[i].Oct += vOrd[j].Profit;

break;

case 11:

vProf[i].Nov += vOrd[j].Profit;

break;

case 12:

vProf[i].Dec += vOrd[j].Profit;

break;

}//конец switch(vOrd.Month)

}

}

if(match == 0) //если не существует вектора прибыли в данном году,введенный пользователем,то создаем его и записываем в конкретный месяц прибыль

{

switch(vOrd[j].Month)

{

case 1:

vProf.push\_back(ProfitRecord(vOrd[j].Year,vOrd[j].Profit,0,0,0,0,0,0,0,0,0,0,0));

break;

case 2:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,vOrd[j].Profit,0,0,0,0,0,0,0,0,0,0));

break;

case 3:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,vOrd[j].Profit,0,0,0,0,0,0,0,0,0));

break;

case 4:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,vOrd[j].Profit,0,0,0,0,0,0,0,0));

break;

case 5:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,vOrd[j].Profit,0,0,0,0,0,0,0));

break;

case 6:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,vOrd[j].Profit,0,0,0,0,0,0));

break;

case 7:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,vOrd[j].Profit,0,0,0,0,0));

break;

case 8:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,vOrd[j].Profit,0,0,0,0));

break;

case 9:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,vOrd[j].Profit,0,0,0));

break;

case 10:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,0,vOrd[j].Profit,0,0));

break;

case 11:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,0,0,vOrd[j].Profit,0));

break;

case 12:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,0,0,0,vOrd[j].Profit));

break;

}//конец switch(vOrd.Month)

}

}

}

void ListClear()

{

vCake.clear();

vPastry.clear();

vBread.clear();

vOrd.clear();

vProf.clear();

}

eexpenserecord.cpp

#include "bekker.h"

#include "products.h"

vector <ExpenseRecord> v;

ExpenseRecord::ExpenseRecord(int Year,float Jan,float Feb,float Mar,float Apr,float May,float Jun,float Jul,float Aug,float Sep,float Oct,float Nov,float Dec){

this->Year = Year; //по году будет происходить поиск нужного объекта

this->Jan = Jan;// расходы за Январь

this->Feb = Feb;// расходы за Февраль

this->Mar = Mar;// расходы за Март

this->Apr = Apr;// расходы за Апрель

this->May = May;// расходы за Май

this->Jun = Jun;// расходы за Июнь

this->Jul = Jul;// расходы за Июнь

this->Aug = Aug;// расходы за Август

this->Sep = Sep;// расходы за Сентабрь

this->Oct = Oct;// расходы за Октябрь

this->Nov = Nov;// расходы за Ноябрь

this->Dec = Dec;// расходы за Декабрь

}

void ExpenseRecord::InputExpense()//ввод затрат за год по месяцам

{

int \*YearIn = new int;

int \*JanIn = new int;

int \*FebIn = new int;

int \*MarIn = new int;

int \*AprIn = new int;

int \*MayIn = new int;

int \*JunIn = new int;

int \*JulIn = new int;

int \*AugIn = new int;

int \*SepIn = new int;

int \*OctIn = new int;

int \*NovIn = new int;

int \*DecIn = new int;

cout<<"Enter Year"<<endl;

cin>>\*YearIn;

cout<<"Enter all Expenses from January with spaces in between\n";

cin>>\*JanIn>>\*FebIn>>\*MarIn>>\*AprIn>>\*MayIn>>\*JunIn>>\*JulIn>>\*AugIn>>\*SepIn>>\*OctIn>>\*NovIn>>\*DecIn;

v.push\_back(ExpenseRecord(\*YearIn,\*JanIn,\*FebIn,\*MarIn,\*AprIn,\*MayIn,\*JunIn,\*JulIn,\*AugIn,\*SepIn,\*OctIn,\*NovIn,\*DecIn));

delete YearIn;

delete JanIn;

delete FebIn;

delete MarIn;

delete AprIn;

delete MayIn;

delete JunIn;

delete JulIn;

delete AugIn;

delete SepIn;

delete OctIn;

delete NovIn;

delete DecIn;

}

void ExpenseRecord::display()

{

int control = 0;

int menuu;

cout<<"Enter year: ";

cin>>menuu; //ввод года проверки

cout << setiosflags(ios::left);

for (unsigned int j = 0; j<v.size(); j++)

{

if (v[j].Year == menuu) //ищет вектор с нужным годом

{

control++;

cout<<"\n Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec\n";

cout<<"-----------------------------------------------------------------------------------------------------"<<endl;

//Затраты

cout<< setw(10) <<"Expenses"<< setw(8) << v[j].Jan<< setw(8) << v[j].Feb<< setw(8) << v[j].Mar<< setw(8) << v[j].Apr

<< setw(8) << v[j].May<< setw(8) << v[j].Jun<< setw(8) << v[j].Jul<< setw(8) << v[j].Aug<< setw(8) << v[j].Sep

<< setw(8) << v[j].Oct<< setw(8) << v[j].Nov<< setw(8) << v[j].Dec<<endl;

for(unsigned i = 0;i<vProf.size();i++)

{

if(vProf[i].Year == v[j].Year)

{

//Доходы

cout<< setw(10) << "Income" << setw(8) << vProf[i].Jan<< setw(8) << vProf[i].Feb<< setw(8) << vProf[i].Mar<< setw(8) << vProf[i].Apr

<< setw(8) << vProf[i].May<< setw(8) << vProf[i].Jun<< setw(8) << vProf[i].Jul<< setw(8) << vProf[i].Aug<< setw(8) << vProf[i].Sep

<< setw(8) << vProf[i].Oct<< setw(8) << vProf[i].Nov<< setw(8) << vProf[i].Dec<<endl;

//Доходы - Затраты

cout<< setw(10) << "Profit"<< setw(8) << vProf[i].Jan-v[j].Jan<< setw(8) << vProf[i].Feb-v[j].Feb<< setw(8) << vProf[i].Mar-v[j].Mar

<< setw(8) << vProf[i].Apr-v[j].Apr<< setw(8) << vProf[i].May-v[j].May<< setw(8) << vProf[i].Jun-v[j].Jun<< setw(8) << vProf[i].Jul-v[j].Jul

<< setw(8) << vProf[i].Aug-v[j].Aug<< setw(8) << vProf[i].Sep-v[j].Sep<< setw(8) << vProf[i].Oct-v[j].Oct<< setw(8) << vProf[i].Nov-v[j].Nov<< setw(8) << vProf[i].Dec-v[j].Dec<<endl;

cout<<"-----------------------------------------------------------------------------------------------------"<<endl;

//Профит

cout<<"Profit: "<<vProf[i].Jan-v[j].Jan+vProf[i].Apr-v[j].Apr+vProf[i].Jul-v[j].Jul+vProf[i].Oct-v[j].Oct+vProf[i].Feb-v[j].Feb+vProf[i].May-v[j].May+vProf[i].Aug-v[j].Aug+vProf[i].Nov-v[j].Nov+vProf[i].Mar-v[j].Mar+vProf[i].Jun-v[j].Jun+vProf[i].Sep-v[j].Sep+vProf[i].Dec-v[j].Dec<<endl;

}

}

}else{if (j == v.size()-1 && control == 0){

cout<<"Not found"<<endl;}

}

cout << endl;

}

}

void ClearExpense()

{

v.clear();

}

main.cpp

#include "bekker.h"

#include "products.h"

int main()

{

MainInterface \*MainMenu = new MainInterface;

MainMenu->interactMain();

delete MainMenu;

}

maininteface.cpp

#include "bekker.h"

#include "products.h"

void MainInterface::interactMain()

{

ProdInit();

do

{

cout<<"Choose profile:\n 1. Cashier \n 2. Manager \n 3. Exit\n";

cin>>menu;

switch (menu)

{

case '1': //cashier

do

{

system("cls");

cout<<" Add order 'a'\n"

<<" Exit. 'e'\n" ;

cin>>ch;

switch(ch)

{

case 'a': // добавление заказ

{

Order \*Temp = new Order;//создаем обьект класса Order

Temp->AddOrder();//отправляем указатель на созданный обьект в метод добавление заказа

delete Temp;//опустошаем указатель

}

break;

case 'e': // выход

break;

default:

cout<<"\a";

break;

}

}while(ch!='e');

break;

case '2': //Manager

do

{

system("cls");

cout << " AnnualReport 'r',\n"

<< " Add report 'o',\n"

<< " Add product 'a',\n"

<< " Product List 'l',\n"

<< " Exit 'e':\n ";

cin>>ch2;

switch (ch2)

{

case 'r': //просмотр годового отчета

{

//Автоматический Перещёт Доходов

ProfitRecord \*Temp = new ProfitRecord;

Temp->InputProfit();

delete Temp;

//Вывод годового отчёта

ExpenseRecord \*Temp2 = new ExpenseRecord;

Temp2->display();

system("pause");

delete Temp2;

}

break;

case 'o': //описание Расходов за год

{

//Ввод расходов за год по месяцам

ExpenseRecord \*Temp = new ExpenseRecord;

Temp->InputExpense();

delete Temp;

}

break;

case 'a':

{

do{

system("cls");

cout << " Add Cake 'c',\n"

<< " Add bread 'b',\n"

<< " Add pastry 'p',\n"

<< " Go back 'e',\n";

cin>>chp;

switch (chp)

{

case 'c': //добавление торта в список продуктов

{

Cake \*Temp = new Cake;

Temp->addCake();

delete Temp;

}

break;

case 'b'://добавление хлеба в список продуктов

{

Bread \*Temp = new Bread;

Temp->addBread();

delete Temp;

}

break;

case 'p'://добавление конд изделий в список продуктов

{

Pastry \*Temp = new Pastry;

Temp->addPastry();

delete Temp;

}

break;

case 'e':

break;

default:

cout<<"\a";

break;

}//конец switch (chp)

}while(chp != 'e');

}//конец 'a'

break;

case 'l': //вывод списка продуктов

{

system("cls");

Cake \*Temp = new Cake;

Bread \*Temp2 = new Bread;

Pastry \*Temp3 = new Pastry;

cout << setiosflags(ios::left);//выравнивание по левой части

cout<< setw(3) << "No" << setw(15) << "Name" << setw(15) << "Weight" << setw(5) << "Price" <<endl;

Temp->CakeList();

Temp2->BreadList();

Temp3->PastryList();

delete Temp;

delete Temp2;

delete Temp3;

system("pause");

}

break;

case 'e':

break;

default:

cout<<"\a";

break;

}//конец switch (ch2)

system("cls");

}while(ch2!='e');

break;

default:

cout<<"\a";

break;

}//конец switch (menu)

system("cls");

}while(menu != '3');

ListClear();

ClearExpense();

cout<<"\nAdieu!\n";

}

prodconstruct.cpp

#include "products.h"

Cake::Cake(string Name,int Weight,float CostPrice,float Price){

this->Name=Name;

this->Weight=Weight;

this->CostPrice=CostPrice;

this->Price=Price;

}

Pastry::Pastry(string Name,int Weight,float CostPrice,float Price){

this->Name=Name;

this->Weight=Weight;

this->CostPrice=CostPrice;

this->Price=Price;

}

Bread::Bread(string Name,int Weight,float CostPrice,float Price){

this->Name=Name;

this->Weight=Weight;

this->CostPrice=CostPrice;

this->Price=Price;

}

Order::Order(int Year,int Month,float Sum ,float Profit){

this->Year=Year;

this->Month=Month;

this->Sum=Sum;

this->Profit=Profit;

}

# 15. Руководство пользователя

Для работы с программой предполагается два разных человека. Один работает кассиром, второй является представителем управляющего персонала. В случае если программу использует кассир, ему необходимо:

1. Запустить программу

2. Ввести число 1, чтобы войти в меню кассира

3. Ввести букву “a” английскую, чтобы сделать заказ

4. Выбрать в меню нужный тип продукта и ввести соответствующий символ (“c” для тортов,”b” для хлеба,”p” для булок)

5. Выбрать нужный продукт, введя соответствующую цифру

6. Ввести количество требуемого товара

7. Нажать “d” для завершения заказа

8. На экран будет выведена цена, чтобы выйти нажмите на любую клавишу

9. Чтобы выйти в главное меню нажмите “e”

В случае если программу использует управляющий персонал:

1. Запустить программу

2. Ввести цифру 2 для входа в интерфейс менеджера

3. Выбрать из меню функции «Annual report», «Add report», «Add product» или «Product list»

Для выбора меню Annual report необходимо предварительно создать отчет с помощью функции Add report

Если выбрано меню «Add report»:

1) Ввести букву o для входа в меню

2) Ввести год

3) Ввести расходы за 12 месяцев через пробел

Если выбрано меню «Annual report»:

1) Ввести букву r для входа в меню

2) Ввести год, за который необходим отчет

3) После ознакомления с отчетом нажать любую клавишу для выхода

Если выбрано меню «Add product»:

1) Ввести букву a для входа в меню

2) Выбрать категорию, в которую будет добавлен новый продукт (“c” для тортов,”b” для хлеба,”p” для булок)

3) Ввести название нового продукта

4) Ввести вес продукта

5) Ввести себестоимость

6) Ввести стоимость нового продукта

7) Для выхода ввести букву e

Если выбрано меню «Product list»

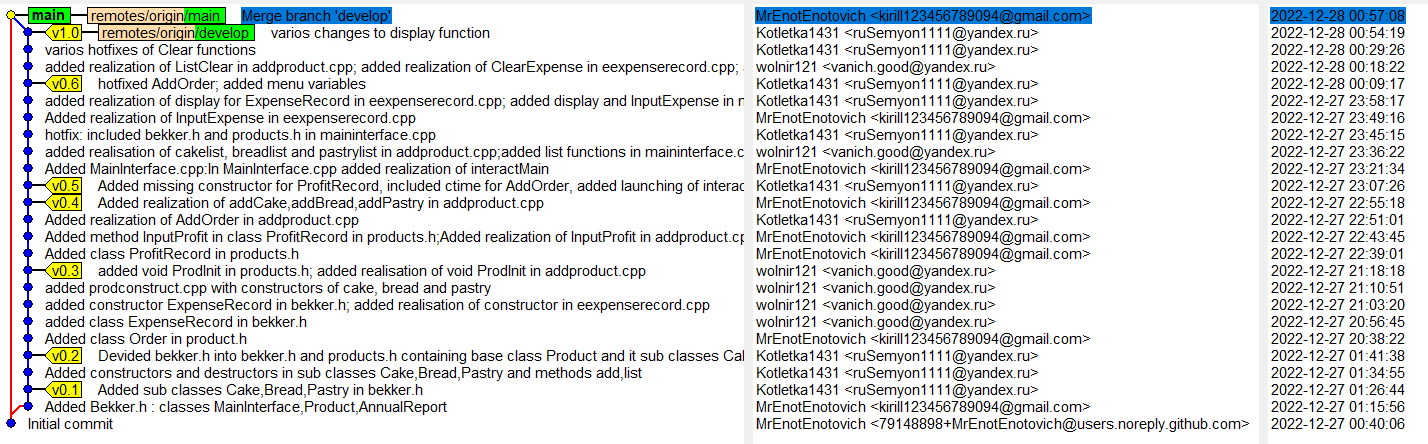
1) Ввести букву l для входа в меню

2) После ознакомления со списком нажать любую клавишу для выхода

# Ссылка на репозиторий GitHub

https://github.com/MrEnotEnotovich/TeamProjectBekker

# Графическая интерпретация истории проекта на GitHub с графического подсмотрщик истории “gitk”

  
Рисунок 5 – История проекта на Github

# Мой вклад в проект

bekker.h

#ifndef BEKKER\_H

#define BEKKER\_H

#include <iostream>

#include <string>

using namespace std;

class AnnualReport

{

protected:

int Year;

int Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec;

};

class MainInterface

{

private: //Поле

char ch,ch2,chp,menu;

public: //Методы

MainInterface(){}

~MainInterface() {}

void interactMain();

};

#endif // BEKKER\_H

products.h

//Базовый класс Product

class Product

{

public:

string Name;

int Weight; // вес

float CostPrice; //себестоимость

float Price; //рыночная стоимость(на полке на магазина)

};

//производный класс Order от базового Product(чтобы иметь доступ к полям класса Product)

//Добавляет возможность заказа из списка продуктов

class Order : public Product

{

public:

int Year,Month;

float Sum;

float Profit;

Order(int Year,int Month,float Sum ,float Profit);

Order(){}

~Order(){}

};

//Класс ProfitRecord для расчетов доходов с заказов Order за каждый месяц

class ProfitRecord

{

public:

int Year;

int Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec;

ProfitRecord(){}

ProfitRecord(int ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float ,float );

~ProfitRecord(){}

void InputProfit();

};

extern vector <ProfitRecord> vProf;

addproduct.cpp

vector <Order> vOrd;

vector <ProfitRecord> vProf;

void ProfitRecord::InputProfit()

{

int match;

vProf.clear();

for (unsigned int j = 0; j<vOrd.size() ; j++ ) //проходим по вектору указателей на обьекты заказов(сколько заказов столько и выполнится for)

{

match = 0;

for(unsigned int i = 0; i<vProf.size() ; i++ ) //проходим по вектору указателей на обьекты прибыли(сколько записей прибыли столько и выполнится)

{

if(vProf[i].Year == vOrd[j].Year) //ищем в векторе прибыли обьект с таким же годом,как в векторе заказов

{

match = 1; // если существует вектора прибыли с таким годом,то записываем в него данные

switch(vOrd[j].Month)

{

case 1://суммируем в конкретный месяц вектора прибыли <- поле прибыли с вектора заказов

vProf[i].Jan += vOrd[j].Profit;

break;

case 2:

vProf[i].Feb += vOrd[j].Profit;

break;

case 3:

vProf[i].Mar += vOrd[j].Profit;

break;

case 4:

vProf[i].Apr += vOrd[j].Profit;

break;

case 5:

vProf[i].May += vOrd[j].Profit;

break;

case 6:

vProf[i].Jun += vOrd[j].Profit;

break;

case 7:

vProf[i].Jul += vOrd[j].Profit;

break;

case 8:

vProf[i].Aug += vOrd[j].Profit;

break;

case 9:

vProf[i].Sep += vOrd[j].Profit;

break;

case 10:

vProf[i].Oct += vOrd[j].Profit;

break;

case 11:

vProf[i].Nov += vOrd[j].Profit;

break;

case 12:

vProf[i].Dec += vOrd[j].Profit;

break;

}//конец switch(vOrd.Month)

}

}

if(match == 0) //если не существует вектора прибыли в данном году,введенный пользователем,то создаем его и записываем в конкретный месяц прибыль

{

switch(vOrd[j].Month)

{

case 1:

vProf.push\_back(ProfitRecord(vOrd[j].Year,vOrd[j].Profit,0,0,0,0,0,0,0,0,0,0,0));

break;

case 2:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,vOrd[j].Profit,0,0,0,0,0,0,0,0,0,0));

break;

case 3:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,vOrd[j].Profit,0,0,0,0,0,0,0,0,0));

break;

case 4:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,vOrd[j].Profit,0,0,0,0,0,0,0,0));

break;

case 5:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,vOrd[j].Profit,0,0,0,0,0,0,0));

break;

case 6:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,vOrd[j].Profit,0,0,0,0,0,0));

break;

case 7:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,vOrd[j].Profit,0,0,0,0,0));

break;

case 8:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,vOrd[j].Profit,0,0,0,0));

break;

case 9:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,vOrd[j].Profit,0,0,0));

break;

case 10:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,0,vOrd[j].Profit,0,0));

break;

case 11:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,0,0,vOrd[j].Profit,0));

break;

case 12:

vProf.push\_back(ProfitRecord(vOrd[j].Year,0,0,0,0,0,0,0,0,0,0,0,vOrd[j].Profit));

break;

}//конец switch(vOrd.Month)

}

}

}

void Cake::addCake() //добавление новых позиций

{

string \*CName = new string;

int \*CWeight = new int;

float \*CCostPrice = new float;

float \*CPrice = new float;

cout<<"Enter Cake name:\n";

cin>>\*CName;

cout<<"\nEnter Cake weight:\n";

cin>>\*CWeight;

cout<<"\nEnter Cake cost price:\n";

cin>>\*CCostPrice;

cout<<"\nEnter Cake price:\n";

cin>>\*CPrice;

vCake.push\_back(Cake(\*CName,\*CWeight,\*CCostPrice,\*CPrice));

delete CName;

delete CWeight;

delete CCostPrice;

delete CPrice;

}

void Bread::addBread()

{

string \*CName = new string;

int \*CWeight = new int;

float \*CCostPrice = new float;

float \*CPrice = new float;

cout<<"Enter Bread name:\n";

cin>>\*CName;

cout<<"\nEnter Bread weight:\n";

cin>>\*CWeight;

cout<<"\nEnter Bread cost price:\n";

cin>>\*CCostPrice;

cout<<"\nEnter Bread price:\n";

cin>>\*CPrice;

vBread.push\_back(Bread(\*CName,\*CWeight,\*CCostPrice,\*CPrice));

delete CName;

delete CWeight;

delete CCostPrice;

delete CPrice;

}

void Pastry::addPastry()

{

string \*CName = new string;

int \*CWeight = new int;

float \*CCostPrice = new float;

float \*CPrice = new float;

cout<<"Enter Pastry name:\n";

cin>>\*CName;

cout<<"\nEnter Pastry weight:\n";

cin>>\*CWeight;

cout<<"\nEnter Pastry cost price:\n";

cin>>\*CCostPrice;

cout<<"\nEnter Pastry price:\n";

cin>>\*CPrice;

vPastry.push\_back(Pastry(\*CName,\*CWeight,\*CCostPrice,\*CPrice));

delete CName;

delete CWeight;

delete CCostPrice;

delete CPrice;

}

void MainInterface::interactMain()

{

ProdInit();

do

{

cout<<"Choose profile:\n 1. Cashier \n 2. Manager \n 3. Exit\n";

cin>>menu;

switch (menu)

{

case '1': //cashier

do

{

system("cls");

cout<<" Add order 'a'\n"

<<" Exit. 'e'\n" ;

cin>>ch;

switch(ch)

{

case 'a': // добавление заказ

{

Order \*Temp = new Order;//создаем обьект класса Order

Temp->AddOrder();//отправляем указатель на созданный обьект в метод добавление заказа

delete Temp;//опустошаем указатель

}

break;

case 'e': // выход

break;

default:

cout<<"\a";

break;

}

}while(ch!='e');

break;

case '2': //Manager

do

{

system("cls");

cout << " AnnualReport 'r',\n"

<< " Add report 'o',\n"

<< " Add product 'a',\n"

<< " Product List 'l',\n"

<< " Exit 'e':\n ";

cin>>ch2;

switch (ch2)

{

case 'r': //просмотр годового отчета

{

//Автоматический Перещёт Доходов

ProfitRecord \*Temp = new ProfitRecord;

Temp->InputProfit();

delete Temp;

//Вывод годового отчёта

}

break;

case 'o': //описание Расходов за год

{

//Ввод расходов за год по месяцам

}

break;

case 'a':

{

do{

system("cls");

cout << " Add Cake 'c',\n"

<< " Add bread 'b',\n"

<< " Add pastry 'p',\n"

<< " Go back 'e',\n";

cin>>chp;

switch (chp)

{

case 'c': //добавление торта в список продуктов

{

Cake \*Temp = new Cake;

Temp->addCake();

delete Temp;

}

break;

case 'b'://добавление хлеба в список продуктов

{

Bread \*Temp = new Bread;

Temp->addBread();

delete Temp;

}

break;

case 'p'://добавление конд изделий в список продуктов

{

Pastry \*Temp = new Pastry;

Temp->addPastry();

delete Temp;

}

break;

case 'e':

break;

default:

cout<<"\a";

break;

}//конец switch (chp)

}while(chp != 'e');

}//конец 'a'

break;

case 'l': //вывод списка продуктов

{

system("cls");

Cake \*Temp = new Cake;

Bread \*Temp2 = new Bread;

Pastry \*Temp3 = new Pastry;

cout << setiosflags(ios::left);//выравнивание по левой части

cout<< setw(3) << "No" << setw(15) << "Name" << setw(15) << "Weight" << setw(5) << "Price" <<endl;

Temp->CakeList();

Temp2->BreadList();

Temp3->PastryList();

delete Temp;

delete Temp2;

delete Temp3;

system("pause");

}

break;

case 'e':

break;

default:

cout<<"\a";

break;

}//конец switch (ch2)

system("cls");

}while(ch2!='e');

break;

default:

cout<<"\a";

break;

}//конец switch (menu)

system("cls");

}while(menu != '3');

cout<<"\nAdieu!\n";

}

eexpenserecord.cpp

void ExpenseRecord::InputExpense()//ввод затрат за год по месяцам

{

int \*YearIn = new int;

int \*JanIn = new int;

int \*FebIn = new int;

int \*MarIn = new int;

int \*AprIn = new int;

int \*MayIn = new int;

int \*JunIn = new int;

int \*JulIn = new int;

int \*AugIn = new int;

int \*SepIn = new int;

int \*OctIn = new int;

int \*NovIn = new int;

int \*DecIn = new int;

cout<<"Enter Year"<<endl;

cin>>\*YearIn;

cout<<"Enter all Expenses from January with spaces in between\n";

cin>>\*JanIn>>\*FebIn>>\*MarIn>>\*AprIn>>\*MayIn>>\*JunIn>>\*JulIn>>\*AugIn>>\*SepIn>>\*OctIn>>\*NovIn>>\*DecIn;

v.push\_back(ExpenseRecord(\*YearIn,\*JanIn,\*FebIn,\*MarIn,\*AprIn,\*MayIn,\*JunIn,\*JulIn,\*AugIn,\*SepIn,\*OctIn,\*NovIn,\*DecIn));

delete YearIn;

delete JanIn;

delete FebIn;

delete MarIn;

delete AprIn;

delete MayIn;

delete JunIn;

delete JulIn;

delete AugIn;

delete SepIn;

delete OctIn;

delete NovIn;

delete DecIn;

}

# Заключение

Проект Пекарни был разработан с помощью CASE-системы Qt Creator (версии 5.4.2) на языке C++.Была применена система контроля версий Git. Все задуманные прецеденты были реализованы. Зависаний и сбоев в программе не наблюдалось. При разработке был использован принцип раздельной компиляции, реализована очистка памяти. Неиспользуемых переменных нет, избыточные алгоритмы отсутствуют.

В ходе разработки были созданы следующие UML диаграммы:

- Вариантов использования  
- Действий  
- Классов  
- Последовательностей

Цель, заключающаяся в моделировании ситуации выполнения заказа и разработки программного обеспечения, была достигнута.