**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Физико-технический институт**

**Командное задание по разработке программного обеспечения в рамках “Унифицированного процесса разработки ПО”**

**Выполнили:**

Студенты

Физико-технического

Института группы 21318

Иванов Константин

ПереверзеваДарья

Сергиенко Данил

**Преподаватель:**

Бульба Артем Владимирович

**Петрозаводск 2021**

**Командный проект "Цветочный магазин"**

**Цель работы:** Разработать ПО для учета и обработки данных, цветочного магазина. Вести разработку с помощью системы контроля версий Git через веб-сервис GitHub. Закрепить полученные знания, использования языков С++ и UML, практически применить концепцию “Унифицированного процесса разработки ПО.

**Программная реализация:**

Главный файл:

Main.cpp

Запись информации о цветке и методы работы с ним:

Flower.h

Flower.cpp

Запись сформированного заказа и методы работы с ним:

Order.h

Order.cpp

Список заказов:

Order\_list.h

Order\_list.cpp

Владелец цветочного и его интерфейс взаимодействия с программой

Owner.h

Owner.cpp

Класс собирающий программу в одно целое, вход в систему

Program.h

Program.cpp

Продавец в цветочном

Salesman.h

Salesman.cpp

Список продавцов

Salesman\_list.h

Salesman\_list.cpp

Хранилище цветов

Storage.h

Storage.cpp

Отчет о деятельности цветочного

Summary.h

Summary.cpp

Абстрактный пользователь системы

User.h

User.cpp

**Пошаговое описание процесса разработки**

Цветочный магазин торгует цветами. Цветы поставляются оптом поставщиком. В цветочном магазине работает только продавец, он занимается продажей цветов, а также может, по желанию клиента, упаковывать цветы. Владелец магазина может просматривать небольшой отчет по деятельности магазина.И владелец и продавец имеют доступ к текущему состоянию склада(наличие тех или иных цветов и их количество, цену).

Программе необходимо будет работать с приходной накладной от поставщика, предоставляющий цветы в магазин вида:



Рис1. Пример приходной накладной

Действующие лица(требует доработки)

Продавец

Человек, работающий с клиентами. Он собирает заказ из цветов, предлагает услуги такие как упаковку или доставку. Так же он осуществляет приемку цветов на склад.

Владелец

Является владельцем цветочного магазина. Он имеет доступ к статистике магазина (состояние склада и отчет)

Система

Занимается обработкой информации об изменении количества цветов на складе,

**Описание вариантов использования**

* Запустить программу

При запуске программы пользователю необходимо указать является ли он продавцом или владельцем, указать свой id.

* Принять цветы на склад

Продавец принимает цветы на склад. На складе вид цветка может как быть так и отсутствовать. Если в цветок есть то есть он хотя бы раз присутствовал на складе (то есть от него осталась запись, даже с количеством = 0), то обновляется его количество. Если цветка не было создается новая запись

* Сформировать заказ

Продавец формирует заказ для клиента. По желанию клиента может быть оформлена доставка и упаковка заказа. В заказе указывается id продавца, список купленных цветов(количество и цена), цена доставки и упаковки.

* Упаковать заказ

Упаковка заказа включается продавцом по желанию клиента в цену заказа единоразово

* Оформить доставку заказа

Доставка заказа включается продавцом по желанию клиента в цену заказа единоразово

* Вывести список цветов на складе

Выводит список цветов со склад, для того чтобы продавец мог уточнить количество цветов на складе

* Составить отчет

Система формирует отчет, состоящий из количества цветов на складе к концу периода, выручки за весь период, зарплат сотрудников и чистой прибыли

* Добавить сотрудника

владелец добавляет нового сотрудника в штат

* Удалить сотрудника

владелец увольняет сотрудника

**Диаграммы деятельности**

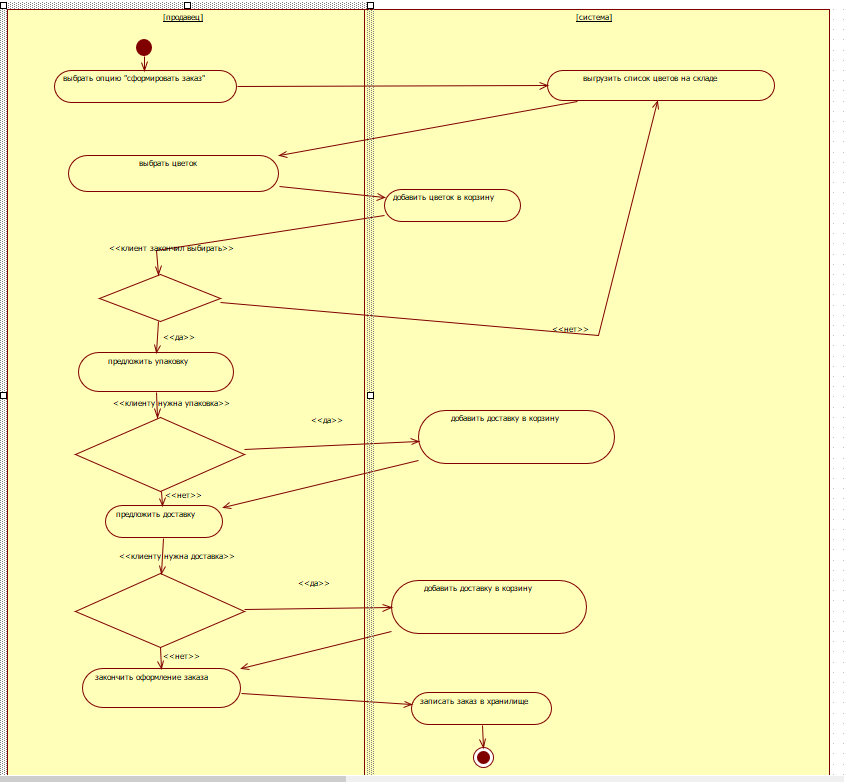


Рис2. Диаграмма деятельности «сформировать заказ»

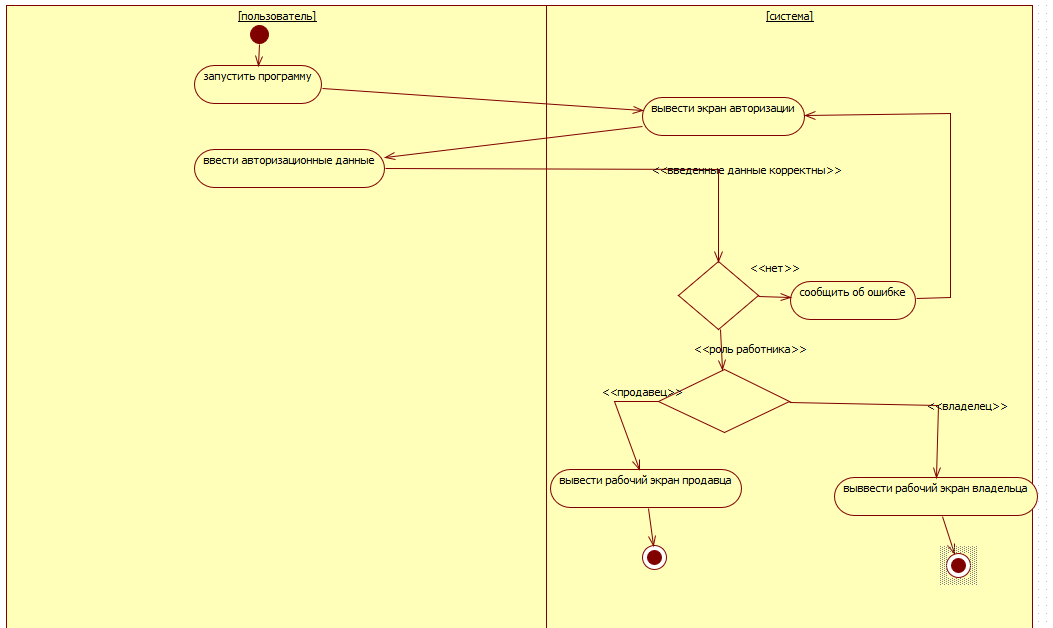


Рис3.диаграмма деятельности «запустить программу»

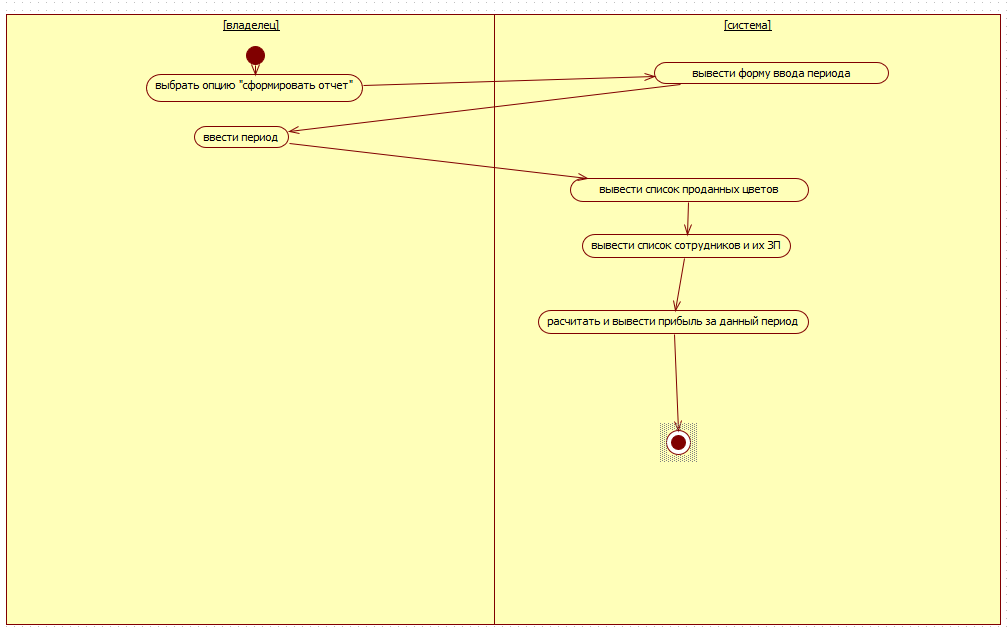


Рис4.диаграмма деятельности «сформировать отчет»

предварительный список существительных

1. Программа
2. Пользователь
3. Владелец
4. Склад цветов
5. Цветок
6. Стоимость цветка
7. Количество цветов на складе
8. штат сотрудников
9. заказ
10. Упаковка
11. Доставка
12. Экран авторизации
13. Экран продавца
14. Экран владельца
15. Экран добавления продавца
16. Экран удаления продавца
17. Экран формирования заказа
18. Экран приема цветов на склад
19. Период
20. Выручка
21. Отчет за период
22. Таблица проданных цветов
23. Таблица зп продавцов
24. Чистая прибыль

**Уточненный список существительных**

1. Программа

Основной класс в нем хранятся экземпляры классов, в нем происходит инициализация программы

1. Пользователь

Абстрактный класс родитель класса продавец и владелец.

1. Цветок

Хранит основную информацию о цветке (название, количество, цена), существует для более удобного взаимодействия классов заказ и склад цветов.

1. Владелец

Учетная запись владельца, через неё осуществляется взаимодействие владельца с системой

1. Продавец

Учетная запись продавца, предоставляет функционал необходимый для работы продавца

1. Заказ

Является записью сформированного заказа, с помощью него происходит формирование заказа продавцом и убыль товара со склада

1. Склад цветов

Хранит в себе список цветов на складе, а также способы взаимодействия с ним

1. Список продавцов

Хранит в себе список продавцов, а также способы взаимодействия с ним

1. Список заказов

Хранит в себе список заказов, а также способы взаимодействия с ним

1. Отчет

Небольшой отчет о деятельности цветочного магазина

**Заголовочные файлы**

Flower.h

#include"string"

#if !defined(\_FLOWER\_H)

#define \_FLOWER\_H

using namespace std;

class Flower {

private:

string flower\_name;

int quantity;

float price;

public:

Flower(string nm, int count, float flw\_price);

string get\_name();

int get\_quantity();

float get\_price();

void set\_quantity(int flw\_count);

void set\_price(float flw\_price);

};

#endif //\_FLOWER\_H

Order.h

#include <ctime>

#include<list>

#include<Storage.h>

#if !defined(\_ORDER\_H)

#define \_ORDER\_H

using namespace std;

class Order {

private:

tm date;

int salesman\_id;

bool is\_packaged;

bool need\_delivery;

float final\_price;

float profit;

public:

Order(int empl\_id,bool pack,bool deliver,float final, float prof);

Order(int empl\_id, bool pack, bool deliver, float final, float prof, tm dt):salesman\_id(empl\_id),is\_packaged(pack),need\_delivery(deliver),final\_price(final),profit(prof),date(dt)

{

/\*пусто\*/

}

void make\_a\_deal();

bool get\_pack();

bool get\_deliver();

int get\_salesman\_id();

float get\_final\_price();

float get\_profit();

tm get\_date();

void set\_pack(bool a);

void set\_diliver(bool a);

};

#endif //\_ORDER\_H

Order\_list.h

#if !defined(\_ORDER\_LIST\_H)

#define \_ORDER\_LIST\_H

#include "Order.h"

#include<iostream>

using namespace std;

class Order\_list {

private:

list<Order\*> order\_list;

list<Order\*>::iterator iter;

public:

~Order\_list();

void add\_order(Order \*order);

void get\_order\_list\_from\_file(char\* path);

void put\_order\_list\_to\_file(char\* path);

void get\_order\_list();

float total\_profit();

float total\_cost();

};

#endif //\_ORDER\_LIST\_H

Owner.h

#if !defined(\_OWNER\_H)

#define \_OWNER\_H

#include "Salesman\_list.h"

#include <string>

#include "User.h"

#include "Summary.h"

class Owner : public User {

private:

Salesman\_list\* slm\_l;

Storage\* storage;

Order\_list\* order\_list;

Summary\* summary;

public:

Owner(Salesman\_list\* ptr\_slm, string nm, Storage \*st, Order\_list \*ol, Summary\* tmp\_summary);

void add\_salesman();

void delete\_salesman();

void *interact*();

void show\_salesman\_list();

};

#endif //\_OWNER\_H

Program.h

#include "Storage.h"

#include "Order\_list.h"

#include "Owner.h"

#include "Salesman\_list.h"

#include "Summary.h"

#include <list>

#include <vector>

#include<iostream>

#include<string>

#include<fstream>

#ifndef \_PROGRAM\_H

#define \_PROGRAM\_H

using namespace std;

class Program

{

private:

Storage storage;

Order\_list order\_list;

Salesman\_list salesman\_list{&storage, &order\_list};

Owner\* owner ;

Summary\* summary;

User\* autorised\_user;

public:

Program(string nm);

~Program();

void log\_in();

};

#endif //\_PROGRAM\_H

Salesman.h

#if !defined(\_SALESMAN\_H)

#define \_SALESMAN\_H

#include "User.h"

#include"string"

#include "Storage.h"

#include "Order\_list.h"

#include<iostream>

using namespace std;

class Salesman : public User {

private:

Storage\* storage;

Order\_list\* order\_list;

Storage\* shoping\_list;

public:

Salesman(string nm, int sId,Storage\* st, Order\_list\* ol);

Salesman(string nm, int sId);

void add\_flower\_to\_storage();

void form\_order();

void *interact*();

};

#endif //\_SALESMAN\_H

Salesman\_list.h

#include"Salesman.h"

#include<list>

#include<fstream>

#include<sstream>

#include<iostream>

#if !defined(\_SALESMAN\_LIST\_H)

#define \_SALESMAN\_LIST\_H

class Salesman\_list {

private:

list<Salesman\*> slm\_list;

list<Salesman\*>::iterator iter;

Storage\* storage;

Order\_list\* order\_list;

public:

Salesman\_list(Storage\* st, Order\_list\* ol):storage(st),order\_list(ol){

/\*пусто\*/

}

~Salesman\_list();

void add\_salerman(Salesman \*tmp\_slm);

void get\_salerman\_list();

void remove\_salesman(int id);

Salesman\* find\_slm(int id);

void put\_list\_to\_file(char\* path);

void get\_list\_from\_file(char \*path);

bool in\_list(int id);

};

#endif //\_SALESMAN\_LIST\_H

Storage.h

#include<Flower.h>

#include<vector>

#include<string>

#include<fstream>

#include<sstream>

#include<iostream>

using namespace std;

#if !defined(\_STORAGE\_H)

#define \_STORAGE\_H

class Storage {

private:

vector<Flower\*> flower\_vect;

vector<Flower\*>::iterator iter;

public:

~Storage();

void add\_flower(Flower \*temp\_flw);

void get\_flower\_list();

void get\_storage\_from\_file(char\* path);

void put\_storage\_to\_file(char\* path);

bool in\_vect(string nm);

Flower\* find\_flower(string nm);

Storage\* choose\_flowers();

float storage\_cost();

void storage\_subtraction(Storage\* storage);

};

#endif //\_STORAGE\_H

Summary.h

#if !defined(\_SUMMARY\_H)

#define \_SUMMARY\_H

#include "Order\_list.h"

#include "Salesman\_list.h"

#include "Storage.h"

class Summary {

private:

Order\_list\* order\_list;

Salesman\_list\* salesman\_list;

Storage\* storage;

public:

Summary(Order\_list\* ord\_list,Salesman\_list\* slm\_list, Storage\* tmp\_storage):order\_list(ord\_list),salesman\_list(slm\_list),storage(tmp\_storage)

{

/\*Пусто\*/

}

void to\_sum\_up();

};

#endif //\_SUMMARY\_H

User.h

#include"string"

#if !defined(\_USER\_H)

#define \_USER\_H

using namespace std;

class User {

protected:

string name;

int id;

public:

virtual int *get\_id*();

virtual string *get\_name*();

virtual void *set\_name*(string nm);

virtual void *set\_id*(int ID);

virtual void *interact*()=0;

};

#endif //\_USER\_H

**Исходные файлы**

Main.cpp

#include <iostream>

#include<Program.h>

#include"string"

using namespace std;

int main()

{

Program program("Owner");

program.log\_in();

}

Flower.cpp

#include "Flower.h"

Flower::Flower(string nm, int count, float flw\_price){

flower\_name=nm;

quantity=count;

price=flw\_price;

}

string Flower::get\_name() {

return flower\_name;

}

int Flower::get\_quantity() {

return quantity;

}

float Flower::get\_price() {

return price;

}

void Flower::set\_price(float flw\_price) {

price=flw\_price;

}

void Flower::set\_quantity(int flw\_count) {

quantity=flw\_count;

}

Order.cpp

#include "Order.h"

char buffer[80];

Order::Order(int empl\_id, bool pack, bool deliver, float final, float prof):salesman\_id(empl\_id),is\_packaged(pack),need\_delivery(deliver),final\_price(final),profit(prof)

{

time\_t date1;

date1 = time(NULL);

localtime\_s(&date, &date1);

char\* format = "%d %m %Y";

strftime(buffer, 80, format, &date);

cout << "Current Date: " << buffer << endl;

}

void Order::make\_a\_deal() {

}

bool Order::get\_pack()

{

return is\_packaged;

}

bool Order::get\_deliver()

{

return need\_delivery;

}

void Order::set\_pack(bool a)

{

is\_packaged=a;

}

void Order::set\_diliver(bool a)

{

need\_delivery= a;

}

int Order::get\_salesman\_id()

{

return salesman\_id;

}

float Order::get\_final\_price()

{

return final\_price;

}

float Order::get\_profit()

{

return profit;

}

tm Order::get\_date()

{

time\_t date1;

date1 = time(NULL);

localtime\_s(&date, &date1);

return date;

}

Order\_list. Cpp

#include "Order\_list.h"

Order\_list::~Order\_list()

{

{

while (!order\_list.empty()){

iter=order\_list.begin();

delete \*iter;

order\_list.erase(iter);

}

}

}

void Order\_list::add\_order(Order\* order) {

order\_list.push\_back(order);

}

void Order\_list::get\_order\_list\_from\_file(char \*path)

{

ifstream fin;

string str;

bool pack, deliver;

int slm\_id;

float fin\_price, prof;

int year,day,mon;

fin.open(path);

if(fin.is\_open()){

while(!fin.eof()){

fin>>str;

if (str==""){

break;

}

pack = (str!="0");

str="";

fin>>str;

if (str==""){

break;

}

deliver=(str!="0");

str="";

fin>>str;

if (str==""){

break;

}

slm\_id=stoi(str);

str="";

fin>>str;

if (str==""){

break;

}

fin\_price=stof(str);

str="";

fin>>str;

if (str==""){

break;

}

prof=stof(str);

str="";

fin>>str;

if (str==""){

break;

}

year=stoi(str);

str="";

fin>>str;

if (str==""){

break;

}

mon=stoi(str);

str="";

fin>>str;

if (str==""){

break;

}

day=stoi(str);

tm date;

date.tm\_year;

date.tm\_mon;

date.tm\_mday;

Order\* tmp\_order = new Order(slm\_id,pack,deliver,fin\_price,prof, date);

order\_list.push\_back(tmp\_order);

}

}

fin.close();

}

void Order\_list::put\_order\_list\_to\_file(char \*path)

{

if (!order\_list.empty()){

ofstream fout;

fout.open(path);

iter= order\_list.begin();

while(iter!= order\_list.end()){

fout<<(\*iter)->get\_pack()<<endl;

fout<<(\*iter)->get\_deliver()<<endl;

fout<<(\*iter)->get\_salesman\_id()<<endl;

fout<<(\*iter)->get\_final\_price()<<endl;

fout<<(\*iter)->get\_profit()<<endl;

tm time = (\*iter)->get\_date();

fout<<time.tm\_year<<endl;

fout<<time.tm\_mon<<endl;

fout<<time.tm\_mday<<endl;

\*iter++;

}

fout.close();

}

}

void Order\_list::get\_order\_list()

{

iter= order\_list.begin();

while(iter!= order\_list.end()){

cout<<(\*iter)->get\_pack()<<endl;

cout<<(\*iter)->get\_deliver()<<endl;

cout<<(\*iter)->get\_salesman\_id()<<endl;

cout<<(\*iter)->get\_final\_price()<<endl;

cout<<(\*iter)->get\_profit()<<endl;

tm time = (\*iter)->get\_date();

cout<<time.tm\_year+1900<<endl;

cout<<time.tm\_mon+1<<endl;

cout<<time.tm\_mday<<endl;

\*iter++;

}

}

float Order\_list::total\_profit(){

float profit=0;

if (!order\_list.empty()){

iter= order\_list.begin();

while(iter!= order\_list.end()){

profit += (\*iter)->get\_profit();

\*iter++;

}

}

return profit;

}

float Order\_list::total\_cost()

{

float cost=0;

if (!order\_list.empty()){

iter= order\_list.begin();

while(iter!= order\_list.end()){

cost +=((\*iter)->get\_final\_price())-((\*iter)->get\_profit());

\*iter++;

}

}

return cost;

}

Owner. Cpp

#include<iostream>

#include "Owner.h"

using namespace std;

Owner::Owner(Salesman\_list\* ptr\_slm, string nm, Storage\* st, Order\_list\* ol, Summary \*tmp\_summary):slm\_l(ptr\_slm),storage(st),order\_list(ol),summary(tmp\_summary)

{

User::*set\_id*(0);

User::*set\_name*(nm);

}

void Owner::add\_salesman() {

int id;

string nm;

cout<<"Enter Name:"<<endl;

cin>>nm;

cin.ignore(80, '\n');

while(true){

cout<<"Enter ID:"<<endl;

cin>>id;

cin.ignore(80, '\n');

if (!slm\_l->in\_list(id) && id!=0){

Salesman\* tmp\_slm= new Salesman(nm,id,storage,order\_list);

slm\_l->add\_salerman(tmp\_slm);

break;

}

else{

cout<<"Id is occupied. Choose enother one.";

}

}

}

void Owner::delete\_salesman() {

int id;

while(true){

cout<<"Enter ID"<<endl;

cin>>id;

cout<<id<<endl;

cin.ignore(80, '\n');

if (slm\_l->in\_list(id)){

slm\_l->remove\_salesman(id);

break;

}else{

cout<<"can't find salesman";

}

}

}

void Owner::*interact*() {

while(true){

cout << "Press 'i' to add new salesman, \n"

<< "press 'd' to delete salesman, \n"

<< "press 's' to see sum up,\n"

<< "press 'l' to see salesmans lists, \n"

<< "press 'q' to quit: \n";

char ch = cin.get();

cin.ignore(80, '\n');

switch(ch){

case'i':

add\_salesman();

break;

case'd':

delete\_salesman();

break;

case's':

summary->to\_sum\_up();

break;

case'l':

show\_salesman\_list();

break;

case'q':

return;

break;

default:

cout<<"wrong button. \n";

break;

}

}

}

void Owner::show\_salesman\_list()

{

slm\_l->get\_salerman\_list();

}

Program. Cpp

char\* slm\_path = "salesman\_list.txt";

char\* storage\_path = "storage.txt";

char\* order\_list\_path= "orderlist.txt";

#include "program.h"

Program::Program(string nm){

summary = new Summary(&order\_list,&salesman\_list,&storage);

owner= new Owner(&salesman\_list , nm, &storage, &order\_list,summary);

salesman\_list.get\_list\_from\_file(slm\_path);

storage.get\_storage\_from\_file(storage\_path);

order\_list.get\_order\_list\_from\_file(order\_list\_path);

}

Program::~Program(){

salesman\_list.put\_list\_to\_file(slm\_path);

storage.put\_storage\_to\_file(storage\_path);

order\_list.put\_order\_list\_to\_file(order\_list\_path);

}

void Program::log\_in() {

int Tempid;

while(true){

cout<<"Enter your id"<<endl;

cin >> Tempid;

cin.ignore(80, '\n');

if (Tempid==0){

autorised\_user=owner;

cout<<"Welcome! "<<autorised\_user->*get\_name*()<<endl;

autorised\_user->*interact*();

break;

}

else{

if (salesman\_list.in\_list(Tempid)){

autorised\_user=salesman\_list.find\_slm(Tempid);

cout<<"Welcome! "<<autorised\_user->*get\_name*()<<endl;

autorised\_user->*interact*();

break;

}

else {

cout<<"Inpur error."<<endl;

continue;

}

}

}

}

Salesman. Cpp

#include "Salesman.h"

Salesman::Salesman(string nm, int sId,Storage\* st, Order\_list\* ol):storage(st),order\_list(ol){

User::name=nm;

User::id=sId;

}

void Salesman::add\_flower\_to\_storage() {

string flower\_name;

int count, price;

cout<<"enter flower name: "<<endl;

cin>>flower\_name;

cin.ignore(80, '\n');

cout<<"enter flower count"<<endl;

cin>>count;

cin.ignore(80, '\n');

cout<<"enter each flower price"<<endl;

cin>>price;

cin.ignore(80, '\n');

Flower\* ptr\_flower= new Flower(flower\_name,count,price);

storage->add\_flower(ptr\_flower);

storage->get\_flower\_list();

}

void Salesman::form\_order() {

float price, profit;

bool pack, deliver;

Order\* ord;

shoping\_list=storage->choose\_flowers();

price=shoping\_list->storage\_cost()\*1.35f;

cout << "Do client need package(y/n), \n";

while (true){

char ch = cin.get();

cin.ignore(80, '\n');

switch(ch){

case'y':

pack=true;

price+=20;

break;

case'n':

pack=false;

break;

default:

cout<<"wrong button. \n";

continue;

}

break;

}

cout << "Do client need deliver(y/n), \n";

while (true){

char ch = cin.get();

cin.ignore(80, '\n');

switch(ch){

case'y':

deliver=true;

price+=200;

break;

case'n':

deliver=false;

break;

default:

cout<<"wrong button. \n";

continue;

}

break;

}

cout<<"Total price: "<<price<<". Do you agree to the price?(y/n)";

while (true){

char ch = cin.get();

cin.ignore(80, '\n');

profit = price-shoping\_list->storage\_cost();

switch(ch){

case'y':

shoping\_list->storage\_subtraction(storage);

ord = new Order(User::*get\_id*(),pack,deliver,price,profit);

order\_list->add\_order(ord);

cout<<"Deal is done";

break;

case'n':

break;

default:

cout<<"wrong button. \n";

continue;

}

break;

}

}

void Salesman::*interact*() {

while(true){

cout << "Press 'i' to add flower to storage, \n"

<< "press 'f' to form order, \n"

<< "press 's' to see storage, \n"

<< "press 'q' to quit: \n";

char ch = cin.get();

cin.ignore(80, '\n');

switch(ch){

case'i':

add\_flower\_to\_storage();

break;

case'f':

form\_order();

break;

case's':

storage->get\_flower\_list();

break;

case'q':

return;

break;

default:

cout<<"wrong button. \n";

break;

}

}

}

Salesman\_list. Cpp

#include "Salesman\_list.h"

Salesman\_list::~Salesman\_list()

{

{

while (!slm\_list.empty()){

iter=slm\_list.begin();

delete \*iter;

slm\_list.erase(iter);

}

}

}

void Salesman\_list::add\_salerman(Salesman\* tmp\_slm) {

slm\_list.push\_back(tmp\_slm);

}

void Salesman\_list::get\_salerman\_list() {

if (slm\_list.empty()){

cout<<"list is empty"<<endl;

}

else{

cout<<"Salesman list"<<endl;

cout<<"=========================="<<endl;

cout<<"id";

cout<<"\t||\t";

cout<<"name";

cout<<"\t||"<<endl;

cout<<"=========================="<<endl;

iter= slm\_list.begin();

while(iter!= slm\_list.end()){

cout<<(\*iter)->get\_id();

cout<<"\t||\t";

cout<<(\*iter)->get\_name();

cout<<"\t||"<<endl;

\*iter++;

}

cout<<"=========================="<<endl;

}

}

void Salesman\_list::remove\_salesman(int id) {

if (slm\_list.empty()){

cout<<"list is empty"<<endl;

}

else{

iter= slm\_list.begin();

while(iter!= slm\_list.end()){

if ((\*iter)->get\_id() == id){

slm\_list.erase(iter);

break;

}

\*iter++;

}

}

}

Salesman \*Salesman\_list::find\_slm(int id){

if (!slm\_list.empty()){

iter= slm\_list.begin();

while(iter!= slm\_list.end()){

if ((\*iter)->get\_id() == id){

return \*iter;

}

\*iter++;

}

}

}

void Salesman\_list::put\_list\_to\_file(char \*path){

if (!slm\_list.empty()){

ofstream fout;

fout.open(path);

iter= slm\_list.begin();

while(iter!= slm\_list.end()){

fout<<(\*iter)->get\_name()<<endl;

fout<<(\*iter)->get\_id()<<endl;

\*iter++;

}

fout.close();

}

}

void Salesman\_list::get\_list\_from\_file(char \*path){

ifstream fin;

string str, nm;

int id;

fin.open(path);

if(fin.is\_open()){

while(!fin.eof()){

stringstream ss;

str="";

fin>>str;

if (str!=""){

nm=str;

}

str="";

fin>>str;

if (str!=""){

ss<<str;

id=0;

ss>>id;

Salesman\* tmp\_slm = new Salesman(nm,id,storage,order\_list);

slm\_list.push\_back(tmp\_slm);

}

}

}

fin.close();

}

bool Salesman\_list::in\_list(int id)

{

if (!slm\_list.empty()){

iter= slm\_list.begin();

while(iter!= slm\_list.end()){

if ((\*iter)->get\_id() == id){

return true;

}

\*iter++;

}

return false;

}

}

Storage. Cpp

#include "Storage.h"

Storage::~Storage()

{

while (!flower\_vect.empty()){

iter=flower\_vect.begin();

delete \*iter;

flower\_vect.erase(iter);

}

}

void Storage::add\_flower(Flower\* temp\_flw) {

if (in\_vect(temp\_flw->get\_name())){

iter= flower\_vect.begin();

while(iter!= flower\_vect.end()){

if ((\*iter)->get\_name() == temp\_flw->get\_name()){

int tmp=temp\_flw->get\_quantity()+(\*iter)->get\_quantity();

cout<<tmp;

(\*iter)->set\_quantity(tmp);

if (temp\_flw->get\_price()>(\*iter)->get\_price()){

(\*iter)->set\_price(temp\_flw->get\_price());

}

break;

}

\*iter++;

}

}

else{

flower\_vect.push\_back(temp\_flw);

}

}

void Storage::get\_flower\_list() {

if (flower\_vect.empty()){

cout<<"list is empty"<<endl;

}

else{

cout<<"Storage"<<endl;

cout<<"=================================================="<<endl;

cout<<"flower name";

cout<<"\t||\t";

cout<<"quantity";

cout<<"\t||";

cout<<"price";

cout<<"\t||"<<endl;

cout<<"=================================================="<<endl;

iter= flower\_vect.begin();

while(iter!= flower\_vect.end()){

cout<<(\*iter)->get\_name();

cout<<"\t||\t";

cout<<(\*iter)->get\_quantity();

cout<<"\t||";

cout<<(\*iter)->get\_price();

cout<<"\t||"<<endl;

\*iter++;

}

cout<<"=================================================="<<endl;

}

}

void Storage::get\_storage\_from\_file(char \*path)

{

ifstream fin;

string str,nm;

int count;

float price;

fin.open(path);

if(fin.is\_open()){

while(!fin.eof()){

stringstream ss;

stringstream ss1;

str="";

fin>>str;

if (str!=""){

nm=str;

}

str="";

fin>>str;

if (str!=""){

ss<<str;

count=0;

ss>>count;

}

str="";

fin>>str;

if (str!=""){

ss1<<str;

price=0;

ss1>>price;

Flower\* tmp\_flw = new Flower(nm,count,price);

add\_flower(tmp\_flw);

}

}

}

fin.close();

}

void Storage::put\_storage\_to\_file(char\* path)

{

if (!flower\_vect.empty()){

ofstream fout;

fout.open(path);

iter= flower\_vect.begin();

while(iter!= flower\_vect.end()){

fout<<(\*iter)->get\_name()<<endl;

fout<<(\*iter)->get\_quantity()<<endl;

fout<<(\*iter)->get\_price()<<endl;

\*iter++;

}

fout.close();

}

}

bool Storage::in\_vect(string nm)

{

if (!flower\_vect.empty()){

iter= flower\_vect.begin();

while(iter!= flower\_vect.end()){

if ((\*iter)->get\_name() == nm){

return true;

}

\*iter++;

}

return false;

}

}

Flower \*Storage::find\_flower(string nm)

{

if (!flower\_vect.empty()){

iter= flower\_vect.begin();

while(iter!= flower\_vect.end()){

if ((\*iter)->get\_name() == nm){

return \*iter;

}

\*iter++;

}

return NULL;

}

}

Storage \*Storage::choose\_flowers()

{

Storage\* shop\_list= new Storage;

int count;

Flower\* tmp\_flw;

if (!flower\_vect.empty()){

iter= flower\_vect.begin();

while (true){

cout<<(\*iter)->get\_name();

cout<<"\t||\t";

cout<<(\*iter)->get\_quantity();

cout<<"\t||";

cout<<((\*iter)->get\_price()\*1.35);

cout<<"\t||"<<endl;

cout << "Press 'a' to add flower to shoping list, \n"

<< "press 'n' to move to the next flower, \n"

<< "press 'p' to move to the previos flower, \n"

<< "press 'q' to end coosing flowers: \n";

char ch = cin.get();

cin.ignore(80, '\n');

switch(ch){

case'a':

do{

cout<<"enter number of flower client want to buy:";

cin>>count;

cin.ignore(80, '\n');

}while(count>(\*iter)->get\_quantity() || (count<0));

if (!shop\_list->in\_vect((\*iter)->get\_name())){

tmp\_flw = new Flower((\*iter)->get\_name(),count,(\*iter)->get\_price());

shop\_list->add\_flower(tmp\_flw);

}

else{

tmp\_flw=shop\_list->find\_flower((\*iter)->get\_name());

tmp\_flw->set\_quantity(count);

}

break;

case'n':

if(iter!= flower\_vect.end()-1)

\*iter++;

break;

case'p':

if(iter!= flower\_vect.begin())

\*iter--;

break;

case'q':

return shop\_list;

break;

default:

cout<<"wrong button. \n";

break;

}

}

}

return shop\_list;

}

float Storage::storage\_cost()

{

float sum = 0.f;

if (!flower\_vect.empty()){

iter= flower\_vect.begin();

while(iter!= flower\_vect.end()){

sum+=(\*iter)->get\_quantity()\*(\*iter)->get\_price();

\*iter++;

}

}

return sum;

}

void Storage::storage\_subtraction(Storage \*storage)

{

if (!flower\_vect.empty()){

int count;

iter= flower\_vect.begin();

while(iter!= flower\_vect.end()){

count=(-1)\*(\*iter)->get\_quantity();

(\*iter)->set\_quantity(count);

storage->add\_flower(\*iter);

\*iter++;

}

}

}

Summary. Cpp

#include "Summary.h"

void Summary::to\_sum\_up() {

cout<<"stock balance"<<endl;

storage->get\_flower\_list();

cout<<"Total profit: "<<order\_list->total\_profit()<<endl;

cout<<"General expenses: "<<order\_list->total\_cost()<<endl;

system("pause");

}

User. Cpp

#include "User.h"

int User::*get\_id*() {

return id;

}

string User::*get\_name*() {

return name;

}

void User::*set\_id*(int ID){

id=ID;

}

void User::*set\_name*(string nm){

name=nm;

}

**Списки атрибутов для классов**

1. Программа

* Список заказов

Необходим для хранения заказов в системе и их дальнейшего использования в ходе работы программы

* Список сотрудников

Хранит список объектов класса сотрудник, необходим для хранения данных о продавцах и проверки данных при авторизации.

* Владелец

Хранит объект класса владелец, необходим для хранения данных о владельце и проверки данных при авторизации.

* Склад цветов

Хранит массив объектов класса цветы, необходим для хранения данных о наличии цветов и их цветов

* Id авторизованного пользователя

Служит для того чтобы программа могла определить кто именно в данный момент пользуется программой

1. Пользователь

* ID

Идентификационный номер пользователя, необходим для авторизации пользователя(будь то владелец или продавец)

* Имя

Более человечный идентификатор, для облегчения понимания кто скрывается под идентификатором (более понятный для человека)

1. Цветок

* Название цветка

Хранит название цветка (при нахождении в списке или массиве, можно назвать первичным ключом)

* Количество цветка

Отображает количество цветков

* Цена цветка

Отображает количество цветков (если при приеме на склад цена будет иной, чем на складе, будет выбрана наибольшая)

1. Владелец

* Наследуются от пользователя
* Склад

Предоставляет доступ к классу склад (далее передается классу «продавец»)

* Список продавцов

Предоставляет доступ к классу список продавцов для создания и удаления продавцов

* Список заказов

Предоставляет доступ к классу список заказов (далее передается классу «продавец»)

* Отчет

Предоставляет доступ к классу отчет и просмотру небольшой отчет о деятельности цветочного магазина

1. Продавец

* Наследуются от пользователя
* Склад

Предоставляет доступ к классу склад. Используется для добавления цветов на склад, а также их изъятие

* Список заказов

Предоставляет доступ к классу список заказов. Используется для формирования заказов

1. Заказ

* Дата

Дата формирования заказа

* Id продавца

Идентификатор продавца проводившего заказ, используется для вычисления заработной платы сотрудника

* Нужна ли упаковка

Показывает нужна ли упаковка заказа, влияет на конечную цену

* Нужна ли доставка

Показывает нужна ли доставка заказа, влияет на конечную цену

* Список цветов

Список купленных клиентом цветов

* Конечная цена

Итоговая цена заказа

* Доход

Доход за отдельный заказ

1. Список заказов

* Список заказов

Хранит в себе список заказов

* Итератор

Помогает передвигаться по списку

1. Склад

* Вектор цветов

Хранит в себе цветы

* Итератор

Помогает передвигаться по вектору

1. Список продавцов

* Список продавцов

Хранит в себе список продавцов

* Итератор

Помогает передвигаться по списку

* Список заказов

Предоставляет доступ к классу список заказов(далее передается классу «продавец»)

* Склад

Предоставляет доступ к классу склад (далее передается классу «продавец»)

1. Отчет

* Склад

Предоставляет доступ к классу склад. Необходим для формирования отчета

* Список заказов

Предоставляет доступ к классу список заказов. Необходим для формирования отчета

**Список сообщений**

* Создать продавца

Владелец нанимает нового продавца и добавляет его в систему

* Удалить продавца

Владелец увольняет продавца и удаляет его из системы

* Получить список продавцов

Владелец просматривает список продавцов

* Сформировать заказ

Продавец формирует заказ для клиента. Информация о заказе сохраняется в системе и купленные цветы убираются со склада

* Добавить цветок на склад

Продавец добавляет цветы на склад в связи с привозом

* Получить список со склада

Вывести остатки цветов со склада в виде таблицы(далее будет применятся в формировании отчет)

* Авторизовать пользователя

Выполнить вход в систему для получения доступа к соответствующему функционалу

* Вывести отчет

Вывести отчет о деятельности цветочного магазина

* Взаимодействовать с программой как продавец

Позволяет осуществить доступ авторизованному продавцу к необходимому в его работе функционалу

* Взаимодействовать с программой как владелец

Позволяет осуществить доступ авторизованному владельцу к необходимому в его работе функционалу

**Диаграмма классов**

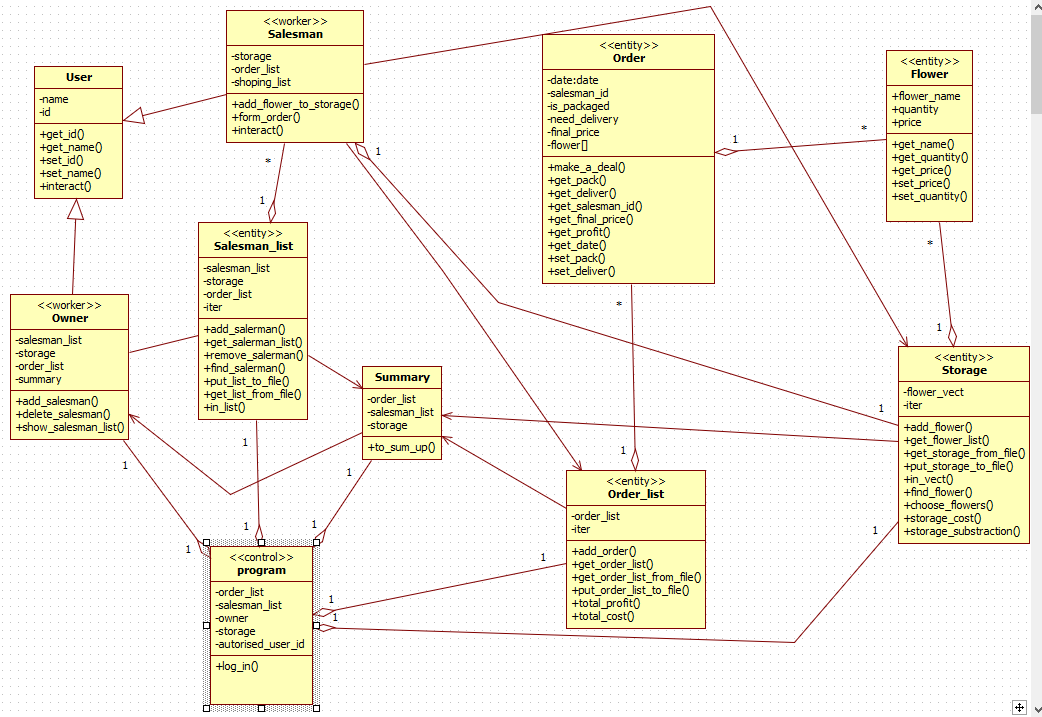
****

Рис5.диаграмма класса

**Диаграммы последовательности**

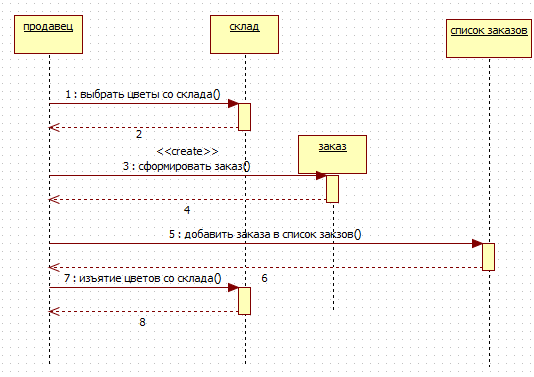
****

Рис6. Диаграмма последовательности «сформировать заказ»

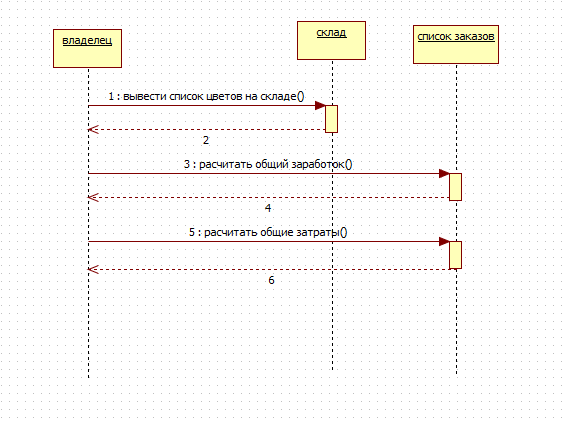


Рис7. Диаграмма последовательности «составить отчет»

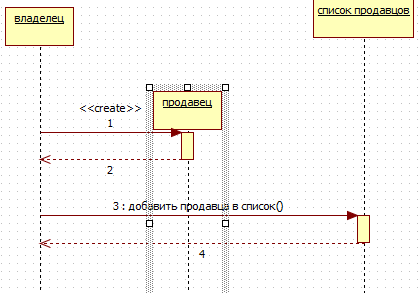


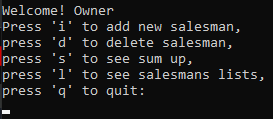
Рис8. Диаграмма последовательности «добавить продавца»

**Руководство пользователя**

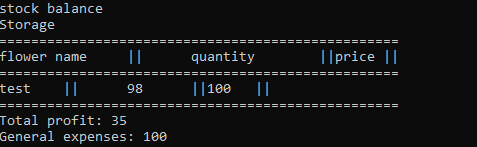
1. Запустите программу
2. Введите свой id(для владельца = 0, продавцы-остальные)

Рис9. Ввод id

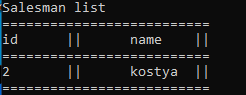
1. Если вы ввели 0 то программа выдаст на выбор 5 опций

рис10. Меню владельца

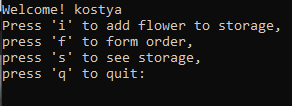
1. Если нажать i то вы перейдете в меню добавления продавцов
2. Когда программа выдаст «Enter name:» введите имя нового продавца
3. Когда программа выдаст «Enter ID» введите id нового сотрудника
4. Если выбранный вами ID уже занят, программа предложит ввести другой ID
5. После успешного ввода продавца вы вернетесь в п.3
6. Если нажать d, то вы перейдете в меню удаления продавца
7. Когда программа выдаст «Enter ID» введите ID продавца которого хотите удалить
8. Если вы ввели несуществующий ID продавца, программа предложит ввести другой
9. Если вы ввели ID верно программа удалит нужного вам продавца и вернет вас в п.3
10. Если нажать s, то программа выведет небольшой отчет о деятельности цветочного магазина(список цветов на складе, общий заработок, общие затраты)

рис11. Вид отчета

1. Ознакомившись с отчетом нажмите любую клавишу, чтобы вернутся в п.3
2. Если нажать l, то программа выведет вам список продавцов, и вернет вас в п.3

рис12. Вид списка продавцов

1. Чтобы выйти из программы нажмите q
2. Если вы зашли как продавец то программа выдаст на выбор 4 опций

Рис13. Меню продавца

1. Если нажать i то вы перейдете в меню добавления цветов на склад
2. Когда программа выдаст «enter flower name:», введите название цветка
3. Когда программа выдаст «enter flower count:», введите количество цветов
4. Когда программа выдаст «enter each flower price:», введите цену поставки 1 цветка
5. После этого программа добавит цветы на склад и вернет вас в п17
6. Если нажать f, то вы перейдете в меню формирования заказа, состоящая из записи об одном цветке и 4 опций

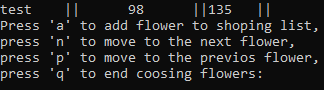


Рис14. Меню выбора цветов в заказ(начало формирования заказа)

1. Чтобы добавить текущий цветок в корзину, нужно нажать I, и далее ввести необходимое количество
2. Для перемещения по списку цветов используются кнопки n и p, следующий и предыдущий соответственно
3. Для завершения выбора цветов нажмите q
4. Далее программа предложит вам упаковку заказа нажмите y, если она нужна и n, если не нужна
5. Далее программа предложит вам доставку заказа нажмите y, если она нужна и n, если не нужна
6. Далее программа выдаст окончательную стоимость заказа
7. Если вы согласны с ценой нажмите y, если нет – n

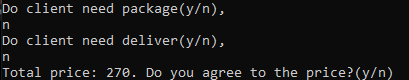


Рис15. Продолжение формирования заказа

1. После этого вас вернет в п17
2. Если нажать s то программа выдаст список цветов на складе
3. Чтобы выйти из программы нажмите q

**История проекта на GitHub.com**(git clone https://github.com/kkksas/command\_project.git)

1. Add .h and .cpp files (Иванов)

Добавлены пустые заголовочные и исходные файлы(не все, тк приходилось добавлять)

1. Released salerman list and user interface plug (Иванов)

Добавлены классы storage, salesman\_list и order\_list. Сделаны интерфейсные заглушки для меню owner и salesman

1. Add and delete salesman user interface (Сергиенко)

Добавлен ввод для добавления и удаления продавца (класс Owner)

1. add add\_salesman, delete\_salesman, show\_salesman\_list functions (Сергиенко)

добавлен вывод списка продавцов, доделаны добавление удаление продавцов

5.merge branch “feature\_owner” into develop(Иванов)

Слияние веток

6.Flower and storage classes is done (Иванов)

Реализованы классы flower и storage

7.add Summary.h and Summary.cpp files(Иванов)

Добавлен новый класс Summary(пустой)

8.class order and order\_list is done(Иванов)

Реализованы классы order и order\_list

9.preparations for employee class realization (перепутал salesman) (Иванов)

Небольшие исправления в salesman

10. add salesman class method add\_flower\_to\_storage() (Переверзева)

Полностью сделан метод

1. add find\_flower(), choose\_flowers(), storage\_cost(), storage\_substraction() functions to storage class and form\_order() to salesman class(Переверзева)

добавлена функция формирования заказа form\_order(), а также добавлены функции необходимые для его реализации в хранилище choose\_flowers()-формирование корзины цветов, storage\_cost – расчет цены корзины, storage\_substraction – вычитание купленного количества цветов со склада, find\_flower()- поиск цветка по названию

1. merge branch ‘feature\_salesman’ into develop(Переверзева)

слияние веток

1. summary class is done (Иванов)

реализован класс Summary (Иванов)

1. add destructors to storage, order\_list, salesman\_list

добавил деструкторы в хранилища

1. merge branch ‘develop’ (Иванов)

слияние с мастер-веткой

1. last commit (Иванов)

добавлен этот отчет

**Заключение**

В ходе выполнения командного задания использовалась среда разработки qt язык c++.

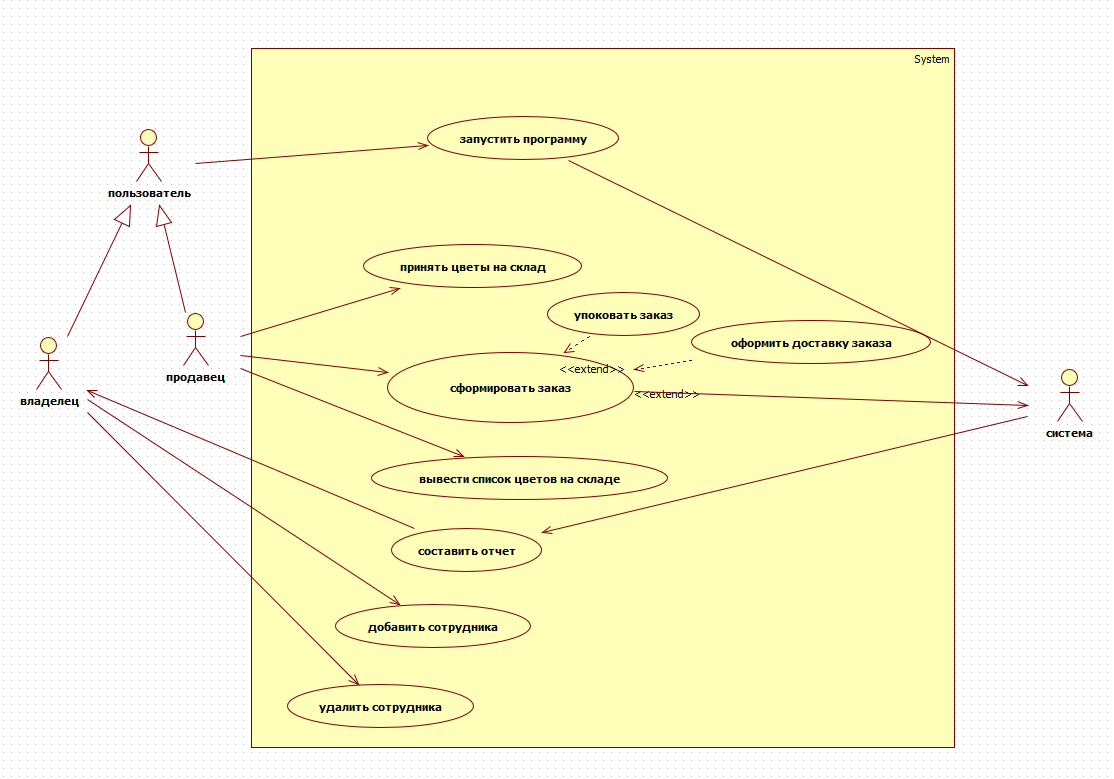
Применялась система контроля версий git. Реализованы все прецеденты кроме упаковки и доставки заказа (в программе просто добавляется определенная сумма к заказу). В последней версии программы зависания или сбои не были обнаружены. Для классов хранилищ(storage, salesman\_list и order\_list) была реализована очистка памяти в их деструкторах. Неиспользуемых переменные в ходе разработки старались удалять. В алгоритмы методов старались не добавлять не нужных циклов, массивов и др. К отчету приложены диаграмма вариантов использования, диаграммы деятельности, диаграммы последовательности и диаграмма классов. Все они, за исключением диаграммы вариантов использования, находятся в пошаговом описании процесса разработки. Мы считаем что наша команда достигла поставленных целей.

Рис16.Диаграмма вариантов использования