Задание

Реализуйте с использованием паттернов проектирования про- стейшую систему планирования задач. Должна быть возмож- ность создания списка дел, установки приоритетов, установки дат выполнения, удаление и изменения дел.

Каждому делу можно установить тег. Список дел можно загру- жать и сохранять в файл. Необходимо реализовать возможность поиска конкретного дела. Критерии поиска: по датам, по тегам, по приоритету и так далее.

#include<iostream>

#include <Windows.h>

#include <list>

#include <fstream>

using namespace std;

class Date

{

int day;

int mis;

int year;

public:

Date()

{

}

Date(int day, int mis, int year)

{

this->day = day;

this->mis = mis;

this->year = year;

}

int& Get\_Day()

{

return day;

}

int& Get\_Mis()

{

return mis;

}

int& Get\_Year()

{

return year;

}

bool operator==(const Date& obj)

{

return day == obj.day && mis == obj.mis && year == obj.year;

}

friend ostream& operator<<(ostream& out, const Date& w);

friend istream& operator>>(istream& in, Date& obj);

};

ostream& operator<<(ostream& out, const Date& w)

{

out << w.day << "." << w.mis << "." << w.year;

return out;

}

istream& operator>>(istream& in, Date& obj)

{

cout << "Введіть число->";

in >> obj.day;

cout << "Введіть місяць->";

in >> obj.mis;

cout << "Введіть рік->";

in >> obj.year;

return in;

}

class Work

{

string name;

int priority;

Date date;

string tag;

public:

Work()

{

}

int Get\_Priority()

{

return priority;

}

string Get\_Tag()

{

return tag;

}

void Set\_Priority(int p)

{

priority = p;

}

string GetName()

{

return name;

}

Date& GetDate()

{

return date;

}

void SetWork(string name, int priority, Date date, string tag = "#")

{

this->name = name;

this->priority = priority;

this->date = date;

this->tag = "#";

if (tag != "#")

{

this->tag += tag;

if (this->tag[0] == '#' && this->tag[1] == '#')

this->tag.erase(this->tag.begin());

}

}

Work(string name, int priority, Date date, string tag="#")

{

this->name = name;

this->priority = priority;

this->date = date;

this->tag = "#";

if (tag != "#")

{

this->tag += tag;

if (this->tag[0] == '#' && this->tag[1] == '#')

this->tag.erase(this->tag.begin());

}

}

friend ostream& operator<<(ostream& out, const Work& w);

};

ostream& operator<<(ostream& out, const Work& w)

{

cout << "------------------------------------\n";

out << w.name << endl;

out << w.priority << endl;

cout << w.date << endl;

out << w.tag << endl;

return out;

}

class ListWorks

{

list<Work\*> listworks;

public:

list<Work\*>& GetLst()

{

return listworks;

}

~ListWorks()

{

listworks.clear();

}

void AddWork(Work\* w)

{

this->listworks.push\_back(w);

}

void ShowAll()

{

for (auto it = this->listworks.begin(); it != this->listworks.end(); it++)

{

cout << \*(\*it);

}

}

void DeleteWork(string name, Date date)

{

for (auto it = this->listworks.begin(); it != this->listworks.end(); it++)

{

if ((\*it)->GetName() == name && (\*it)->GetDate()==date)

{

this->listworks.erase(it);

break;

}

}

}

void OpenList()

{

fstream f("listwork.txt", ios::in);

string name,tag;

Date d;

int priority;

while (!f.eof())

{

f >> name;

f >> d.Get\_Day();

f >> d.Get\_Mis();

f >> d.Get\_Year();

f >> priority;

f >> tag;

listworks.push\_back(new Work(name,priority,d,tag));

}

f.close();

}

void SaveList()

{

fstream f("listwork.txt", ios::out);

for (auto it = this->listworks.begin(); it != this->listworks.end(); it++)

{

f << endl << (\*it)->GetName() << " " << (\*it)->GetDate().Get\_Day()<<" "<<(\*it)->GetDate().Get\_Mis()<<" "<<(\*it)->GetDate().Get\_Year()<<" "<<(\*it)->Get\_Priority()<<" "<<(\*it)->Get\_Tag();

}

f.close();

}

void RedactWork(string name, Date date)

{

for (auto it = this->listworks.begin(); it != this->listworks.end(); it++)

{

if ((\*it)->GetName() == name && (\*it)->GetDate() == date)

{

cout << \*(\*it);

int k, p;

cout << "Змінити пріоритет->1, змінити дату->2, вихід->0\n";

cout << "->";

cin >> k;

while (k)

{

switch (k)

{

case 1:

cout << "->";

cin >> p;

(\*it)->Set\_Priority(p);

break;

case 2:

cin >> (\*it)->GetDate();

break;

}

cout << "Змінити пріоритет->1, змінити дату->2, вихід->0\n";

cout << "->";

cin >> k;

}

}

}

}

};

class AbstactBuilder

{

protected:

ListWorks\* lst;

public:

AbstactBuilder()

{

}

ListWorks\* GetLst()

{

return lst;

}

virtual void Search() = 0;

};

class BuilderSearchByTag:public AbstactBuilder

{

public:

BuilderSearchByTag(ListWorks\* lst)

{

this->lst = lst;

}

void Search()

{

string tag="#", tag2;

cout << "Введіть тег для пошуку->";

cin >> tag2;

tag += tag2;

for (auto it = GetLst()->GetLst().begin(); it != this->GetLst()->GetLst().end(); it++)

{

if ((\*it)->Get\_Tag() == tag)

{

cout << \*(\*it);

}

}

}

};

class BuilderSearchByName :public AbstactBuilder

{

public:

BuilderSearchByName(ListWorks\* lst)

{

this->lst = lst;

}

void Search()

{

string name;

cout << "Введіть назву справи для пошуку->";

cin >> name;

for (auto it = this->GetLst()->GetLst().begin(); it != this->GetLst()->GetLst().end(); it++)

{

if ((\*it)->GetName() == name)

{

cout << \*(\*it);

}

}

}

};

class BuilderSearchByDate :public AbstactBuilder

{

public:

BuilderSearchByDate(ListWorks\* lst)

{

this->lst = lst;

}

void Search()

{

Date d;

cin >> d;

for (auto it = this->GetLst()->GetLst().begin(); it != this->GetLst()->GetLst().end(); it++)

{

if ((\*it)->GetDate() == d)

{

cout << \*(\*it);

}

}

}

};

class BuilderSearchByPriority :public AbstactBuilder

{

public:

BuilderSearchByPriority(ListWorks\* lst)

{

this->lst = lst;

}

void Search()

{

int p;

cout << "Введіть пріоритет справи->";

cin >> p;

for (auto it = this->GetLst()->GetLst().begin(); it != this->GetLst()->GetLst().end(); it++)

{

if ((\*it)->Get\_Priority() == p)

{

cout << \*(\*it);

}

}

}

};

class ConstructorBuilder

{

public:

void Construct(AbstactBuilder\* abstactBuiler)

{

abstactBuiler->Search();

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string name, tag;

Date d;

int priority;

Work w;

ListWorks\* lst = new ListWorks;

ConstructorBuilder\* cb = new ConstructorBuilder();

AbstactBuilder\* b;

int k,k1;

cout << "Додати справу->1, видалити справу->2, редагувати справу->3, зчитати список справ->4, зберегти список справ->5, знайти справи(6-за назвою, 7-за тегом, 8-за датою, 9-за пріоритетом), 0-вихід\n";

cout << "->";

cin >> k;

while (k)

{

switch (k)

{

case 1:

cout << "Введіть назву справи->";

cin >> name;

cin >> d;

cout << "Введіть пріоритет справи(0-...)->";

cin >> priority;

cout << "Додати тег справи(1-так, 0-ні)->";

cin >> k1;

switch (k1)

{

case 0:

w.SetWork(name, priority, d);

lst->AddWork(&w);

break;

case 1:

cout << "Введіть тег справи->";

cin >> tag;

w.SetWork(name, priority, d, tag);

lst->AddWork(&w);

break;

}

break;

case 2:

cout << "Введіть назву справи->";

cin >> name;

cin >> d;

lst->DeleteWork(name, d);

break;

case 3:

cout << "Введіть назву справи->";

cin >> name;

cin >> d;

lst->RedactWork(name, d);

break;

case 4:

lst->OpenList();

break;

case 5:

lst->SaveList();

break;

case 6:

cout << "Пошук...\n";

b = new BuilderSearchByName(lst);

cb->Construct(b);

delete b;

break;

case 7:

cout << "Пошук...\n";

b = new BuilderSearchByTag(lst);

cb->Construct(b);

delete b;

break;

case 8:

cout << "Пошук...\n";

b = new BuilderSearchByDate(lst);

cb->Construct(b);

delete b;

break;

case 9:

cout << "Пошук...\n";

b = new BuilderSearchByPriority(lst);

cb->Construct(b);

delete b;

break;

}

cout << "Додати справу->1, видалити справу->2, редагувати справу->3, зчитати список справ->4, зберегти список справ->5, знайти справи(6-за назвою, 7-за тегом, 8-за датою, 9-за пріоритетом), 0-вихід\n";

cout << "->";

cin >> k;

}

delete lst;

system("pause");

return 0;

}