Пример 1. Использование паттерна “Подписчик-издатель” (Qt).

# include <iostream>

# include <string>

# include <QObject>

using namespace std;

class Employer : public QObject // Publisher

{

Q\_OBJECT

private:

const int day\_salary = 15;

const int day\_advance = 25;

public:

Employer() = default;

void notifyEmployeis(int day);

signals:

void salaryPayment();

void taskIssuance();

};

# pragma region Method Publisher

void Employer::notifyEmployeis(int day)

{

if (day == day\_salary || day == day\_advance)

{

emit salaryPayment();

}

else

{

emit taskIssuance();

}

}

# pragma endregion

enum class Mood

{

happy,

sad

};

class Employee : public QObject // Subscribe

{

Q\_OBJECT

private:

string name;

Mood mood;

public:

Employee(string name) : name(name), mood(Mood::sad) {}

string getName() { return name; }

Mood getMood() { return mood; }

void subscribeOnEmployer(const Employer\* ptrEmployer);

void unsubscribeFromEmployer(const Employer\* ptrEmployer);

public slots:

void onSalaryPayment() { mood = Mood::happy; }

void onTaskIssuance() { mood = Mood::sad; }

};

# pragma region Methods Subscribe

void Employee::subscribeOnEmployer(const Employer\* ptrEmployer)

{

QObject::connect(ptrEmployer, SIGNAL(salaryPayment()), this, SLOT(onSalaryPayment()));

QObject::connect(ptrEmployer, SIGNAL(taskIssuance()), this, SLOT(onTaskIssuance()));

}

void Employee::unsubscribeFromEmployer(const Employer\* ptrEmployer)

{

QObject::disconnect(ptrEmployer, SIGNAL(salaryPayment()), this, SLOT(onSalaryPayment()));

QObject::disconnect(ptrEmployer, SIGNAL(taskIssuance()), this, SLOT(onTaskIssuance()));

}

# pragma endregion

ostream& operator<< (ostream &out, const Employee &employee)

{

string mood = employee.getMood() == Mood::happy ?

"Happy with a lot of money!!!!))))" :

"Sad with a lot of work....((((";

return out << "Name: " << employee.getName() << ", Mood: " << mood << endl;

}

int main()

{

//Создадим работодателя, который выступает в роли объекта publisher

Employer google;

//Создадим несколько работников, которые выступают в роли объекта subscriber

Employee vanya("Vanya");

Employee artem("Artem");

Employee maxim("Maxim");

Employee dmitrii("Dmitrii");

//Проверим текущее состояние работников

cout << "Employees states after creation:" << endl;

cout << vanya;

cout << artem;

cout << maxim;

cout << dmitrii << endl;

//Подпишем нескольких работников на объект работодателя

vanya.subscribeOnEmployer(&google);

artem.subscribeOnEmployer(&google);

dmitrii.subscribeOnEmployer(&google);

//Проверим состояние после подписки на объект работодателя

cout << "Employees states after publishing:" << endl;

cout << vanya;

cout << artem;

cout << maxim;

cout << dmitrii << endl;

//Вызовем метод получения оповещения для всех подписчиков, в котором вызовется сигнал salaryPayment

int day\_salary = 25;

google.notifyEmployeis(day\_salary);

//Проверим состояние работников, после получения зарплаты

cout << "Employees states after salary event:" << endl;

cout << vanya;

cout << artem;

cout << maxim;

cout << dmitrii << endl;

//Вызовем метод получения оповещения для всех подписчиков, в котором вызовется сигнал taskIssuance

int someDay = 10;

google.notifyEmployeis(someDay);

//Проверим состояние работников, после получения задания

cout << "Employees states after task event:" << endl;

cout << vanya;

cout << artem;

cout << maxim;

cout << dmitrii << endl;

return 0;

}

Пример 2. Использование паттерна “Подписчик-издатель” (C++ CLR).

using namespace System;

delegate void Eventhandler();

public ref class Employer // Publisher

{

private:

const int day\_salary = 15;

const int day\_advance = 25;

public:

void NotifyEmployeis(int day);

event Eventhandler^ onSalaryPayment;

event Eventhandler^ onTaskIssuance;

};

# pragma region Method Publisher

void Employer::NotifyEmployeis(int day)

{

if (day == day\_salary || day == day\_advance)

{

onSalaryPayment();

}

else

{

onTaskIssuance();

}

}

# pragma endregion

enum class Mood

{

happy,

sad

};

public ref class Employee abstract // Subscribe

{

protected:

String^ name;

Mood mood;

public:

Employee(String^ nm) : name(nm), mood(Mood::sad) {}

String^ GetName() { return name; }

Mood GetMood() { return mood; }

virtual void OnSalaryPayment() { mood = Mood::happy; }

virtual void OnTaskIssuance() { mood = Mood::sad; }

void SubscribeOnEmployer(Employer^ ptrEmployer);

void UnsubscribeFromEmployer(Employer^ ptrEmployer);

};

public ref class Coder : public Employee

{

public:

Coder(String^ name) : Employee(name) {}

};

# pragma region Methods Subscribe

void Employee::SubscribeOnEmployer(Employer^ ptrEmployer)

{

ptrEmployer->onSalaryPayment += gcnew Eventhandler(this, &Employee::OnSalaryPayment);

ptrEmployer->onTaskIssuance += gcnew Eventhandler(this, &Employee::OnTaskIssuance);

}

void Employee::UnsubscribeFromEmployer(Employer^ ptrEmployer)

{

ptrEmployer->onSalaryPayment -= gcnew Eventhandler(this, &Employee::OnSalaryPayment);

ptrEmployer->onTaskIssuance -= gcnew Eventhandler(this, &Employee::OnTaskIssuance);

}

# pragma endregion

public value class Print

{

public:

static void WriteLine(Employee^ employee)

{

String^ mood = gcnew String(employee->GetMood() == Mood::happy ?

L"Happy with a lot of money!!!!))))" :

L"Sad with a lot of work....((((");

Console::WriteLine(L"Name: " + employee->GetName() + L", Mood: " + mood);

}

};

int main()

{

//Создадим работодателя, который выступает в роли объекта publisher

Employer^ google = gcnew Employer();

//Создадим несколько работников, которые выступают в роли объекта subscriber

Employee^ vanya = gcnew Coder("Vanya");

Employee^ artem = gcnew Coder("Artem");

Employee^ maxim = gcnew Coder("Maxim");

Employee^ dmitrii = gcnew Coder("Dmitrii");

//Проверим текущее состояние работников

Console::WriteLine(L"Employees states after creation:");

Print::WriteLine(vanya);

Print::WriteLine(artem);

Print::WriteLine(maxim);

Print::WriteLine(dmitrii);

Console::WriteLine();

//Подпишем нескольких работников на объект работодателя

vanya->SubscribeOnEmployer(google);

artem->SubscribeOnEmployer(google);

dmitrii->SubscribeOnEmployer(google);

//Проверим состояние после подписки на объект работодателя

Console::WriteLine(L"Employees states after publishing:");

Print::WriteLine(vanya);

Print::WriteLine(artem);

Print::WriteLine(maxim);

Print::WriteLine(dmitrii);

Console::WriteLine();

//Вызовем метод получения оповещения для всех подписчиков, в котором вызовется сигнал salaryPayment

int day\_salary = 25;

google->NotifyEmployeis(day\_salary);

//Проверим состояние работников, после получения зарплаты

Console::WriteLine(L"Employees states after salary event:");

Print::WriteLine(vanya);

Print::WriteLine(artem);

Print::WriteLine(maxim);

Print::WriteLine(dmitrii);

Console::WriteLine();

//Вызовем метод получения оповещения для всех подписчиков, в котором вызовется сигнал taskIssuance

int someDay = 10;

google->NotifyEmployeis(someDay);

//Проверим состояние работников, после получения задания

Console::WriteLine(L"Employees states after task event:");

Print::WriteLine(vanya);

Print::WriteLine(artem);

Print::WriteLine(maxim);

Print::WriteLine(dmitrii);

Console::WriteLine();

return 0;

}

Пример 3. Использование паттерна “Подписчик-издатель” (C++ CLR).

using namespace System;

delegate void Eventhandler(int a);

public ref class Manager

{

public:

event Eventhandler^ onHandler;

void Method()

{

onHandler(10);

}

};

public ref class Watcher

{

public:

Watcher(Manager^ manager)

{

manager->onHandler += gcnew Eventhandler(this, &Watcher::Handler);

}

void Handler(int a)

{

Console::WriteLine(L"method Handler!");

}

};

int main()

{

Manager^ manager = gcnew Manager();

Watcher^ watcher = gcnew Watcher(manager);

manager->Method();

return 0;

}