

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Новосибирский государственный университет экономики и управления «НИНХ»**

**(ФГБОУ ВО «НГУЭУ», НГУЭУ)**

Кафедра базовой подготовки

**Расчетно-графическая работа**

Дисциплина: Языки программирования

Ф.И.О студента: Караваев Данил Евгеньевич

Направление: 02.03.02 Фундаментальная информатика и информационные технологии

Специализация: Программная инженерия

Номер группы: ФИ202

Номер зачетной книжки: 221282

Проверил: Ковригин Алексей Викторович, канд. педагогических наук,

преподаватель

Новосибирск 2024

**Оглавление**

[1. Описание задачи 3](#_Toc155670202)

[2. Структура проекта 3](#_Toc155670203)

[3. Код программы 4](#_Toc155670204)

# **Описание задачи**

**Управление Заказами на Производстве**

Цель проекта состоит в создании программы для управления заказами на производстве с использованием объектно-ориентированного программирования (ООП) и реализации различных паттернов (decorator, controller, date). Программа должна позволять пользователю взаимодействовать с данными о заказах, выполнять операции добавления, просмотра, редактирования и удаления заказов. Данные должны храниться в структурированном XML файле, и программа должна поддерживать сохранение и загрузку данных из файла между запусками.

# **Структура проекта**

1. main.cpp: Основной файл программы, который содержит функцию main. В этом файле создается экземпляр модели, представления и контроллера.
2. Record.h: Заголовочный файл, который содержит определение класса Record. Класс Record представляет запись о заказе и содержит информацию о дате, заказчике (класс People), и заказе (класс Order).
3. People.h: Заголовочный файл, который содержит определение класса People. Класс People представляет информацию о заказчике, включая имя и возраст.
4. Order.h: Заголовочный файл, который содержит определение класса Order. Класс Order представляет информацию о заказе, включая идентификатор заказа, продукт и количество.
5. View.h: Заголовочный файл, который содержит определение интерфейса ViewInterface и его конкретной реализации View. Интерфейс ViewInterface определяет методы для отображения записей и меню. Класс View реализует эти методы.
6. Model.h: Заголовочный файл, который содержит определение интерфейса ModelInterface и его конкретной реализации Model. Интерфейс ModelInterface определяет методы для управления данными о заказах. Класс Model реализует эти методы и хранит данные о заказах.
7. Controller.h: Заголовочный файл, который содержит определение интерфейса ControllerInterface и его конкретной реализации Controller. Интерфейс ControllerInterface определяет метод для обработки ввода пользователя. Класс Controller реализует этот метод и обеспечивает взаимодействие между моделью и представлением.
8. multiplTree.h: Заголовочный файл, который содержит определение интерфейса MultiplTree и его конкретной реализации. Этот файл отвечает за работу с множеством узлов дерева, предоставляя методы для добавления, удаления и поиска узлов в дереве.
9. rec.xml: xml файл, в котором хранятся данные о заказах между запусками программы.

**Общий ход выполнения программы:**

1. Создание записей о заказах.
2. Отображение записей о заказах.
3. Возможность удаления выбранных записей.
4. Сохранение и загрузка данных из XML файла.
5. Взаимодействие с пользователем через простое текстовое меню.

Каждый из вышеупомянутых компонентов проекта взаимодействует между собой с помощью паттерна MVC (Model-View-Controller), что обеспечивает четкое разделение ответственностей и легкость расширения функциональности.

# **Код программы**

class Record {

public:

Record(const std::string& date, const std::string& customer, const Order& order);

std::string getDate() const;

std::string getCustomer() const;

const Order& getOrder() const;

private:

std::string date;

std::string customer;

Order order;

};

Record::Record(const std::string& date, const std::string& customer, const Order& order)

: date(date), customer(customer), order(order) {}

std::string Record::getDate() const {

return date;

}

std::string Record::getCustomer() const {

return customer;

}

const Order& Record::getOrder() const {

return order;

}

class People {

public:

People(const std::string& name, const std::string& role);

std::string getName() const;

std::string getRole() const;

private:

std::string name;

std::string role;

};

People::People(const std::string& name, const std::string& role)

: name(name), role(role) {}

std::string People::getName() const {

return name;

}

std::string People::getRole() const {

return role;

}

class Order {

public:

Order(int orderId, const std::string& product, int quantity);

int getOrderId() const;

std::string getProduct() const;

int getQuantity() const;

private:

int orderId;

std::string product;

int quantity;

};

Order::Order(int orderId, const std::string& product, int quantity)

: orderId(orderId), product(product), quantity(quantity) {}

int Order::getOrderId() const {

return orderId;

}

std::string Order::getProduct() const {

return product;

}

int Order::getQuantity() const {

return quantity;

}

class MultiplTree {

public:

MultiplTree(const std::string& name = "", const std::string& value = "")

: name(name), value(value) {}

void setName(const std::string& name) { this->name = name; }

std::string getName() const { return name; }

void setValue(const std::string& value) { this->value = value; }

void setValue(double value) { this->value = std::to\_string(value); }

void setValue(int value) { this->value = std::to\_string(value); }

std::string getValue() const { return value; }

bool isInt() const {

try {

std::stoi(value);

return true;

} catch (...) {

return false;

}

}

bool isDouble() const {

try {

std::stod(value);

return true;

} catch (...) {

return false;

}

}

int getInt() const {

try {

return std::stoi(value);

} catch (...) {

return 0;

}

}

double getDouble() const {

try {

return std::stod(value);

} catch (...) {

return 0.0;

}

}

size\_t getCountSub() const { return children.size(); }

std::shared\_ptr<MultiplTree> getSub(size\_t i) const {

if (i < children.size()) {

return children[i];

}

return nullptr;

}

std::shared\_ptr<MultiplTree> getSub(const std::string& name) const {

for (const auto& child : children) {

if (child->getName() == name) {

return child;

}

}

return nullptr;

}

void addSub(const std::shared\_ptr<MultiplTree>& subNode) {

children.push\_back(subNode);

}

void removeSub(const std::string& name) {

auto it = std::remove\_if(children.begin(), children.end(),

[&](const std::shared\_ptr<MultiplTree>& child) {

return child->getName() == name;

});

children.erase(it, children.end());

}

std::shared\_ptr<MultiplTree> findNode(const std::string& name) {

if (this->name == name) {

return std::shared\_ptr<MultiplTree>(this);

}

for (const auto& child : children) {

auto found = child->findNode(name);

if (found != nullptr) {

return found;

}

}

return nullptr;

}

private:

std::string name;

std::string value;

std::vector<std::shared\_ptr<MultiplTree>> children;

};

class Model {

public:

void addRecord(const Record& record);

void saveToFile() const;

void loadFromFile();

void removeRecord(const std::string& customer);

const MultiplTree& getRecords() const;

private:

MultiplTree records;

};

void Model::addRecord(const Record& record) {

auto customerNode = std::make\_shared<MultiplTree>(record.getCustomer());

auto dateNode = std::make\_shared<MultiplTree>("Date");

dateNode->setValue(record.getDate());

auto productNode = std::make\_shared<MultiplTree>("Product");

productNode->setValue(record.getOrder().getProduct());

auto quantityNode = std::make\_shared<MultiplTree>("Quantity");

quantityNode->setValue(record.getOrder().getQuantity());

customerNode->addSub(dateNode);

customerNode->addSub(productNode);

customerNode->addSub(quantityNode);

records.addSub(customerNode);

}

void Model::saveToFile() const {

std::ofstream file("records.xml");

if (file.is\_open()) {

file << "<records>\n";

for (size\_t i = 0; i < records.getCountSub(); ++i) {

auto customerNode = records.getSub(i);

file << " <record>\n";

file << " <customer>" << customerNode->getName() << "</customer>\n";

for (size\_t j = 0; j < customerNode->getCountSub(); ++j) {

auto detailNode = customerNode->getSub(j);

file << " <" << detailNode->getName() << ">" << detailNode->getValue() << "</" << detailNode->getName() << ">\n";

}

file << " </record>\n";

}

file << "</records>\n";

file.close();

std::cout << "Заказы сохранены в файл\n";

} else {

std::cout << "Ошибка: Невозможно сохранить заказы\n";

}

}

void Model::loadFromFile() {

std::ifstream file("records.xml");

if (file.is\_open()) {

records = MultiplTree("Root"); //Очистить текущие записи

std::string line, tag;

while (std::getline(file, line)) {

std::istringstream iss(line);

iss >> tag;

if (tag == "<record>") {

std::string date, customer, product;

int quantity;

std::getline(file, line);

customer = line.substr(line.find(">") + 1);

customer = customer.substr(0, customer.find("<"));

std::getline(file, line);

date = line.substr(line.find(">") + 1);

date = date.substr(0, date.find("<"));

std::getline(file, line);

product = line.substr(line.find(">") + 1);

product = product.substr(0, product.find("<"));

std::getline(file, line);

std::string quantityStr = line.substr(line.find(">") + 1);

quantityStr = quantityStr.substr(0, quantityStr.find("<"));

quantity = std::stoi(quantityStr);

Order order(0, product, quantity);

Record record(date, customer, order);

addRecord(record);

}

}

file.close();

std::cout << "Заказы загружены из файла\n";

} else {

std::cout << "Ошибка: Невозможно загрузить заказы\n";

}

}

void Model::removeRecord(const std::string& customer) {

records.removeSub(customer);

}

const MultiplTree& Model::getRecords() const {

return records;

}

class Controller {

public:

Controller(Model& model, View& view);

void displayRecords() const;

void addRecord();

void saveRecords() const;

void loadRecords();

void deleteRecord();

private:

Model& model;

View& view;

};

Controller::Controller(Model& model, View& view) : model(model), view(view) {}

void Controller::displayRecords() const {

view.displayRecords(model.getRecords());

}

void Controller::addRecord() {

std::string date, customer, product;

int quantity;

std::cout << "Введите дату (Год-Месяц-День): ";

std::cin >> date;

std::cout << "Введите имя заказчика: ";

std::cin >> customer;

std::cout << "Введите название продукта: ";

std::cin >> product;

std::cout << "Введите количество: ";

std::cin >> quantity;

Order order(0, product, quantity);

Record record(date, customer, order);

model.addRecord(record);

}

void Controller::saveRecords() const {

model.saveToFile();

}

void Controller::loadRecords() {

model.loadFromFile();

}

void Controller::deleteRecord() {

std::string customer;

std::cout << "Введите имя заказчика для удаления: ";

std::cin >> customer;

model.removeRecord(customer);

}

class View {

public:

void displayRecords(const MultiplTree& records) const;

void displayMenu() const;

};

void View::displayRecords(const MultiplTree& records) const {

std::cout << "Список заказов:\n";

for (size\_t i = 0; i < records.getCountSub(); ++i) {

auto customerNode = records.getSub(i);

if (customerNode) {

std::cout << "Заказчик: " << customerNode->getName() << "\n";

for (size\_t j = 0; j < customerNode->getCountSub(); ++j) {

auto detailNode = customerNode->getSub(j);

if (detailNode) {

std::cout << " " << detailNode->getName() << ": " << detailNode->getValue() << "\n";

}

}

std::cout << "-----------------------\n";

}

}

}

void View::displayMenu() const {

std::cout << "1. Вывести список заказов\n";

std::cout << "2. Добавить новый заказ\n";

std::cout << "3. Сохранить заказы\n";

std::cout << "4. Загрузить заказы\n";

std::cout << "5. Удалить заказ\n";

std::cout << "0. Выход\n";

}

int main() {

SetConsoleCP(CP\_UTF8);

SetConsoleOutputCP(CP\_UTF8);

Model model;

View view;

Controller controller(model, view);

int choice = -1;

while (choice != 0) {

view.displayMenu();

std::cout << "Выберите пункт меню: ";

std::cin >> choice;

switch (choice) {

case 1:

controller.displayRecords();

break;

case 2:

controller.addRecord();

break;

case 3:

controller.saveRecords();

break;

case 4:

controller.loadRecords();

break;

case 5:

controller.deleteRecord();

break;

case 0:

std::cout << "Выход\n";

break;

default:

std::cout << "Неверный выбор. Попробуйте снова.\n";

break;

}

}

return 0;

}