**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

**СРС**

**по дисциплине «Объектно-ориентированное программирование»**

Выполнил: студент группы ПИ-3-21

Джанышбекова Акмарал

Проверил: Мусабаев Э.Б.

**Бишкек 2024**

Оглавление

[**Задание 1. (17.41).** 3](#_Toc165340786)

[**Постановка задачи** 3](#_Toc165340787)

[**Цель** 3](#_Toc165340788)

[**Исходный код программы на С++** 3](#_Toc165340789)

[**Структура классов:** 7](#_Toc165340790)

[**Диаграмма Классов** 9](#_Toc165340791)

[**Вывод** 9](#_Toc165340792)

[**Задание 1. (17.18).** 10](#_Toc165340793)

[**Постановка задачи** 10](#_Toc165340794)

[**Цель** 10](#_Toc165340795)

[**Исходный код программы на С++** 10](#_Toc165340796)

[**Структура классов:** 13](#_Toc165340797)

[**Диаграмма Классов** 15](#_Toc165340798)

[**Вывод** 15](#_Toc165340799)

# **Задание 1. (17.41).**

## **Постановка задачи**

*Каталог запчастей автомобиля.*

В автомобиле насчитывается несколько тысяч деталей; некоторые используются в разных марках. Таблицы: страна, фирма-изготовитель, марка автомобиля, агрегат, узел, деталь. Учет взаимозаменяемости. Пользователи: работник автосервиса, магазинов запчастей; поставщики-оптовики.

## **Цель**

Цель - представить обзор структуры классов автомобильного каталога, который содержит информацию о запчастях автомобиля, их производителях, странах производства, а также о пользователях системы.

## **Исходный код программы на С++**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

// Базовый класс User

class User {

protected:

string name; string role;

public:

User(const string& n, const string& r) : name(n), role(r) {}

virtual void performTask() const = 0;

const string& getName() const {

return name;

}

const string& getRole() const {

return role;

}

};

// Класс Работник автосервиса

class ServiceWorker : public User {

public:

ServiceWorker(const string& n) : User(n, "Работник автосервиса") {}

void performTask() const override {

cout << "Выполняю задачу в автосервисе..." << endl;

}

};

// Класс Работник магазина запчастей

class PartsStoreEmployee : public User {

public:

PartsStoreEmployee(const string& n) : User(n, "Работник магазина запчастей") {}

void performTask() const override {

cout << "Помогаю клиенту в магазине запчастей..." << endl;

}

};

// Класс Поставщик-оптовик

class WholesaleSupplier : public User {

public:

WholesaleSupplier(const string& n) : User(n, "Поставщик-оптовик") {}

void performTask() const override {

cout << "Поставляю запчасти оптом..." << endl;

}

};

// Класс Деталь

class Part

{

private:

string name;

string creationDate;

vector<string> alternativeNames;

public:

Part(const string& n, const string& date) : name(n), creationDate(date) {}

const string& getName() const

{ return name; }

const string& getCreationDate() const

{ return creationDate; }

// Добавление взаимозаменяемой детали по имени

void addAlternative(const string& altPartName)

{ // Проверяем, что такой детали еще нет в списке

if (find(alternativeNames.begin(), alternativeNames.end(), altPartName) == alternativeNames.end())

{ alternativeNames.push\_back(altPartName);

cout << "Альтернативная деталь '" << altPartName << "' добавлена для '" << name << "'." << endl; }

else

{ cout << "Ошибка: Деталь '" << altPartName << "' уже является альтернативой для '" << name << "'." << endl; }

}

// Замена выбранной детали на одну из альтернативных деталей

bool replaceWithAlternative(const string& altPartName)

{ // Проверяем, что альтернативная деталь существует в списке альтернатив

auto it = find(alternativeNames.begin(), alternativeNames.end(), altPartName); if (it != alternativeNames.end())

{ // Заменяем текущую деталь на выбранную альтернативу

name = altPartName; return true;

}

else {

cout << "Ошибка: Деталь '" << altPartName << "' не является альтернативой для '" << name << "'." << endl;

return false;

}

}

// Удаление взаимозаменяемой детали по имени

void removeAlternative(const string& altPartName)

{ auto it = find(alternativeNames.begin(), alternativeNames.end(), altPartName);

if (it != alternativeNames.end())

{ alternativeNames.erase(it);

cout << "Альтернативная деталь '" << altPartName << "' удалена для '" << name << "'." << endl;

} else

{ cout << "Ошибка: Деталь '" << altPartName << "' не является альтернативой для '" << name << "'." << endl;

}

}

// Вывод списка взаимозаменяемых деталей

void printAlternatives() const

{ cout << "Альтернативные детали для '" << name << "':" << endl;

for (const auto& altName : alternativeNames)

{ cout << "- " << altName << endl; }

}

friend ostream& operator<<(ostream& os, const Part& part)

{

os << "Деталь: " << part.getName() << endl;

os << "Дата создания: " << part.getCreationDate() << endl;

return os;

}

};

// Класс Узел

class Component

{

private:

string name; vector<pair<Part, int>> partsRequired; string creationDate;

public:

Component(const string& n, const string& date) : name(n), creationDate(date) {}

void addPart(const Part& part, int quantity)

{ partsRequired.push\_back(make\_pair(part, quantity)); }

const string& getName() const

{ return name; }

const string& getCreationDate() const

{ return creationDate; }

const vector<pair<Part, int>>& getRequiredParts() const

{ return partsRequired; }

friend ostream& operator<<(ostream& os, const Component& component)

{ os << "Узел: " << component.getName() << endl;

os << "Дата создания: " << component.getCreationDate() << endl;

os << "Требуемые детали:" << endl;

for (const auto& pair : component.partsRequired)

{ os << " - " << pair.first.getName() << " (количество: " << pair.second << ")" << endl; }

return os;

}

};

// Класс Агрегат

class Assembly

{

private:

string name; vector<pair<Component, int>> componentsRequired;

string creationDate;

public:

Assembly(const string& n, const string& date) : name(n), creationDate(date) {}

void addComponent(const Component& component, int quantity)

{ componentsRequired.push\_back(make\_pair(component, quantity)); }

const string& getName() const

{ return name; }

const string& getCreationDate() const

{ return creationDate; }

const vector<pair<Component, int>>& getRequiredComponents() const

{ return componentsRequired; }

friend ostream& operator<<(ostream& os, const Assembly& assembly)

{ os << "Агрегат: " << assembly.getName() << endl;

os << "Дата создания: " << assembly.getCreationDate() << endl;

os << "Требуемые компоненты:" << endl;

for (const auto& pair : assembly.componentsRequired)

{ os << " - " << pair.first.getName() << " (количество: " << pair.second << ")" << endl; } return os; }

};

// Класс Страна

class Country

{

private:

string name;

public:

Country(const string& n) : name(n) {}

const string& getName() const

{ return name; }

friend ostream& operator<<(ostream& os, const Country& country)

{ os << "Страна: " << country.getName() << endl;

return os; }

};

// Класс Фирма-изготовитель

class Manufacturer

{

private:

string name; Country country; string creationDate;

public:

Manufacturer(const string& n, const Country& c, const string& date) : name(n), country(c), creationDate(date) {}

const string& getName() const

{ return name;}

const Country& getCountry() const

{ return country; }

const string& getCreationDate() const

{ return creationDate; }

friend ostream& operator<<(ostream& os, const Manufacturer& manufacturer)

{ os << "Фирма-изготовитель: " << manufacturer.getName() << endl;

os << "Страна: " << manufacturer.getCountry().getName() << endl;

os << "Дата создания: " << manufacturer.getCreationDate() << endl;

return os; }

};

// Класс Марка автомобиля

class CarBrand

{

private:

string name; Manufacturer manufacturer; string creationDate;

vector<Assembly> assemblies;

public:

CarBrand(const string& n, const Manufacturer& m, const string& date) : name(n), manufacturer(m), creationDate(date) {}

const string& getName() const

{ return name; }

const Manufacturer& getManufacturer() const

{ return manufacturer; }

const string& getCreationDate() const

{ return creationDate; }

void addAssembly(const Assembly& assembly)

{ assemblies.push\_back(assembly); }

const vector<Assembly>& getAssemblies() const

{ return assemblies; }

friend ostream& operator<<(ostream& os, const CarBrand& carBrand)

{

os << "Марка автомобиля: " << carBrand.getName() << endl;

os << "Фирма-изготовитель: " << carBrand.getManufacturer().getName() << endl; os << "Дата создания: " << carBrand.getCreationDate() << endl;

os << "Агрегаты:" << endl;

for (const auto& assembly : carBrand.assemblies)

{ os << assembly << endl; // Используем перегруженный оператор << для вывода каждого агрегата

} return os; }};

// Функция для вывода объектов определенного типа

template<typename T>

void printObjects(const vector<T>& objects)

{ if (objects.empty())

{ cout << "Нет добавленных объектов данного типа." << endl;

return; }

for (const auto& obj : objects)

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

cout << obj; // Используем перегруженный оператор << для вывода объекта

}

}

## **Структура классов:**

**User**: Базовый класс, представляющий пользователя системы.

Атрибуты:

* name (string): имя пользователя.
* role (string): роль пользователя.

Методы:

* void performTask() const: чисто виртуальный метод для выполнения задачи.

**ServiceWorker, PartsStoreEmployee, WholesaleSupplier**: Подклассы пользователя, представляющие работника автосервиса, работника магазина запчастей и поставщика-оптовика соответственно.

Атрибуты:

* Унаследованы от класса User: name, role.

Методы:

* Все методы унаследованы от базового класса User

**Part:** Класс, представляющий деталь или запчасть автомобиля.

Атрибуты:

* name (string): имя детали.
* creationDate (string): дата создания детали.
* alternativeNames (vector<string>): список взаимозаменяемых деталей.

Методы:

* void addAlternative(const string& altPartName): добавляет взаимозаменяемую деталь по имени.
* bool replaceWithAlternative(const string& altPartName): заменяет текущую деталь на выбранную альтернативу.
* void removeAlternative(const string& altPartName): удаляет взаимозаменяемую деталь по имени.
* void printAlternatives() const: выводит список взаимозаменяемых деталей.

**Componen**t: Класс, представляющий узел автомобиля, содержащий требуемые детали и их количество.

Атрибуты:

* name (string): имя узла.
* partsRequired (vector<pair<Part, int>>): список требуемых деталей и их количества.
* creationDate (string): дата создания узла.

Методы:

* void addPart(const Part& part, int quantity): добавляет деталь и ее количество к узлу.
* const vector<pair<Part, int>>& getRequiredParts() const: возвращает требуемые детали и их количество.

**Assembly**: Класс, представляющий агрегат автомобиля, содержащий требуемые компоненты и их количество.

Атрибуты:

* name (string): имя агрегата.
* componentsRequired (vector<pair<Component, int>>): список требуемых компонентов и их количества.
* creationDate (string): дата создания агрегата.

Методы:

* void addComponent(const Component& component, int quantity): добавляет компонент и его количество к агрегату.
* const vector<pair<Component, int>>& getRequiredComponents() const: возвращает требуемые компоненты и их количество.

**Country**: Класс, представляющий страну происхождения.

Атрибуты:

* name (string): название страны.

Методы:

* метод для вывода данных

**Manufacturer**: Класс, представляющий фирму-изготовителя деталей или автомобилей.

Атрибуты:

* name (string): название производителя.
* country (Country): страна производителя.
* creationDate (string): дата создания производителя.

Методы:

* метод для вывода данных

**CarBrand**: Класс, представляющий марку автомобиля, содержащий информацию о фирме-изготовителе и списке агрегатов.

Атрибуты:

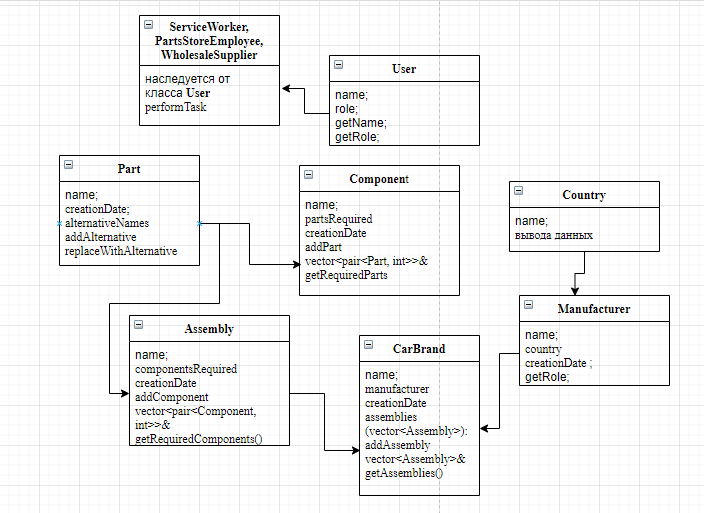
* name (string): название марки автомобиля.
* manufacturer (Manufacturer): производитель марки автомобиля.
* creationDate (string): дата создания марки автомобиля.
* assemblies (vector<Assembly>): список агрегатов марки автомобиля.

Методы:

* void addAssembly(const Assembly& assembly): добавляет агрегат к марке автомобиля.
* const vector<Assembly>& getAssemblies() const: возвращает список агрегатов.

**printObjects**: Функция для вывода объектов определенного типа из вектора.

## **Диаграмма Классов**



## **Вывод**

Классы автомобильного каталога представляют собой структуру данных, которая позволяет организовать информацию о запчастях, компонентах, агрегатах и производителях автомобилей. Каждый класс имеет определенную функциональность, позволяющую эффективно управлять данными и выполнять операции с ними.

# **Задание 1. (17.18).**

## **Постановка задачи**

*Шеф-повар.*

База рецептур блюд: раскладка, рецепт приготовления. База продуктов на складе: наименование, цена, количество. Формирование меню на день (на заданное число персон), корректировка блюд, продуктов.

## **Цель**

Цель - представить обзор структуры программы, предназначенной для управления базой рецептур блюд, базой продуктов на складе и формирования меню на день для заданного числа персон. Программа написана на языке C++ с использованием базы данных MySQL для хранения информации.

## **Исходный код программы на С++**

#include <stdlib.h>

#include <iostream>

#include "Menu.h"

#include "dish.h"

#include "ingredients.h"

#include "Products.h"

#include <stdlib.h>

#include <iostream>

#include "mysql\_connection.h"

#include <cppconn/driver.h>

#include <cppconn/exception.h>

#include <cppconn/prepared\_statement.h>

using namespace std;

const string server = "tcp://127.0.0.1";

const string username = "root";

const string password = "";

const string DB = "Chef\_povar";

sql::Connection\* ConectDB();

#pragma once

#include "Connect.h"

#include "table.h"

#include <iostream>

#include <vector>

using namespace std;

class dish :public tables

{

private:

vector<int> ids; vector<string> names;

public:

int Get\_id();

void Update(int id, string name);

void Insert(string name);

void Delete(int id);

int Check();

void Print();

void Print\_by\_id(int id);

int cnt();

};

#pragma once

#include "Connect.h"

#include "table.h"

#include <iostream>

#include <vector>

using namespace std;

class ingredients

{

private:

vector<int> ids; vector<int> id\_prod; vector<int> id\_dish;

vector<int> quantity;

public:

int Get\_id(int id\_dish);

void Update(int id\_ing,int id\_prod, int quantity);

void Insert(int id\_ing,int id\_prod,int quantity);

void Delete(int id\_ing);

int Check();

void Print();

void Print\_by\_id(int id);

};

#pragma once

#include "Connect.h"

#include "table.h"

#include <iostream>

#include <vector>

using namespace std;

class Products :public tables

{

private:

vector<int> ids; vector<string> names; vector<int> amount;

vector<int> quantity;

public: int Get\_id();

void Update(int id, string name);

void Update\_A(int id, int amount);

void Update\_Q(int id, int quantiyt);

void Update(int id, string name, int amount, int quantiyt);

void Insert(string name);

void Insert(string name,int amount,int quntity);

void Delete(int id);

int Check();

void Print();

void Print\_by\_id(int id);

};

#pragma once

#include <iostream>

using namespace std;

class tables

{

virtual int Get\_id() = 0;

virtual void Insert(string name) = 0;

virtual void Update(int id, string name) = 0;

virtual void Delete(int id) = 0;

virtual int Check() = 0;

virtual void Print() = 0;

virtual void Print\_by\_id(int id) = 0;

};

#pragma once

#include "Connect.h"

#include "table.h"

#include <iostream>

#include <vector>

using namespace std;

class Menu

{

private:

vector<int> ids; vector<string> date;

vector<int> id\_dish; vector<int> quantity;

public:

int Get\_id() {

cout << "hello";

}

void Update(int id, string name);

void Insert(string date,int id\_dish,int quantity) {

sql::PreparedStatement\* pstmt;

sql::Connection\* con;

try

{ con = ConectDB();

pstmt = con->prepareStatement("INSERT INTO menu\_for\_day (date,id\_dish,quantity) VALUES(?,?,?)");

pstmt->setString(1, date);

pstmt->setInt(2, id\_dish);

pstmt->setInt(3, quantity);

pstmt->execute();

cout << "Saved" << endl;

delete pstmt;

delete con;

}

catch (sql::SQLException e)

{cout << "Cannot insert. Error message: " << e.what() << endl;

system("pause"); }

}

void Delete(int id);

int Check(int id\_dish,int quantity) {

sql::PreparedStatement\* pstmt;

sql::Connection\* con;

sql::ResultSet\* result;

int a = 0;

con = ConectDB();

pstmt = con->prepareStatement("Call SP\_Checkprod(?,?)");

pstmt->setInt(1, id\_dish);

pstmt->setInt(2, quantity);

result = pstmt->executeQuery();

if (result->next()) {

a= result->getInt(1);

delete pstmt;

delete con;

delete result;

return a;

}

else

{

delete pstmt;

delete con;

delete result;

return 0;

}

}

void Print() {

sql::PreparedStatement\* pstmt;

sql::Connection\* con;

sql::ResultSet\* result;

con = ConectDB();

pstmt = con->prepareStatement("CALL showMenu() ");

result = pstmt->executeQuery();

cout << endl << "Список Menu" << endl;

cout << "Date" << "\t\tName" << "\t\tQuantity" << endl;

while (result->next()) {

cout << result->getString("date") <<"\t" << result->getString("name") <<"\t" << result->getInt("quantity") << endl;

}

delete result;

delete pstmt;

delete con;

system("pause");

system("cls");

}

void Print\_by\_id(int id);

};

## **Структура классов:**

**Connect**: файл, содержащий функции для соединения с базой данных MySQL.

Методы:

* sql::Connection\* ConectDB() - устанавливает соединение с базой данных.

**dish**: Класс dish, представляющий таблицу с информацией о блюдах.

Атрибуты:

* vector<int> ids;
* vector<string> names;

Методы:

* int Get\_id() - получает ID блюда.
* void Update(int id, string name) - обновляет название блюда.
* void Insert(string name) - вставляет новое блюдо в таблицу.
* void Delete(int id) - удаляет блюдо из таблицы.
* int Check() - проверяет наличие записей в таблице.
* void Print() - выводит содержимое таблицы.
* void Print\_by\_id(int id) - выводит информацию о блюде по его ID.

**ingredients:** Класс ingredients, представляющий таблицу с информацией о составе блюд.

Атрибуты:

* vector<int> ids;
* vector<int> id\_prod;
* vector<int> id\_dish;
* vector<int> quantity;

Методы:

* int Get\_id(int id\_dish) - получает ID ингредиента для указанного блюда.
* void Update(int id\_ing,int id\_prod, int quantity) - обновляет информацию об ингредиенте.
* void Insert(int id\_ing,int id\_prod,int quantity) - добавляет новый ингредиент.
* void Delete(int id\_ing) - удаляет ингредиент из таблицы.
* int Check() - проверяет наличие записей в таблице.
* void Print() - выводит содержимое таблицы.
* void Print\_by\_id(int id) - выводит информацию о составе блюда по его ID.

**Products**: Класс Products, представляющий таблицу с информацией о продуктах на складе.

Атрибуты:

* vector<int> ids;
* vector<string> names;
* vector<int> amount;
* vector<int> quantity;

Методы:

* int Get\_id() - получает ID продукта.
* void Update(int id, string name) - обновляет название продукта.
* void Update\_A(int id, int amount) - обновляет количество продукта на складе.
* void Update\_Q(int id, int quantiyt) - обновляет количество продукта в одной единице.
* void Update(int id, string name, int amount, int quantiyt) - обновляет информацию о продукте.
* void Insert(string name) - вставляет новый продукт в таблицу.
* void Insert(string name,int amount,int quntity) - вставляет новый продукт с указанным количеством и количеством в одной единице.
* void Delete(int id) - удаляет продукт из таблицы.
* int Check() - проверяет наличие записей в таблице.
* void Print() - выводит содержимое таблицы.
* void Print\_by\_id(int id) - выводит информацию о продукте по его ID.

**Menu**: Класс Menu, представляющий таблицу с информацией о меню на день.

Атрибуты:

* vector<int> ids;
* vector<string> date;
* vector<int> id\_dish;
* vector<int> quantity;

Метод:

* int Get\_id() - получает ID меню.
* void Update(int id, string name) - обновляет название блюда в меню.
* void Insert(string date,int id\_dish,int quantity) - вставляет новое блюдо в меню на указанную дату с указанным количеством порций.
* void Delete(int id) - удаляет блюдо из меню.
* int Check(int id\_dish,int quantity) - проверяет наличие достаточного количества ингредиентов на складе для приготовления блюда.
* void Print() - выводит содержимое меню.
* void Print\_by\_id(int id) - выводит информацию о блюде в меню по его ID.

**table**: Класс tables, представляющий общий интерфейс для работы с таблицами базы данных.

Методы: (виртуальные):

int Get\_id() - получает ID записи.

void Insert(string name) - вставляет новую запись в таблицу.

void Update(int id, string name) - обновляет информацию о записи.

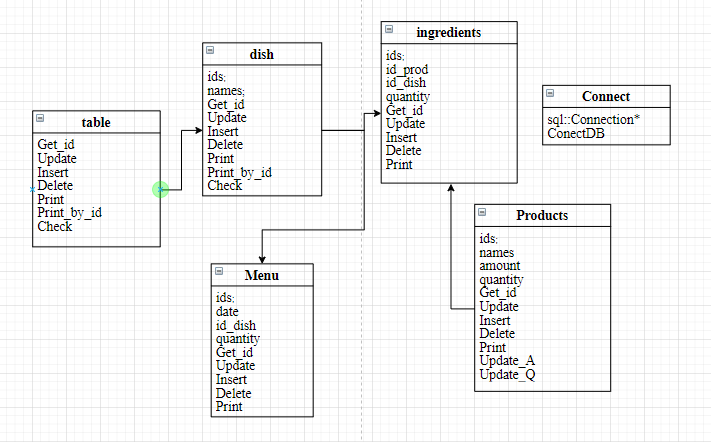
void Delete(int id) - удаляет запись из таблицы.

int Check() - проверяет наличие записей в таблице.

void Print() - выводит содержимое таблицы.

void Print\_by\_id(int id) - выводит информацию о записи по ее ID.

## **Диаграмма Классов**



## **Вывод**

Программа предоставляет удобный интерфейс для управления базой данных рецептур блюд, продуктов на складе и формирования меню на день. Реализована модульная структура с использованием классов для каждой таблицы базы данных, что обеспечивает легкость в расширении и поддержке программы.