Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

кафедра ПМ иК

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Тема: «Кофемашина»

Выполнил: студент группы ИП-317

Монастырный М.Е.

Проверил: ассистент кафедры ПМиК

Сороковых Д.А.

Новосибирск – 2024

Содержание

Постановка задачи

Написать полноценную структуру кофемашины, с иерархией ингредиентов и ее составляющими, используя технологии ООП, такие как инкапсуляция, наследование и т.д. Также реализовать пользовательский ввод заказа.

Структура проекта

Файловая структура

/bin – **исполнительный файл**

| coffee_machine.exe

/headers – **заголовочные файлы**

| coffee_machine.h

| ingredients.h

| menu.h

| order_parser.h

| order.h

| parser.h

| recipes.h

/src - **реализация**

| coffee_machine.cpp

| ingredients.cpp

| menu.cpp

| order_parser.cpp

| order.cpp

| parser.cpp

```
|    recipes.cpp
order.txt
recipes.txt
makefile
```

Технологии ООП

Инкапсуляция

Инкапсуляция — это механизм ООП, при котором данные (поля класса) и методы, работающие с этими данными, объединяются в одном классе. Все поля данных скрыты от внешнего доступа, а доступ к ним осуществляется только через специальные методы (геттеры и сеттеры).

```
class Recipe
{
private:
    std::string name;              // приватные поля
    std::map<std::string, unsigned> ingredients;


public:
    void setName(std::string n);
    std::string getName() const;   // доступы к ним(геттеры, сеттеры)
    void setIngredients(std::map<std::string, unsigned>
&newIngredients);
    std::map<std::string, unsigned> getIngredients() const;
};
```

Наследование

Наследование – это механизм, который позволяет одному классу (потомку) наследовать свойства и методы другого класса (родителя). Наследование может быть как явным (класс наследует все публичные и защищенные

члены), так и абстрактным (класс может быть абстрактным, если не реализует все методы).

```
class DryIngredient : public Ingredient {
    //
};
```

Полиморфизм

Полиморфизм — это способность объектов разных классов реагировать на одинаковые сообщения (вызовы методов) по-разному, что достигается через переопределение методов.

```
void setIngredients(std::map<std::string, unsigned> &newIngredients);
std::map<std::string, unsigned> getIngredients() const;
```

Конструкторы и перегрузка конструкторов

Конструктор — это специальная функция-член класса, которая вызывается при создании объекта. Перегрузка конструкторов — это возможность иметь несколько конструкторов в классе с разными параметрами.

```
// Конструктор в Ingredient
Ingredient() : temperature(24), amount(0), time(0) {}
```

Списки инициализации

Список инициализации — это способ инициализации членов данных в конструкторе до выполнения его тела. Это необходимо для инициализации константных членов, ссылок или для более эффективной инициализации.

```
// Инициализация в CoffeeBeans
CoffeeBeans() : is_grinded(false) {}
```

Виртуальная функция

Виртуальная функция — это функция-член класса, которая объявляется с ключевым словом virtual. Она позволяет переопределять поведение функции в классах-наследниках. Виртуальные функции поддерживают механизм полиморфизма, который позволяет использовать одну и ту же сигнатуру функции, но с разным поведением для объектов разных классов.

```
virtual void parseOrder(std::string filename, std::vector<Order>
&ords) = 0;

virtual ~IOrderParser() {}
```

Использование объектов в качестве аргументов или возвращаемых значений

Использование объектов в качестве аргументов или возвращаемых значений означает передачу объектов классов как параметров в функции или возврат объектов из функций. Это позволяет функции работать с объектами, манипулировать их данными или создавать новые объекты, не изменяя саму структуру программы.

```
bool Menu::canPrepareRecipe(const Order& order, const Recipe& recipe)
{
      // Содержимое метода
}
```

Заголовочные файлы и структуры классов

Файл headers/coffee_machine.h

Описание структуры основных компонентов, определение некоторых методов, например prepareCoffee().

```
#ifndef COFFEE_MACHINE_H

#define COFFEE_MACHINE_H
```

```
#include "ingredients.h"
#include "recipes.h"
#include "order.h"
#include "parser.h"
#include "order_parser.h"
#include <cstdlib>
#include <ctime>

// Класс кофемолка
class CoffeeGrinder
{
public:
    void grind(CoffeeBeans &c); // помол
};

// Класс "группа"
class Group
{
public:
    void boil(CoffeeBeans &c); // варка
};

// Класс бойлер
class Boiler
{
public:
```

```
    void heat(Water &w); // нагрев
};

// Класс форсунок
class Nozzle
{
public:
    void heat(WetIngredient &ingredient); // нагрев
    void mix(DryIngredient &ingredient);  // размешать
};

// Класс кофемашины целиком
class CoffeeMachine
{
private:
    CoffeeGrinder grinder;
    Group group;
    Boiler boiler;
    Nozzle nozzle;

public:
    bool found = false;
    int prepareCoffee(const Recipe &r);
};

#endif
```

Файл headers/ingredients.h

Класс и подклассы различных ингредиентов.

```
#ifndef INGREDIENTS_H
#define INGREDIENTS_H

// Класс ингредиентов
class Ingredient {
private:
    int temperature; // температура
    int amount; // количество
    int time; // время, которое они требуют
public:
    Ingredient() : temperature(24), amount(0), time(0) {}
    void setTemperature(int t);
    int getTemperature() const;
    void setAmount(int a);
    int getAmount() const;
    void setTime(int t);
    int getTime() const;
};

// Класс мокрых ингредиентов
class WetIngredient : public Ingredient {
    //
};

// Класс сухих ингредиентов
```

```
class DryIngredient : public Ingredient {
    //
};

// Класс кофейных зёрен
class CoffeeBeans : public DryIngredient {
private:
    bool is_grinded;
public:
    CoffeeBeans() : is_grinded(false) {}
    void setStatus(bool s);
    bool getStatus() const;
};

// Класс сахар
class Sugar : public DryIngredient {
    //
};

// Класс молоко
class Milk : public WetIngredient {
    //
};

// Класс вода
class Water : public WetIngredient {
    //
```

```
};

#endif // INGREDIENTS_H
```

Файл headers/menu.h

Класс Menu. В нем определеные такие методы, как start(), canPrepareRecipe().

```
#ifndef MENU_H

#define MENU_H


#include <string>

#include <vector>

#include <iostream>

#include <map>


#include "recipes.h"

#include "order.h"

#include "coffee_machine.h"


// Класс вывода структур данных на экран

class Print

{

public:

    void printMap(std::map<std::string, unsigned> ingredients);

    void printVector(std::vector<Recipe> r);

};


// Класс меню приложения

class Menu
```

```
{
public:
    int numberName();
    std::string searchName(std::vector<Recipe> &r, int &input);
    void start(std::vector<Recipe> r, CoffeeMachine &cm,
std::vector<Order> o);

private:
    bool canPrepareRecipe(const Order &order, const Recipe &recipe);
};

#endif // MENU_H
```

Файл headers/order_parser.h

Интерфейс парсера заказов.

```
#ifndef ORDER_PARSER_H
#define ORDER_PARSER_H

#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <map>
#include <sstream>

#include "order.h"
```

```
// Интерфейс парсера
class IOrderParser
{
public:
    virtual void parseOrder(std::string filename, std::vector<Order>
&ords) = 0;
    virtual ~IOrderParser() {}
};

// Реализация парсера для файлов .txt
class TextOrderParser : public IOrderParser
{
public:
    void parseOrder(std::string filename, std::vector<Order> &ords)
override;
};

#endif // PARSER_H
```

Файл headers/order.h

Класс заказа.

```
#ifndef ORDER_H
#define ORDER_H

#include <map>
#include <string>
```

```
class Order
{
private:
    std::map<std::string, unsigned> ingredients;

public:
    void setIngredients(std::map<std::string, unsigned>
&newIngredients);
    std::map<std::string, unsigned> getIngredients() const;
};

#endif
```

Файл headers/parser.h

Интерфейс парсера рецептов.

```
#ifndef PARSER_H
#define PARSER_H

#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <map>
#include <sstream>

#include "recipes.h"
```

```
// class Recipe;


// Интерфейс парсера
class IRecipeParser
{
public:
    virtual void parse(std::string filename, std::vector<Recipe>
&recipes) = 0;
    virtual ~IRecipeParser() {}
};


// Реализация парсера для файлов .txt
class TextRecipeParser : public IRecipeParser
{
public:
    void parse(std::string filename, std::vector<Recipe> &recipes)
override;
};


#endif // PARSER_H
```

Файл headers/recipes.h

Класс рецепта.

```
#ifndef RECIPES_H
#define RECIPES_H
```

```
#include <map>
#include <string>

// Класс рецепта
class Recipe
{
private:
    std::string name;
    std::map<std::string, unsigned> ingredients;

public:
    void setName(std::string n);
    std::string getName() const;
    void setIngredients(std::map<std::string, unsigned>
&newIngredients);
    std::map<std::string, unsigned> getIngredients() const;
};

#endif // RECIPES_H
```

Программная реализация

Файл src/coffee_machine.cpp

В этом файлы реализуются методы класса кофемашины, а также ее компонентов и методов.

```
#include "coffee_machine.h"
```

```
void CoffeeGrinder::grind(CoffeeBeans &c)
{
    int time = 14 + rand() % (19 - 14 + 1);
    c.setTime(c.getTime() + time);
    c.setStatus(true);
}

void Group::boil(CoffeeBeans &c)
{
    if (c.getStatus())
    {
        int time = 19 + rand() % (24 - 19 + 1);
        c.setTime(c.getTime() + time);
    }
    else
    {
        return;
    }
}

void Boiler::heat(Water &w)
{
    int toAdd = 100 - w.getTemperature();
    w.setTemperature(w.getTemperature() + toAdd);
    w.setTime(w.getTime() + toAdd);
}
```

```
void Nozzle::heat(WetIngredient &wi)
{
    int toAdd = (65 + rand() % (70 - 65 + 1)) - wi.getTemperature();
    wi.setTemperature(wi.getTemperature() + toAdd);
    wi.setTime(wi.getTime() + toAdd);
}

void Nozzle::mix(DryIngredient &di)
{
    di.setTime(di.getTime() + 10);
}

int CoffeeMachine::prepareCoffee(const Recipe &r)
{
    r.getIngredients();
    int allTime = 0;
    for (auto ingredient : r.getIngredients())
    {
        if (ingredient.first == "milk" || ingredient.first == "cream")
        {
            srand(time(NULL));
            for (int i = 0; i < ingredient.second; i++)
            {
                Milk milk;
                nozzle.heat(milk);
                allTime += milk.getTime();
            }
```

```
}
else if (ingredient.first == "coffee")
{
    srand(time(NULL));
    for (int i = 0; i < ingredient.second; i++)
    {
        CoffeeBeans cb;
        grinder.grind(cb);
        group.boil(cb);
        allTime += cb.getTime();
    }
}
else if (ingredient.first == "water")
{
    for (int i = 0; i < ingredient.second; i++)
    {
        Water w;
        boiler.heat(w);
        allTime += w.getTime();
    }
}
else if (ingredient.first == "sugar")
{
    for (int i = 0; i < ingredient.second; i++)
    {
        DryIngredient sugar;
        nozzle.mix(sugar);
```

```
                allTime += sugar.getTime();
            }
        }
    }
    return allTime;
}
```

Файл src/ingredients.cpp

В этом файле реализуются методы класса Ingredient.

```
#include "ingredients.h"

void Ingredient::setTemperature(int newTemperature)
{
    temperature = newTemperature;
}

int Ingredient::getTemperature() const
{
    return temperature;
}

void Ingredient::setAmount(int newAmount)
{
    amount = newAmount;
}

int Ingredient::getAmount() const
```

```
{
    return amount;
}

void Ingredient::setTime(int newTime)
{
    time = newTime;
}

int Ingredient::getTime() const
{
    return time;
}

void CoffeeBeans::setStatus(bool s)
{
    is_grinded = s;
}

bool CoffeeBeans::getStatus() const
{
    return is_grinded;
}
```

Файл src/menu.cpp

В этом файле реализуются основные методы класса Menu – start() и canPrepareRecipe()

```
#include "menu.h"
```

```
bool Menu::canPrepareRecipe(const Order &order, const Recipe &recipe)
{
    auto recipeIngredients = recipe.getIngredients();
    auto orderIngredients = order.getIngredients();

    // Проверяем, что все ингредиенты рецепта есть в заказе и в нужном количестве
    for (const auto &ingredient : recipeIngredients)
    {
        const std::string &name = ingredient.first;
        unsigned requiredAmount = ingredient.second;

        // Проверяем, есть ли ингредиент в заказе
        if (orderIngredients.find(name) == orderIngredients.end())
        {
            return false;
        }

        // Проверяем, соответствует ли количество ингредиента
        unsigned availableAmount = orderIngredients.at(name);
        if (availableAmount != requiredAmount)
        {
            return false;
        }
    }

    // Проверяем, что в заказе нет лишних ингредиентов
```

```
    for (const auto &ingredient : orderIngredients)
    {
        const std::string &name = ingredient.first;
        if (recipeIngredients.find(name) == recipeIngredients.end())
        {
            return false;
        }
    }

    return true;
}

void Menu::start(std::vector<Recipe> r, CoffeeMachine &cm,
std::vector<Order> o)
{
    std::cout << "+-------------------------------------------------
----------------+" << std::endl;
    std::cout << "|                            Добро пожаловать!
|" << std::endl;
    std::cout << "| С помощью нашей кофемашины вы можете сварить себе
вкуснейший кофе!|" << std::endl;
    std::cout << "|                                d( > _ • )
|" << std::endl;
    std::cout << "+-------------------------------------------------
----------------+" << std::endl;

    std::string off;
    std::cout << "+---------------+" << std::endl;
    std::cout << "| Начать (1/0)? |\n";
```

```
    std::cout << "+---------------+" << std::endl;
    std::cin >> off;
    for (const Order &order : o)
    {
        std::cout << "+-----------------------------------------------
--------------------+" << std::endl;
        std::cout << "|            Ваш заказ содержит следующие
ингредиенты:               |" << std::endl;
        std::cout << "+-----------------------------------------------
--------------------+" << std::endl;
        for (const auto &ingredient : order.getIngredients())
        {
            std::cout << " - " << ingredient.first << " (" <<
ingredient.second << ")" << std::endl;
        }

        bool recipeFound = false;
        for (const Recipe &recipe : r)
        {
            if (canPrepareRecipe(order, recipe))
            {
                std::cout << "+-------------------------------------
----------------------------+" << std::endl;
                std::cout << "|                Название рецепта: " <<
recipe.getName() << std::endl;
                std::cout << "+-------------------------------------
----------------------------+" << std::endl;
                std::cout << "Ингредиенты рецепта:" << std::endl;
                for (const auto &ingredient : recipe.getIngredients())
```

```
                {
                    std::cout << " - " << ingredient.first << " (" <<
ingredient.second << ")" << std::endl;
                }

                int totalPreparationTime = cm.prepareCoffee(recipe);
                std::cout << "+-------------------------------------
----------------------------+" << std::endl;
                std::cout << "Приготовление: " << recipe.getName()
                          << " (Итоговое время: " <<
totalPreparationTime
                          << " секунд)." << std::endl;
                std::cout << "+-------------------------------------
----------------------------+" << std::endl;
                recipeFound = true;
                break;
            }
        }

        if (!recipeFound)
        {
            std::cout << "Ошибка: такой рецепт неизвестен или
недостаточно ингредиентов." << std::endl;
        }
    }
}
```

Файл src/order_parser.cpp

Здесь реализуется сам парсер заказа

```
#include "order_parser.h"


void TextOrderParser::parseOrder(std::string filename,
std::vector<Order> &o)
{
    std::ifstream file(filename);
    if (!file.is_open())
    {
        std::cerr << "Ошибка открытия файла: " << filename <<
std::endl;
        return;
    }


    std::string line;
    while (std::getline(file, line))
    {
        if (line.empty())
            continue; // Пропуск пустых строк


        Order order;
        std::map<std::string, unsigned> ingredients;


        size_t colonPos = line.find(':');
        if (colonPos == std::string::npos)
            continue;
```

```
        line = line.substr(colonPos + 1); // Убираем название "Заказ:"
        std::istringstream ss(line);
        std::string ingredientPair;

        while (std::getline(ss, ingredientPair, ','))
        {
            size_t nameEnd = ingredientPair.find('(');
            size_t amountEnd = ingredientPair.find(')');

            if (nameEnd == std::string::npos || amountEnd ==
std::string::npos)
                continue;

            std::string name = ingredientPair.substr(0, nameEnd);
            std::string count = ingredientPair.substr(nameEnd + 1,
amountEnd - nameEnd - 1);

            name.erase(std::remove_if(name.begin(), name.end(),
::isspace), name.end());
            count.erase(std::remove_if(count.begin(), count.end(),
::isspace), count.end());

            try
            {
                int n = std::stoi(count);
                ingredients[name] = n;
            }
```

```
            catch (const std::exception &)
            {
            }
        }

        order.setIngredients(ingredients);
        o.push_back(order);
    }
    file.close();
}
```

Файл src/order.cpp

В этом файле происходит реализация класса заказа.

```
#include "order.h"

void Order::setName(std::string n)
{
    name = n;
}

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

void Order::setIngredients(std::map<std::string, unsigned>
&newIngredients)
```

```
{
    ingredients = newIngredients;
}

std::map<std::string, unsigned> Order::getIngredients() const
{
    return ingredients;
}
```

Файл src/parser.cpp

Здесь реализуется парсер рецепта.

```
#include "parser.h"

void TextRecipeParser::parse(std::string filename, std::vector<Recipe>
&r)
{
    std::ifstream file(filename);
    if (!file.is_open())
    {
        std::cerr << "Ошибка открытия файла: " << filename <<
std::endl;
        return;
    }

    std::string line;
    while (std::getline(file, line))
    {
        if (line.empty())
```

```
            continue;

        size_t colonPos = line.find(':');
        if (colonPos == std::string::npos)
            continue;

        Recipe recipe;
        std::string recipeName = line.substr(0, colonPos);
        recipeName.erase(std::remove_if(recipeName.begin(),
recipeName.end(), ::isspace), recipeName.end());
        recipe.setName(recipeName);

        line = line.substr(colonPos + 1);
        std::istringstream ss(line);
        std::string ingredientPair;

        std::map<std::string, unsigned> ingredients;
        while (std::getline(ss, ingredientPair, ','))
        {
            size_t nameEnd = ingredientPair.find('(');
            size_t amountEnd = ingredientPair.find(')');

            if (nameEnd == std::string::npos || amountEnd ==
std::string::npos)
                continue;

            std::string name = ingredientPair.substr(0, nameEnd);
```

```
            std::string count = ingredientPair.substr(nameEnd + 1,
amountEnd - nameEnd - 1);


            name.erase(std::remove_if(name.begin(), name.end(),
::isspace), name.end());
            count.erase(std::remove_if(count.begin(), count.end(),
::isspace), count.end());


            try
            {
                int n = std::stoi(count);
                ingredients[name] = n;
            }
            catch (const std::exception &)
            {
            }
        }


        recipe.setIngredients(ingredients);
        r.push_back(recipe);
    }
    file.close();
}
```

Файл src/recipes.cpp

В этом исполнительном файле реализованы метода класса Recipe.

```
#include "recipes.h"


void Recipe::setName(std::string n)
```

```
{
    name = n;
}

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

void Recipe::setIngredients(std::map<std::string, unsigned>
&newIngredients)
{
    ingredients = newIngredients;
}

std::map<std::string, unsigned> Recipe::getIngredients() const
{
    return ingredients;
}
```

Файл src/main.cpp

А это основной файл в котором вызываются основные методы и объявляются основные объекты классов.

```
#include <iostream>
#include "coffee_machine.h"
#include "ingredients.h"
#include "recipes.h"
#include "order.h"
#include "parser.h"
#include "order_parser.h"
#include "menu.h"

using namespace std;

int main()
{
    Menu menu;
    TextRecipeParser parser;
    TextOrderParser orderParser;
    CoffeeMachine cm;
    std::vector<Recipe> r;
    std::vector<Order> o;

    parser.parse("recipes.txt", r);
    orderParser.parseOrder("order.txt", o);
    menu.start(r, cm, o);
```

```
    return 0;
}
```

Результаты работы

Сборка производится с помощью Makefile. Для того чтобы запустить проект, в корневой директории проекта нужно прописать в терминал:

- `Make run`

Таким обзраом, мы соберем приложение и запустим исполняемый файл.

Окно приветсвия:

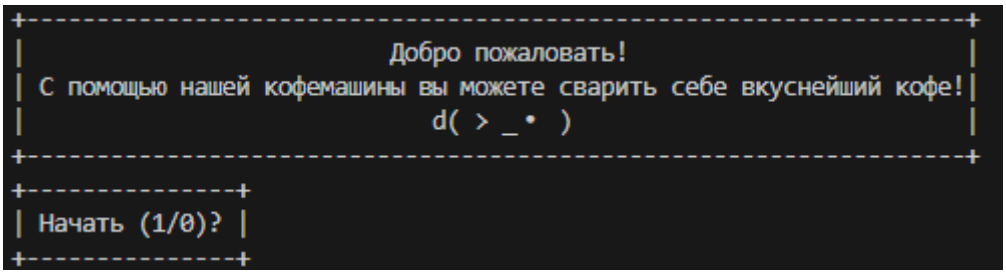

Рисунок 1 – Окно приветсвия

При выборе 1 программа начнет свою работу, при 0 – прекратит.

Программа считывает заказ(ы) из текстового файла order.txt и сравнивает их с каждым рецептом из файла recipes.txt

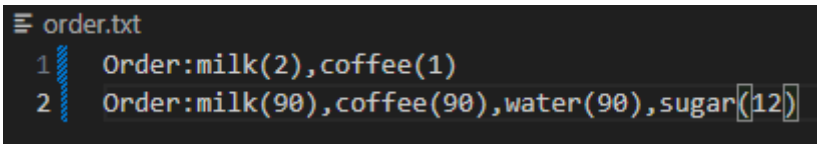

Рисунок 2 – пример заказа

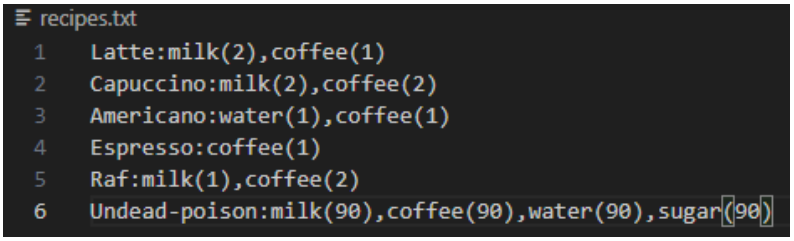

Рисунок 3 – все имеющиеся рецепты

При совпадении заказа с каким-либо из рецептов программа выводит все ингредиенты в этом заказе, а также сам рецепт. Далее вычисляется время приготовления

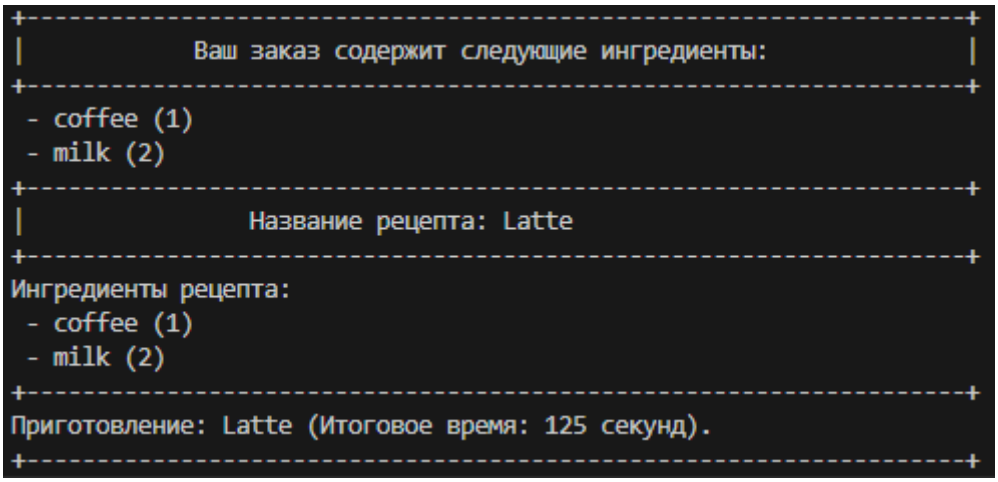

Рисунок 4 – Вывод при соответствии

Во втором заказе намеренно допущена ошибка. При несоответствии количества определенных ингредиентов в заказе и рецептах программа выводит следующее:

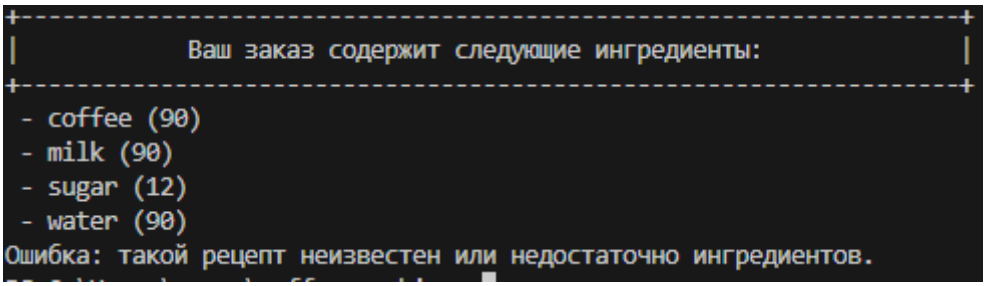

Рисунок 5 – Вывод при несоответствии

Заключение

В течение всего курса я знакомился с ООП, и во время разработки этого проекта я ощутил все плюсы этого подхода к разработке. Принципы ООП позволяют создавать легко масштабируемые приложения с четкой структурой и логикой, которую в дальнейшем легко редактировать, удаляя или добавляя новый функционал. Также код становится более ясным и понятным в глазах

других разработчиков. В ходе разработки этого проекта я использовал принципы ООП, такие как инкапсуляция, наследование и полиморфизм. Также я использовал принципы SOLID, чтобы придерживаться правильной архитектуры приложения.

Используемые источники

1. Курс: Объектно-ориентированное программирование
Электронный ресурс. URL: https://eios.sibsutis.ru/course/view.php?id=291

2. Статья про принципы ООП
Принципы ООП на примерах [Электронный ресурс]. URL:
https://habr.com/ru/companies/otus/articles/764266/

3. Основы ООП https://metanit.com/cpp/tutorial/5.1.php