МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Отчет по лабораторной работе №1
по дисциплине
«Объектно-ориентированное программирование»

Выполнил студент гр. ИВТб-2301-05-00 _____/Макаров С.А./
Руководитель преподаватель _____/Шмакова Н.А./

Киров 2026

Цель работы

Цель работы: изучить и освоить принципы объектно-ориентированного программирования, включая инкапсуляцию, наследование и полиморфизм, путем создания иерархии классов в выбранной предметной области.

Задание

1. Создать иерархию классов состоящую не менее чем из одного родительского и двух дочерних классов.

2. В каждом классе определить не менее двух член данных. не менее двух собственных, а для дочерних не менее двух унаследованных и двух перекрытых член функций.

3. Разработать приложение демонстрирующее принципы инкапсуляции, наследования и полиморфизма.

Решение

Предметная область описывает управление ассортиментом блюд в заведении общественного питания. Система позволяет хранить информацию о различных блюдах, учитывать их специфические характеристики, рассчитывать итоговую цену для клиента в зависимости от этих характеристик и выводить информацию о блюде в удобном виде.

Схема структуры иерархии классов представлена в виде диаграммы классов, представленая на рисунке 1.

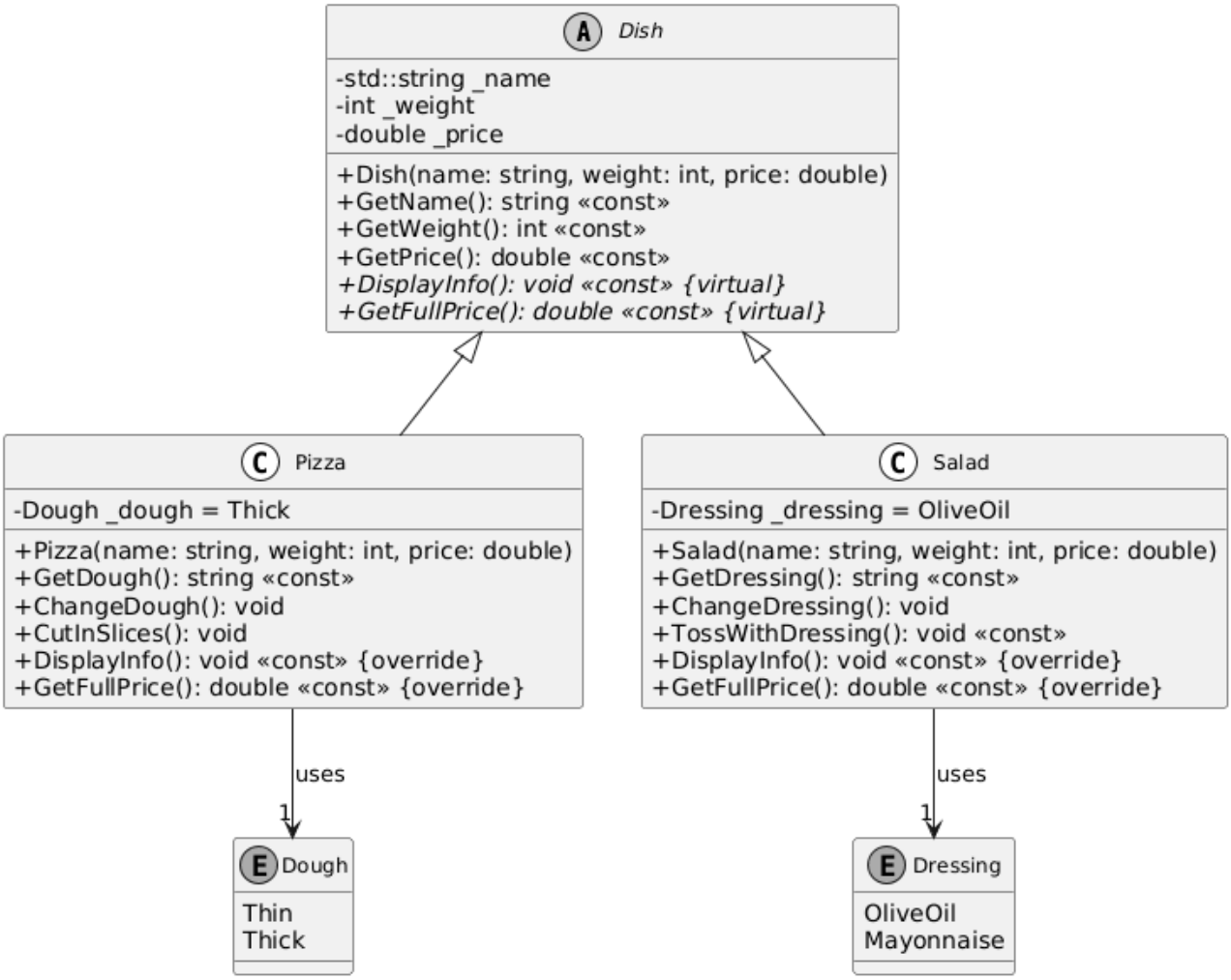

Рисунок 1 – Диаграмма классов

Описание меотодов классов:

– Класс Dish:

  – Dish(string name, int weight, double price) – конструктор, инициализирует название блюда, вес в граммах и базовую цену;

  – string GetName() const – возвращает название блюда;

  – int GetWeight() const – возвращает вес блюда в граммах;

  – double GetPrice() const – возвращает базовую цену блюда;

  – virtual void DisplayInfo() const – выводит основную информацию о блюде: название, вес, итоговую цену;

    virtual double GetFullPrice() const – возвращает цену, которую платит клиент (с наценкой);

- Класс Pizza наследуется от Dish:
  - Pizza(string name, int weight, double price) – конструктор, передаёт параметры в базовый класс Dish, тип теста по умолчанию — Thick;
  - string GetDough() const – возвращает текстовое описание текущего типа теста;
  - void ChangeDough() – переключает тип теста между Thin и Thick;
  - void CutInSlices() – имитирует действие «нарезать пиццу на куски», выводит соответствующее сообщение;
  - void DisplayInfo() const override – переопределённый метод, выводит информацию о пицце: название, тип теста, вес, итоговая цена;
  - double GetFullPrice() const override – возвращает итоговую цену с учётом типа теста;
- Класс Salad наследуется от Dish:
  - Salad(string name, int weight, double price) – конструктор, передаёт параметры в базовый класс Dish, заправка по умолчанию — OliveOil;
  - string GetDressing() const – возвращает текстовое описание текущей заправки;
  - void ChangeDressing() – переключает вид заправки между OliveOil и Mayonnaise;
  - void TossWithDressing() const – bмитирует действие «перемешать салат с заправкой», выводит соответствующее сообщение;
  - void DisplayInfo() const override – gереопределённый метод, выводит информацию о салате: название, вид заправки, вес, цена;
  - double GetFullPrice() const override – возвращает итоговую цену с учётом заправки.

Исходный код класса Dish представлен ниже:

```
#pragma once

#include <string>
#include <iomanip>
#include <iostream>

class Dish
  {
    public:
    Dish(std::string name, int weight, double price)
    : _name(std::move(name))
    , _weight(weight)
    , _price(price)
    {}

    virtual ~Dish() = default;

    std::string GetName()    const { return _name; }
    int            GetWeight() const { return _weight; }
    double        GetPrice()    const { return _price; }

    virtual void DisplayInfo() const
      {
        std::cout << "Dish: " << _name << ", "
        << _weight << "g, $"
        << std::fixed << std::setprecision(2) << GetFullPrice()
        << std::endl;
      }

    virtual double GetFullPrice() const
      {
        return _price * 1.2;
      }

    private:
    std::string _name;
    int            _weight;
    double        _price;
  };
```

Исходный код класса Pizza представлен ниже:

```
#pragma once

#include <iostream>
#include "dish.h"

enum Dough
  {
    Thin,
    Thick
  };

class Pizza : public Dish
  {
    public:
    Pizza(std::string name, int weight, double price)
    : Dish(name, weight, price)
    {}

    std::string GetDough() const
      {
        return _dough == Dough::Thin ? "thin" : "thick";
      }

    void ChangeDough()
    {
        _dough = _dough == Dough::Thin ? Dough::Thick : Dough::Thin;
      }

    void CutInSlices()
    {
        std::cout << "Cutting the pizza into slices..." << std::endl;
      }

    void DisplayInfo() const override
      {
        std::cout << "Pizza: " << GetName() << ", "
        << GetDough() << " dough, "
        << GetWeight() << "g, $"
        << std::fixed << std::setprecision(2) << GetFullPrice()
```

```
        << std::endl;
      }

    double GetFullPrice() const override
      {
        return _dough == Dough::Thin ? GetPrice() * 1.3 : GetPrice() * 1.5;
      }

    private:
    Dough _dough = Dough::Thick;
  };
```

Исходный код класса Salad представлен ниже:

```
#pragma once

#include <iostream>
#include "dish.h"

enum Dressing
  {
    OliveOil,
    Mayonnaise
  };

class Salad : public Dish
  {
    public:
    Salad(std::string name, int weight, double price)
    : Dish(name, weight, price)
    {}

    std::string GetDressing() const
      {
        return _dressing == Dressing::OliveOil ? "olive oil" : "mayonnaise";
      }

    void ChangeDressing()
    {
        _dressing = _dressing == Dressing::OliveOil
        ? Dressing::Mayonnaise
```

```
        : Dressing::OliveOil;
    }

  void TossWithDressing() const
    {
      std::cout << "Tossing the salad with "
      << GetDressing() << "..." << std::endl;
    }

  void DisplayInfo() const override
    {
      std::cout << "Salad: " << GetName() << ", "
      << GetDressing() << " dressing, "
      << std::to_string(GetWeight()) << "g, $"
      << std::fixed << std::setprecision(2) << GetFullPrice()
      << std::endl;
    }

  double GetFullPrice() const override
    {
      return _dressing == OliveOil ? GetPrice() * 1.4 : GetPrice() * 1.3;
    }

  private:
  Dressing _dressing = Dressing::OliveOil;
};
```

Исходный код программы, демонстрирующее использование классов:

```
#include <iostream>
#include "dish.h"
#include "pizza.h"
#include "salad.h"

int main()
{
    Dish sushi("Philadelphia", 320, 15);
    Pizza pizza("Margarita", 450, 12);
    Salad salad("Caesar", 250, 9);

    std::cout << "-------------------------------" << std::endl;
```

```
  sushi.DisplayInfo();
  std::cout << "Full price: $" << sushi.GetFullPrice() << std::endl;

    std::cout << "--------------------------------" << std::endl;

    pizza.DisplayInfo();
    std::cout << "Full price: $" << pizza.GetFullPrice() << std::endl;
  pizza.ChangeDough();
  std::cout << "Full price: $" << pizza.GetFullPrice() << std::endl;
    pizza.CutInSlices();

    std::cout << "--------------------------------" << std::endl;

    salad.DisplayInfo();
    std::cout << "Full price: $" << salad.GetFullPrice() << std::endl;
  salad.TossWithDressing();
  salad.ChangeDressing();
  std::cout << "Full price: $" << salad.GetFullPrice() << std::endl;
  salad.TossWithDressing();

  std::cout << "--------------------------------" << std::endl;
}
```

Вывод

В ходе выполнения лабораторной работы была разработана и реализована иерархия классов, моделирующая управление ассортиментом блюд в заведении общественного питания. Изучены основные принципы объектно-ориентированного программированя: инкапсуляция, наследование, полиморфизм. Разработано приложение, демонстрирующее применение данных принципов.