МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

кафедра программного обеспечения и администрирования

информационных систем

Отчёт

по лабораторной работе №5

«Программирование полиморфных методов. Абстрактные методы и классы»

по дисциплине

###### «Объектно-ориентированные языки и системы»

|  |  |
| --- | --- |
| Выполнил(а): | студент группы 313  Козявин Максим  Сергеевич |
| Проверил: | к.т.н., доцент  кафедры ПОиАИС  Макаров К.С. |

Курск

2023

**Цель работы:** познакомиться с программированием полиморфных методов, абстрактных методов и классов при объектно-ориентированном подходе на языке программирования C#.

**Индивидуальное задание (вариант 4):**

**Задание 1:**

1. Для определения иерархии классов связать отношением наследования классы для заданного варианта. Из перечисленных классов выбрать один, который будет стоять во главе иерархии. Это абстрактный класс.
2. Построить иерархию классов с использованием языка UML.
3. Определить в классах все необходимые конструкторы.
4. Компонентные данные класса специфицировать как protected.
5. Для добавления объекта в список предусмотреть метод класса, то есть объект сам добавляет себя в список. Например, a. Add() – объект a добавляет себя в список.
6. Список просматривать путём вызова виртуального метода Show каждого объекта.
7. Метод просмотра списка вызывать через класс.

деталь, механизм, изделие, узел;

**Задание 2:**

Создайте абстрактный базовый класс с виртуальной функцией – сумма прогрессии. Создайте производные классы: арифметическая прогрессия и геометрическая прогрессия. Каждый класс имеет два поля типа double. Первое – первый член прогрессии, второе – постоянная разность (для арифметической) и постоянное отношение (для геометрической). Определите функцию вычисления суммы, где параметром является количество элементов прогрессии.

**Задание 3:**

Создайте класс – данные – абстрактный базовый класс. Создайте производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

**Основные теоретические положения**

Полиморфизм в контексте ООП означает, что разные объекты могут реагировать на один и тот же запрос, проявляя разное поведение в зависимости от своего типа. Это позволяет сократить дублирование кода, улучшить читаемость и облегчить расширение программы.

Абстрактный метод не имеет реализации. Он лишь создаёт имя и сигнатуру метода, который должен быть реализован во всех производных классах. Абстрактные классы устанавливают базу для производных классов, но создать объект абстрактного класса нельзя.

**Экспериментальные результаты**

Код программы для решения задачи 1:

internal class Program

{

private static void Main(string[] args)

{

Detail d1 = new Detail("D1");

Mechanism m1 = new Mechanism("M1");

Node n1 = new Node("N1");

Node n2 = new Node("N2");

d1.Add();

m1.Add();

n1.Add();

n2.Add();

Product.Print();

}

}

abstract class Product {

protected string name;

static private List<Product> list = new List<Product>();

public Product(string name) {

this.name = name ?? " ";

}

public void Add() {

list.Add(this);

}

public virtual void Show() {

Console.WriteLine(name);

}

public static void Print() {

foreach (var item in list){

item.Show();

}

}

}

class Node : Product {

public Node(string name) : base(name) {}

override public void Show() {

Console.WriteLine("Node: " + name);

}

}

class Mechanism : Product {

public Mechanism(string name) : base(name) {}

override public void Show() {

Console.WriteLine("Mechanism: " + name);

}

}

class Detail : Product {

public Detail(string name) : base(name) {}

override public void Show() {

Console.WriteLine("Detail: " + name);

}

}

Код программы для решения задачи 2:

internal class Program

{

private static void Main(string[] args)

{

Arithmetic a = new(1, 1);

Geometric g = new(2, 2);

Console.WriteLine(a.Sum(5));

Console.WriteLine(g.Sum(3));

}

}

abstract class Progression {

protected double first;

protected double step;

public Progression (double first, double step) {

this.first = first;

this.step = step;

}

public virtual double Sum(int n) {return 0f;}

}

class Geometric : Progression {

public Geometric(double first, double step) : base(first, step) {}

override public double Sum(int n) {

return (first \* (1-Math.Pow(step, n)))/(1-step);

}

}

class Arithmetic : Progression {

public Arithmetic(double first, double step) : base(first, step) {}

override public double Sum(int n) {

return (2\*first + step\*(n-1))\*n/2;

}

}

Код программы для решения задачи 3:

internal class Program

{

private static void Main(string[] args)

{

Signal s1 = new("s1 content");

Auxiliary a1 = new("a1 content");

Result r1 = new("r1 content");

s1.Process();

s1.Save();

a1.Save();

r1.Save();

Data.PrintSaved();

}

}

abstract class Data {

protected string content;

protected bool isProcessed;

static private List<Data> list = new List<Data>();

public Data(string content) {

this.content = content;

}

public virtual void Print() {

Console.WriteLine(content + ", is processed: " + isProcessed);

}

public virtual void Save() {

Data.list.Add(this);

}

public virtual void Process() {

isProcessed = true;

}

public static void PrintSaved() {

foreach (var item in list){

item.Print();

}

}

}

class Signal : Data {

public Signal(string content) : base(content) {}

override public void Print() {

Console.Write("SIGNAL: ");

base.Print();

}

}

class Result : Data {

public Result(string content) : base(content) {}

override public void Print() {

Console.Write("RESULT: ");

base.Print();

}

}

class Auxiliary : Data {

public Auxiliary(string content) : base(content) {}

override public void Print() {

Console.Write("AUXILIARY: ");

base.Print();

}

}

UML диаграмма к заданию 1:

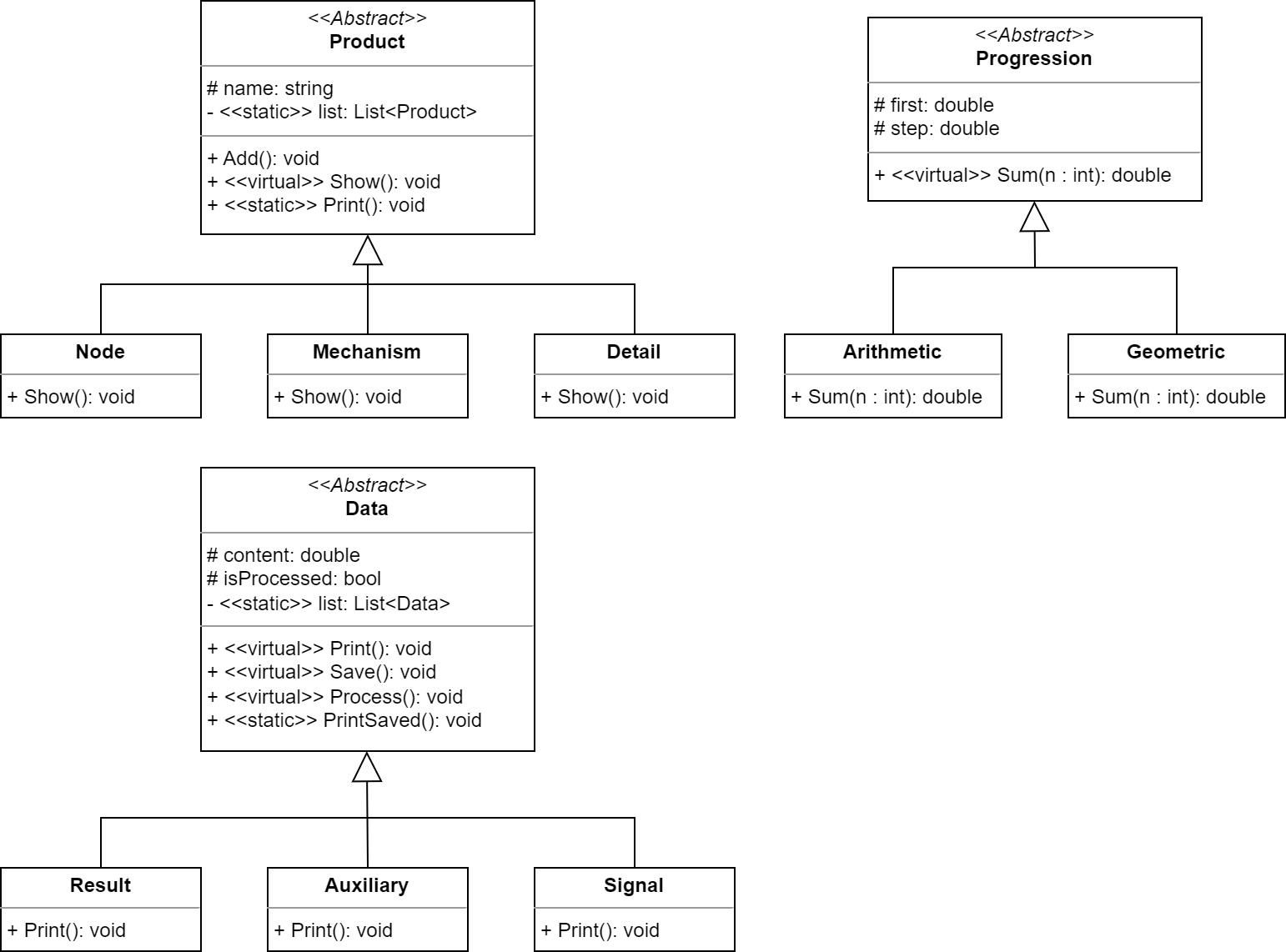


Рисунок 1 – UML диаграмма классов задания 1

UML диаграмма к заданию 2:

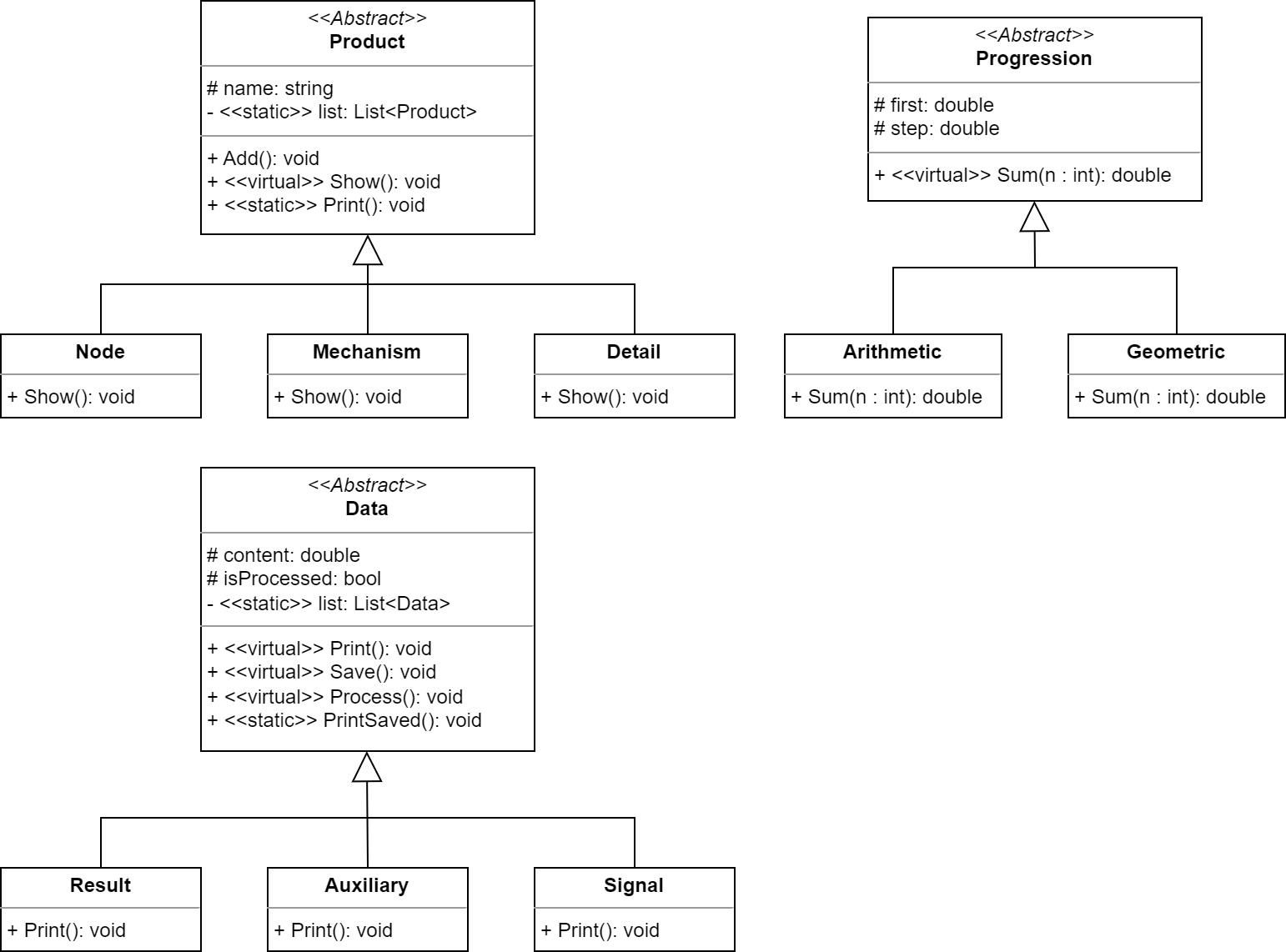


Рисунок 2 – UML диаграмма классов задания 2

UML диаграмма к заданию 3:

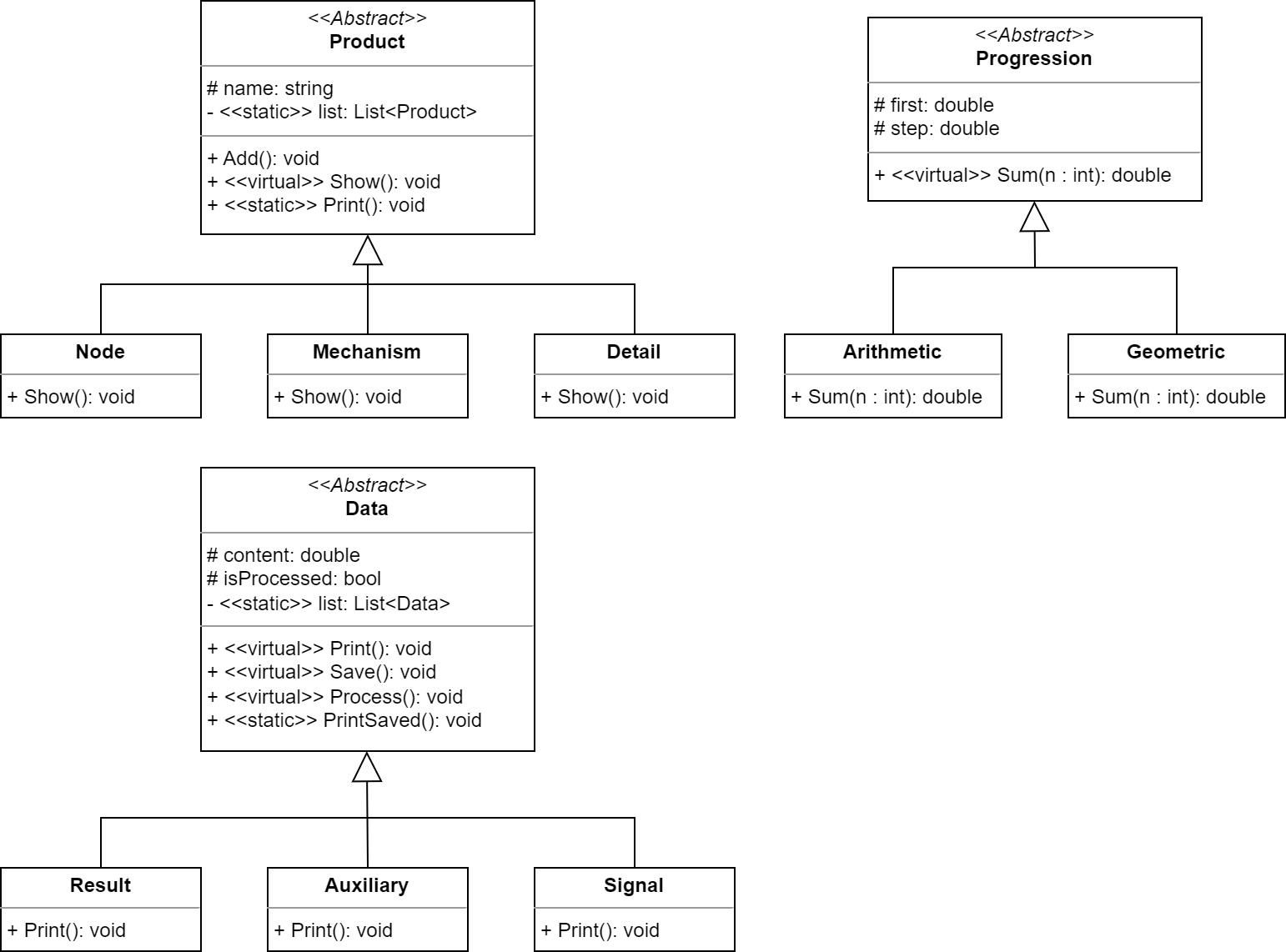


Рисунок 3 – UML диаграмма классов задания 3

Тестирование:

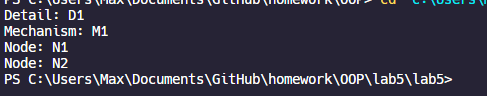


Рисунок 4 – Тест задачи 1

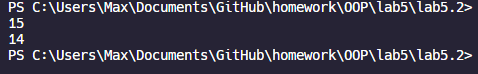


Рисунок 5 – Тест задачи 2

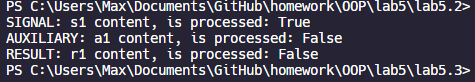


Рисунок 6 – Тест задачи 3

**Вывод:** получил практические навыки по применению механизма полиморфизма при создании объектно-ориентированных программ и закрепил навыки составления UML диаграмм классов.