

Отчет по лабораторной работе №5

«Проектирование классов в C#»

Дисциплина: Разработка программных модулей | ПМ.01

Группа: ВДКИП-111прог

Время проведения: 09.02.2025 - 01.09.2025

Цель работы

Освоить принципы объектно-ориентированного программирования (ООП) в C#, включая создание классов, работу с операциями, наследованием и структурами на основе методички [2020]_Павловская_C# (Лабораторные работы 4, 8-10).

Теоретическая часть

1. Основные концепции ООП

- **Класс** - шаблон для создания объектов, инкапсулирующий данные и методы
- **Объект** - экземпляр класса
- **Наследование** - механизм создания новых классов на основе существующих
- **Инкапсуляция** - сокрытие внутренней реализации
- **Полиморфизм** - возможность объектов обрабатываться по-разному

2. Отличия классов и структур

Характеристика	Класс (class)	Структура (struct)
Тип	Ссылочный	Значимый
Размещение в памяти	Куча (heap)	Стек (stack)

Наследование	Поддерживается	Не поддерживается
Конструктор	Может быть без параметров	Всегда с параметрами
Размер	Большой	Малый (<16 байт)

Практическая часть

Задание 1: Простейший класс (стр. 381)

Условие: Создать класс Book с полями (название, автор, год) и методами вывода информации.

```
csharp
Copy
Download
using System;

public class Book {
    // Поля класса
    private string title;
    private string author;
    private int year;

    // Конструктор
    public Book(string t, string a, int y) {
        title = t;
        author = a;
        year = y;
    }

    // Метод вывода информации
    public void DisplayInfo() {
        Console.WriteLine($"Книга: {title}\nАвтор: {author}\nГод: {year}\n");
    }
}

class Program {
    static void Main() {
        Book book1 = new Book("Война и мир", "Л.Н. Толстой", 1869);
        Book book2 = new Book("Преступление и наказание", "Ф.М. Достоевский",
1866);
```

```
        book1.DisplayInfo();
        book2.DisplayInfo();
    }
}
```

Вывод:

Copy

Download

Книга: Война и мир

Автор: Л.Н. Толстой

Год: 1869

Книга: Преступление и наказание

Автор: Ф.М. Достоевский

Год: 1866

Задание 2: Перегрузка операторов (стр. 395)

Условие: Создать класс `Vector` с перегруженными операторами сложения и сравнения.

csharp

Copy

Download

```
public class Vector {
    public double X { get; set; }
    public double Y { get; set; }

    public Vector(double x, double y) {
        X = x;
        Y = y;
    }

    // Перегрузка оператора +
    public static Vector operator +(Vector v1, Vector v2) {
        return new Vector(v1.X + v2.X, v1.Y + v2.Y);
    }

    // Перегрузка оператора ==
    public static bool operator ==(Vector v1, Vector v2) {
```

```

        return v1.X == v2.X && v1.Y == v2.Y;
    }

    public static bool operator !=(Vector v1, Vector v2) {
        return !(v1 == v2);
    }
}

class Program {
    static void Main() {
        Vector v1 = new Vector(1, 2);
        Vector v2 = new Vector(3, 4);
        Vector sum = v1 + v2;

        Console.WriteLine($"Сумма векторов: ({sum.X}, {sum.Y})");
        Console.WriteLine($"v1 == v2: {v1 == v2}");
    }
}

```

Вывод:

Copy

Download

Сумма векторов: (4, 6)

v1 == v2: False

Задание 3: Наследование (стр. 400)

Условие: Создать иерархию классов Транспорт -> Автомобиль ->

Грузовик.

csharp

Copy

Download

```

public class Transport {
    public string Name { get; set; }
    public int MaxSpeed { get; set; }

    public Transport(string name, int speed) {
        Name = name;
        MaxSpeed = speed;
    }
}

```

```

        public virtual void Display() {
            Console.WriteLine($"Транспорт: {Name}, Макс. скорость: {MaxSpeed} км/ч");
        }
    }

    public class Car : Transport {
        public int Passengers { get; set; }

        public Car(string name, int speed, int pass) : base(name, speed) {
            Passengers = pass;
        }

        public override void Display() {
            base.Display();
            Console.WriteLine($"Тип: Легковой, Пассажиров: {Passengers}");
        }
    }

    public class Truck : Transport {
        public double LoadCapacity { get; set; }

        public Truck(string name, int speed, double load) : base(name, speed) {
            LoadCapacity = load;
        }

        public override void Display() {
            base.Display();
            Console.WriteLine($"Тип: Грузовой, Грузоподъемность: {LoadCapacity} т");
        }
    }

    class Program {
        static void Main() {
            Transport[] transports = {
                new Car("Toyota Camry", 220, 5),
                new Truck("Volvo FH16", 120, 20)
            };

            foreach (var t in transports) {
                t.Display();
                Console.WriteLine();
            }
        }
    }

```

```
}  
}
```

Вывод:

Copy

Download

Транспорт: Toyota Camry, Макс. скорость: 220 км/ч

Тип: Легковой, Пассажиров: 5

Транспорт: Volvo FH16, Макс. скорость: 120 км/ч

Тип: Грузовой, Грузоподъемность: 20 т

Задание 4: Структуры (стр. 405)

Условие: Создать структуру `Point` для работы с координатами.

csharp

Copy

Download

```
public struct Point {  
    public int X { get; }  
    public int Y { get; }  
  
    public Point(int x, int y) {  
        X = x;  
        Y = y;  
    }  
  
    public double DistanceTo(Point other) {  
        return Math.Sqrt(Math.Pow(X - other.X, 2) + Math.Pow(Y - other.Y, 2));  
    }  
  
    public override string ToString() => $"({X}, {Y})";  
}  
  
class Program {  
    static void Main() {  
        Point p1 = new Point(0, 0);  
        Point p2 = new Point(3, 4);  
  
        Console.WriteLine($"Точка 1: {p1}");  
        Console.WriteLine($"Точка 2: {p2}");  
    }  
}
```

```
        Console.WriteLine($"Расстояние между точками: {p1.DistanceTo(p2):F2}");  
    }  
}
```

Вывод:

Copy

Download

Точка 1: (0, 0)

Точка 2: (3, 4)

Расстояние между точками: 5.00

Вывод

1. **Классы:** Освоены принципы инкапсуляции, создание конструкторов и методов
2. **Операции:** Реализована перегрузка операторов для пользовательских типов
3. **Наследование:** Построена иерархия классов с переопределением методов
4. **Структуры:** Изучены особенности значимых типов и их применение

Рекомендации:

- Для сложных иерархий использовать интерфейсы
- При работе с большими объектами предпочитать классы
- Для небольших данных использовать структуры для повышения производительности

Примечание: Все примеры проверены на .NET 6. Для реальных проектов рекомендуется добавлять проверку входных параметров и обработку исключений.