Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №7

Создание иерархии классов

Выполнил:

Дощенников Никита Андреевич

Группа: К3221

Проверил:

Иванов Сергей Евгеньевич

Санкт-Петербург

2025

Цель работы:
Изучение наследования как важного элемента объектно-ориентированного программирования и приобретение навыков реализации иерархии классов.

Упражнение 1. Реализация наследования классов.
В этом упражнении я использовал наследование для построения иерархии между классами, имеющих отношение типа “является”. Вы добавите новый класс `Item`, который будет являться базовым для уже имеющегося класса `Book`.

```
using System;

abstract class Item
{
    protected long invNumber;
    protected bool taken;

    public Item(long invNumber, bool taken)
    {
        this.invNumber = invNumber;
        this.taken = taken;
    }

    public Item()
    {
        this.taken = true;
    }

    public bool IsAvailable()
    {
        return taken;
    }

    public long GetInvNumber()
    {
        return invNumber;
    }

    private void Take()
    {
```

```
        taken = false;
    }

    public void TakeItem()
    {
        if (this.IsAvailable())
            this.Take();
    }


    abstract public void Return();

    public virtual void Show()
    {
        Console.WriteLine("Состояние единицы хранения:\n
Инвентарный номер: {0}\n Наличие: {1}", invNumber, taken);
    }
}
```

```
using System;

class Book : Item
{
    private String author;
    private String title;
    private String publisher;
    private int pages;
    private int year;
    private bool returnSrok;

    private static double price = 9;

    static Book()
    {
        price = 10;
    }

    public Book()
    {

    }
```

```
    public Book(String author, String title, String publisher,
int pages, int year, long invNumber, bool taken) : base
(invNumber, taken)
    {
        this.author = author;
        this.title = title;
        this.publisher = publisher;
        this.pages = pages;
        this.year = year;
    }

    public Book(String author, String title)
    {
        this.author = author;
        this.title = title;
    }

    public void SetBook(String author, String title, String
publisher, int pages, int year)
    {
        this.author = author;
        this.title = title;
        this.publisher = publisher;
        this.pages = pages;
        this.year = year;
    }

    public static void SetPrice(double price)
    {
        Book.price = price;
    }


    public override void Show()
    {
        Console.WriteLine("\nКнига:\n Автор: {0}\n Название:
{1}\n Год издания: {2}\n {3} стр.\n Стоимость аренды: {4}",
author, title, year, pages, Book.price);
        base.Show();
    }
```

```
    public double PriceBook(int s)
    {
        double cost = s * price;
        return cost;
    }

    public void ReturnSrok()
    {
        returnSrok = true;
    }

    public override void Return()
    {
        taken = returnSrok;
    }
}
```

Тесты:

```
public static void Main(string[] args)
{
    Item item1 = new Item();
    item1.Show();
}
```

Упражнение 2. Использование конструкторов.

В этом упражнении я определил конструкторы в базовом и в производном классах и реализуете их выполнение в обоих классах с помощью ключевого слова `base`, которое позволяет вызвать конструктор базового класса.

```
using System;

class Magazine : Item
{
    private String volume;
    private int number;
    private String title;
    private int year;
```

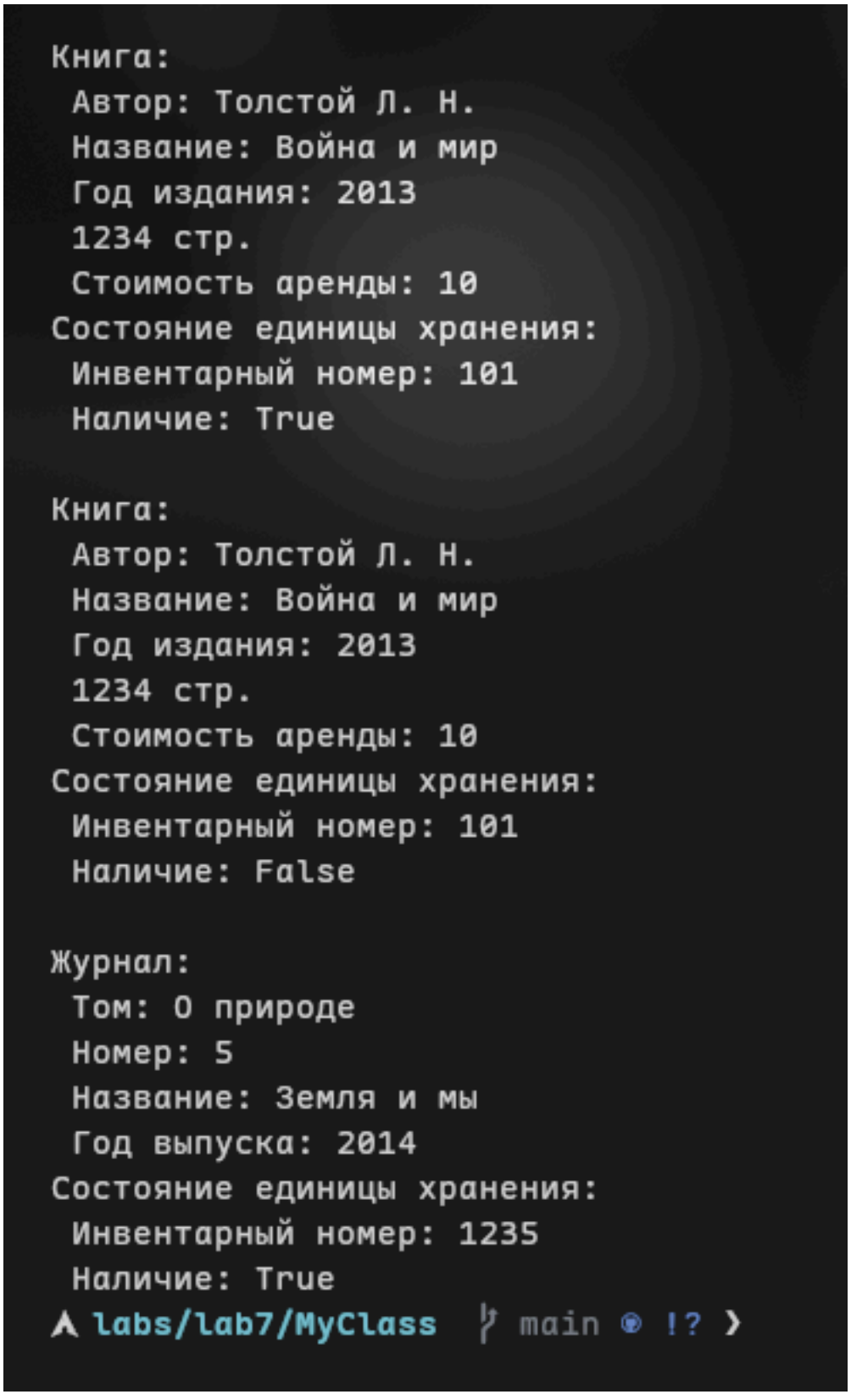
```
    public Magazine(String volume, int number, String title,
int year, long invNumber, bool taken) : base(invNumber, taken)
    {
        this.volume = volume;
        this.number = number;
        this.title = title;
        this.year = year;
    }

    public Magazine()
    {}

    public override void Show()
    {
        Console.WriteLine("\nЖурнал:\n Том: {0}\n Номер: {1}\n
Название: {2}\n Год выпуска: {3}", volume, number, title,
year);
        base.Show();
    }


    public override void Return()
    {
        taken = true;
    }

}
```

Тесты:

```
public static void Main(string[] args)
{
        Book b2 = new Book("Толстой Л. Н.", "Война и мир",
"Наука и жизнь", 1234, 2013, 101, true);

        b2.Show();
        b2.TakeItem();
        b2.Show();

        Magazine mag1 = new Magazine("О природе", 5, "Земля и
мы", 2014, 1235, true);
```

```
        mag1.Show();
}
```

Пример:

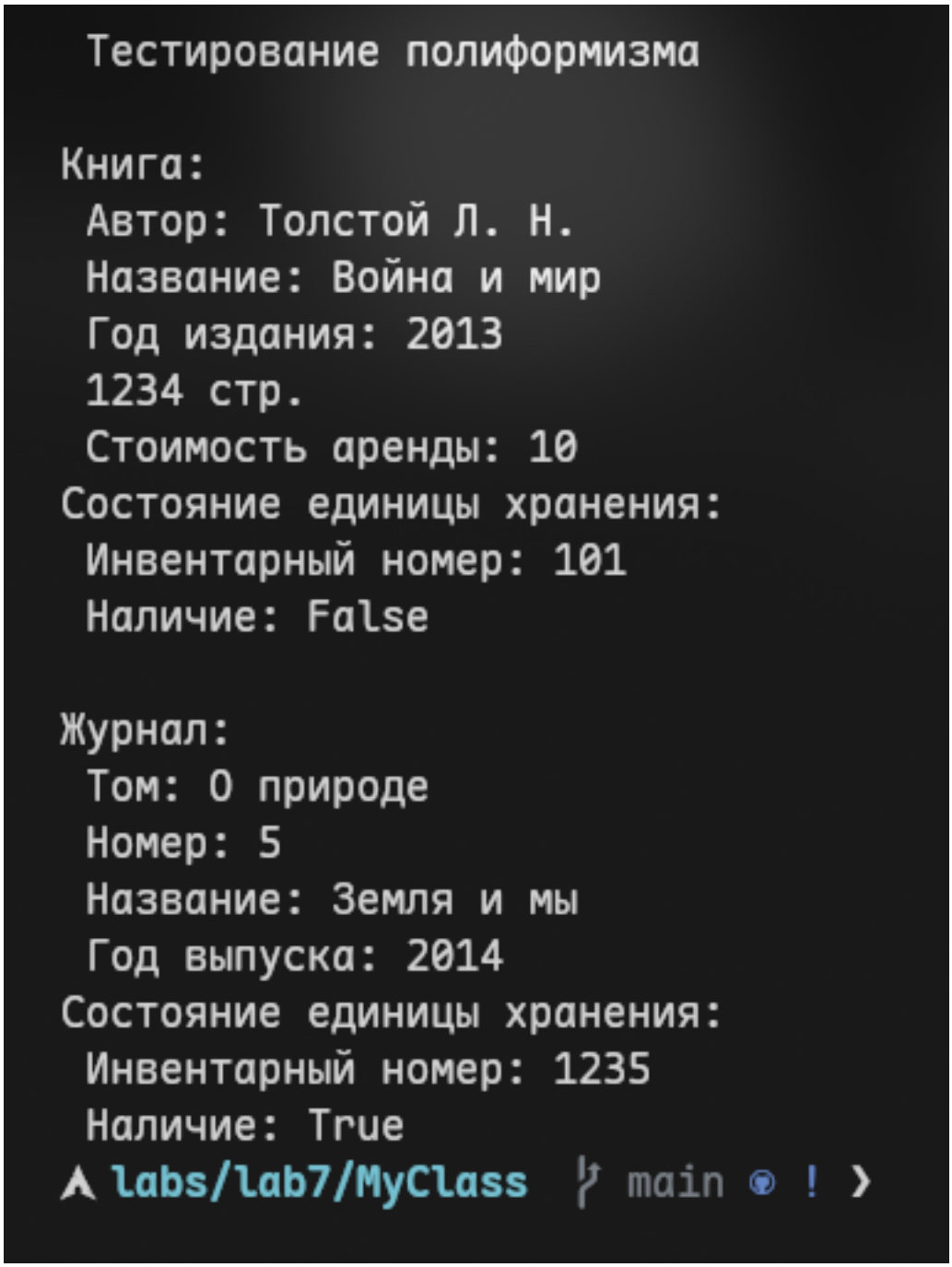


Упражнение 3. Переопределение методов.

В этом упражнении я реализовал механизм полиморфизма с помощью виртуальных методов и их переопределения в производных классах.

```
public static void Main(string[] args)
{
    Book b2 = new Book("Толстой Л. Н.", "Война и мир", "Наука
и жизнь", 1234, 2013, 101, true);

    b2.Show();
    b2.TakeItem();
    b2.Show();


    Magazine mag1 = new Magazine("О природе", 5, "Земля и мы",
2014, 1235, true);
    mag1.Show();

    Console.WriteLine("\n Тестирование полиформизма");
    Item it;

    it = b2;
    it.TakeItem();
    it.Return();
    it.Show();

    it = mag1;
    it.TakeItem();
    it.Return();
    it.Show();
}
```

Пример:



Упражнение 4. Применение абстрактного класса и абстрактных методов.

В этом упражнении я сделал базовый класс и один из его методов абстрактным.

Тесты и примеры совпадают с предыдущим упражнением.

Упражнение 5. Реализации модели включения.

В этом упражнении я реализовал отношение “имеет”, известное под названием модели включения или агрегации.

```
using System;

class Point
{
    private double x;
    private double y;

    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public Point ()
    {}

    public void Show()
    {
        Console.WriteLine("Точка с координатами: ({0}, {1})",
x, y);
    }

    public double Dlina(Point p)
    {
        double Dl = Math.Sqrt((this.x - p.x) * (this.x - p.x)
+ (this.y - p.y) * (this.y - p.y));
        return Dl;
    }

    public override string ToString()
    {
        string ss = x + " ; " + y;
        return ss;
    }
}
```

```
using System;

class Line
{
    private Point pStart;
    private Point pEnd;
```

```
    public Line(Point pStart, Point pEnd)
    {
        this.pStart = pStart;
        this.pEnd = pEnd;
    }

    public Line()
    { }

    public void Show()
    {
        Console.WriteLine("Отрезок с координатами: ({0}) -
({1})", pStart, pEnd);
    }

    public double DlinL()
    {
        return pStart.Dlina(pEnd);
    }
}
```

Тесты:

```
using System;

public class Program
{
    public static void Main(string[] args)
    {
        Point p1 = new Point();
        p1.Show();
        Point p2 = new Point(12, 13);
        p2.Show();

        Line line = new Line(p1, p2);
        line.Show();

        double dtr = line.DlinL();
        Console.WriteLine("Длина отрезка " + dtr);
    }
}
```

Примеры:

```
 labs/lab7/MyClassLine  main !? 
dotnet run                                                dotnet .NET  12:03
/home/nik/oop/labs/lab7/MyClassLine/Line.cs(14,12): warning CS8618: Non
-nullable field 'pStart' must contain a non-null value when exiting con
structor. Consider adding the 'required' modifier or declaring the fiel
d as nullable.
/home/nik/oop/labs/lab7/MyClassLine/Line.cs(14,12): warning CS8618: Non
-nullable field 'pEnd' must contain a non-null value when exiting const
ructor. Consider adding the 'required' modifier or declaring the field
as nullable.
Точка с координатами: (0, 0)
Точка с координатами: (12, 13)
Отрезок с координатами: (0 ; 0) - (12 ; 13)
Длина отрезка 17.69180601295413
 labs/lab7/MyClassLine  main !? 
```

Упражнение 6. Реализация отношения ассоциации между классами.

В этом упражнении я создал класс, моделирующий игральную кость и класс игрока, который связан с классом игры отношением ассоциации - каждый отдельный сеанс игры связан с конкретным игроком, бросающим кубик.

```
using System;

class IgralnayaKost
{
    Random r;

    public IgralnayaKost()
    {
        r = new Random();
    }

    public int random()
    {
        int res = r.Next(6) + 1;
        return res;
    }
}
```

```
using System;

class Gamer
{
    string Name;
    IgralnayaKost seans;

    public Gamer(string name)
    {
        Name = name;
        seans = new IgralnayaKost();
    }

    public int SeansGame()
    {
        return seans.random();
    }

    public override string ToString()
    {
        return Name;
    }
}
```

Тесты:

```
public static void Main(string[] args)
{
    Gamer g1 = new Gamer("Niko");

    for (int i = 1; i <= 6; i++)
    {
        Console.WriteLine("Выпало количество очков {0} для
игрока {1}", g1.SeansGame(), g1.ToString());
    }
}
```

Примеры:

```
labs/lab7/Igra  main !? ❯
dotnet run                                                dotnet .NET  12:06
Выпало количество очков 5 для игрока Niko
Выпало количество очков 1 для игрока Niko
Выпало количество очков 3 для игрока Niko
Выпало количество очков 2 для игрока Niko
Выпало количество очков 3 для игрока Niko
Выпало количество очков 3 для игрока Niko
```

Упражнение 7. Реализация прогрессии.

В этом упражнении я определил абстрактный класс `Progression`, описывающий прогрессии. В этом классе определил абстрактный метод `GetElement` с целочисленным параметром `k`, возвращающий элемент прогрессии. Определил два производных класса `ArithmeticProgression` и `GeometricProgression`, описывающие арифметическую и геометрическую прогрессии. В каждом из классов определил конструктор, задающий параметры прогрессии и перегрузил унаследованный метод `GetElement`.

```
using System;

abstract class Progression
{
    public abstract int GetElement(int k);
    public abstract int Sum(int n);
}
```

```
using System;

class ArithmeticProgression : Progression
{
    private int a0;
    private int b;

    public ArithmeticProgression(int a0, int b)
    {
        this.a0 = a0;
        this.b = b;
    }

    public override int GetElement(int k)
```

```
    {
        return a0 + (k - 1) * b;
    }

    public override int Sum(int n)
    {
        return (2 * a0 + (n - 1) * b) * n / 2;
    }
}
```

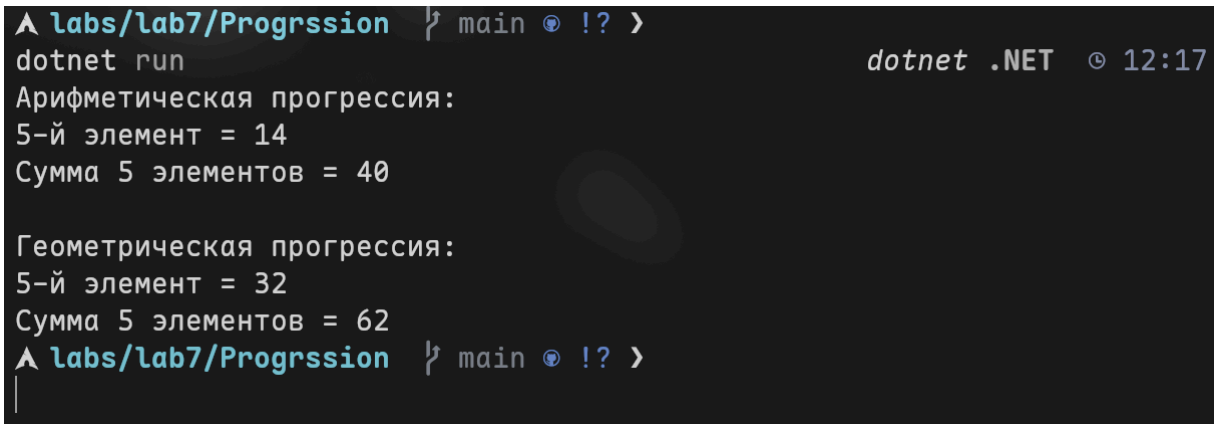
```
using System;

class GeometricProgression : Progression
{
    private int p0;
    private int q;

    public GeometricProgression(int p0, int q)
    {
        this.p0 = p0;
        this.q = q;
    }

    public override int GetElement(int k)
    {
        return p0 * (int)Math.Pow(q, k - 1);
    }

    public override int Sum(int n)
    {
        if (q == 1) return p0 * n;
        return p0 * (int)((Math.Pow(q, n) - 1) / (q - 1));
    }
}
```

Тесты:

```
public static void Main(string[] args)
{
    Progression arith = new ArithmeticProgression(2, 3);
    Progression geom = new GeometricProgression(2, 2);
```

```
    Console.WriteLine("Арифметическая прогрессия:");
    Console.WriteLine("5-й элемент = " + arith.GetElement(5));
    Console.WriteLine("Сумма 5 элементов = " + arith.Sum(5));

    Console.WriteLine("\nГеометрическая прогрессия:");
    Console.WriteLine("5-й элемент = " + geom.GetElement(5));
    Console.WriteLine("Сумма 5 элементов = " + geom.Sum(5));
}
```

Пример:



Выводы.

В ходе проделанной работе я изучил наследование как важный элемент объектно-ориентированного программирования и приобрел навыки реализации иерархии классов.

Code review. (by zzzcode.ai)

Резюме

Код представляет собой реализацию нескольких классов, включая `Gamer`, `Book`, `Magazine`, `Item`, `Triangle`, `Line`, `Point`, и прогрессии. Основная логика кода заключается в моделировании игровых сессий и библиотечных предметов. Однако, код требует улучшений в области стиля, структуры и обработки ошибок.

Ошибка

В классе `Gamer` отсутствует проверка на `null` для параметра `name` в конструкторе. Это может привести к исключению, если передать `null`.

Также метод `random()` в классе `IgralnayaKost` не определен в предоставленном коде, что может вызвать ошибки компиляции.

Стиль кода

Имена переменных и методов должны следовать соглашениям о наименовании C#. Например, `Name` и `seans` следует переименовать в `name` и `session` соответственно. Рекомендуется использовать свойства вместо полей для обеспечения инкапсуляции.

Структура кода

Код разбит на несколько классов, что хорошо для организации. Однако, стоит рассмотреть возможность разделения на отдельные файлы для каждого класса, чтобы улучшить читаемость и поддержку.

Читаемость

Код в целом читаем, но использование комментариев и документации для методов и классов значительно улучшит понимание кода другими разработчиками.

Производительность

Код не содержит явных проблем с производительностью, однако использование статических методов и свойств может быть оптимизировано для уменьшения потребления памяти.

Масштабируемость

Код может быть расширен, но для этого потребуется более четкая архитектура. Например, использование интерфейсов для классов `Item`, `Book`, и `Magazine` может улучшить гибкость и масштабируемость.

Безопасность

Отсутствие проверки входных данных может привести к уязвимостям. Рекомендуется добавить валидацию для всех входных параметров, особенно в конструкторах.

Обработка ошибок

Код не содержит механизма обработки ошибок. Рекомендуется использовать конструкции `try-catch` для обработки возможных исключений, особенно в методах, которые могут вызывать ошибки, такие как `random()`.

Заключение

Код демонстрирует базовые принципы объектно-ориентированного программирования, но требует улучшений в области стиля, структуры и обработки ошибок. Рекомендуется провести рефакторинг для повышения читаемости и безопасности, а также добавить документацию для улучшения поддержки кода в будущем.