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

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №9

Применение делегатов и событий

Выполнил:

Дощенников Никита Андреевич

Группа: К3221

Проверил:

Иванов Сергей Евгеньевич

Санкт-Петербург

2025

Цель работы:
Научиться работать с делегатами и событиями и применять их.

Упражнение 1. Использование делегата при вызове метода.
В этом упражнении я реализовал в классе `Book` возможность вызова метода через делегат.

```
using System;

class Operation
{
    public static void PrintTitle(Book b)
    {
        b.Show();
    }
}
```

```
using System;

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

    private static double price = 9;

    static Book()
    {
        price = 10;
    }

    public Book()
    {

    }


    public Book(String author, String title, String publisher,
```

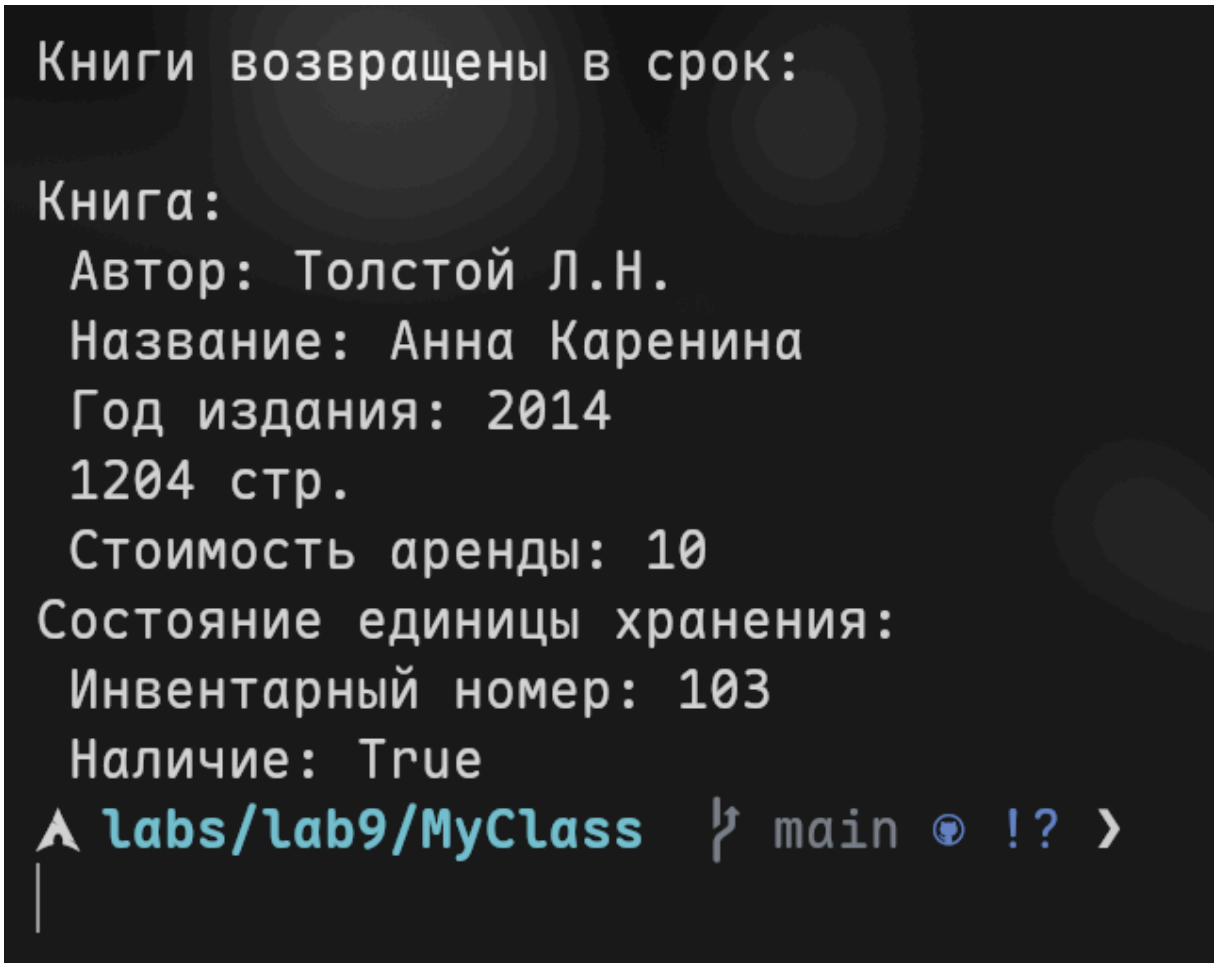
```
int pages, int year, long invNumber, bool taken) : base
(invNumber, taken)
    {
        this.author = author;
        this.title = title;
        this.publisher = publisher;
        this.pages = pages;
        this.year = year;
    }

    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 override void Return()
    {
        taken = ReturnSrok;
    }

    public delegate void ProcessBookDelegate(Book book);

    public bool ReturnSrok { get; set; }

    public void ProcessPaperbackBooks(ProcessBookDelegate
processBook)
    {
        if (ReturnSrok)
        processBook(this);
    }
}
```

Тесты:

```
public static void Main(string[] args)
{
    Book b4 = new Book("Толстой Л.Н.", "Анна Каренина",
"Знание", 1204, 2014, 103, true);
    Book b5 = new Book("Неш Т", "Программирование для
профессионалов", "Вильямс", 1200, 2014, 108, true);

    b4.ReturnSrok = true;
    b5.ReturnSrok = false;

    Console.WriteLine("\nКниги возвращены в срок:");
    b4.ProcessPaperbackBooks(Operation.PrintTitle);
    b5.ProcessPaperbackBooks(Operation.PrintTitle);
}
```

Пример:



Упражнение 2. Работа с событиями.

В этом упражнении я в классе `Book` объявил событие “возвращение книги в срок”, и объекты этого класса смогли уведомлять объекты других классов о данном событии.

```
using System;

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

    private static double price = 9;
    private bool returnSrok = false;
```

```
    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 override void Return()
    {
        taken = ReturnSrok;
    }

    public delegate void ProcessBookDelegate(Book book);

    public void ProcessPaperbackBooks(ProcessBookDelegate
processBook)
    {
        if (ReturnSrok)
        processBook(this);
   }

    public static event ProcessBookDelegate RetSrok;

    public bool ReturnSrok
    {
        get
        {
            return returnSrok;
        }
        set
        {
            returnSrok = value;
```

```
            if (ReturnSrok == true)
                RetSrok(this);
        }
    }

    public override string ToString()
    {
        string str = this.title + ", " + this.author + " Инв.
номер " + this.invNumber;
        return str;
    }
}
```

```
using System;

class Operation
{
    public static void PrintTitle(Book b)
    {
        b.Show();
    }

    public static void MetodObrabotchik(Book b)
    {
        Console.WriteLine("Книга {0} сдана в срок.",
b.ToString());
    }
}
```

Тесты:

```
public static void Main(string[] args)
{
    Book b4 = new Book("Толстой Л.Н.", "Анна Каренина",
"Знание", 1204, 2014, 103, true);
    Book b5 = new Book("Неш Т", "Программирование для
профессионалов", "Вильямс", 1200, 2014, 108, true);

    Book.RetSrok += new
Book.ProcessBookDelegate(Operation.MetodObrabotchik);
```

```
    b4.ReturnSrok = true;
    b5.ReturnSrok = true;

    Console.WriteLine("\nКниги возвращены в срок:");
    b4.ProcessPaperbackBooks(Operation.PrintTitle);
    b5.ProcessPaperbackBooks(Operation.PrintTitle);
}
```

Пример:

Упражнение 3. Реализация события.

В этом упражнении в проекте `IgralnayaKost` я реализовал возникновение события “выпало максимальное количество очков” при броске игрального кубика.

```
using System;

class Gamer
{
    string Name;
    IgralnayaKost seans;

    public Gamer(string name)
    {
        Name = name;
```

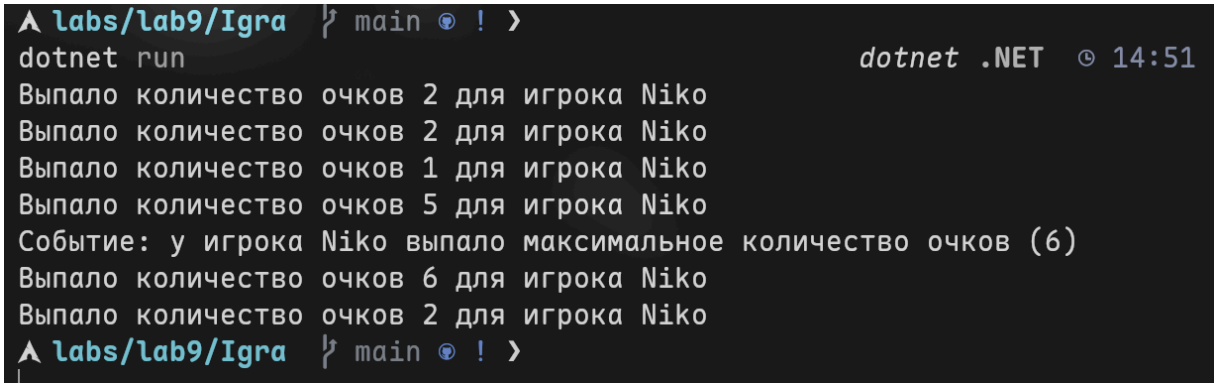
```
        seans = new IgralnayaKost();
        seans.MaxRolled += OnMaxRolled;
    }

    void OnMaxRolled(int value)
    {
        Console.WriteLine("Событие: у игрока {0} выпало
максимальное количество очков ({1})", Name, value);
    }

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

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

```
using System;

class IgralnayaKost
{
    readonly Random r;
    public event Action<int> MaxRolled;

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

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

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());
    }
}
```

Пример:



Упражнение 4. Иерархия классов учебного центра.

В этом задании я реализовал иерархию классов учебного центра.

```
using System;

class Administrator : Person, IEmployee
{
    public string Laboratory { get; }
    public string Department => Laboratory;
    public string Position => "Администратор";
    public int Experience { get; }
    public decimal BaseRate { get; }

    public Administrator(string lastName, DateTime birthDate,
string laboratory, int experience, decimal baseRate)
        : base(lastName, birthDate)
    {
        Laboratory = laboratory;
        Experience = experience;
```

```
        BaseRate = baseRate;
    }

    public override void Show()
    {
        Console.WriteLine("Администратор: {0}, лаб.: {1},
стаж: {2} лет, возраст: {3}", LastName, Laboratory,
Experience, Age());
    }

    public decimal GetMonthlyPay()
    {
        return BaseRate * (1 + 0.05m * Experience);
    }
}
```

```
using System;

interface IEmployee
{
    string Department { get; }
    string Position { get; }
    int Experience { get; }
    decimal GetMonthlyPay();
}
```

```
using System;

class Manager : Person, IEmployee
{
    public string Faculty { get; }
    public string Role { get; }
    public int Experience { get; }
    public decimal BaseRate { get; }
    public string Department => Faculty;
    public string Position => Role;

    public Manager(string lastName, DateTime birthDate, string
faculty, string role, int experience, decimal baseRate)
        : base(lastName, birthDate)
    {
```

```
        Faculty = faculty;
        Role = role;
        Experience = experience;
        BaseRate = baseRate;
    }

    public override void Show()
    {
        Console.WriteLine("Менеджер: {0}, факультет: {1},
должность: {2}, стаж: {3} лет, возраст: {4}", LastName,
Faculty, Role, Experience, Age());
    }

    public decimal GetMonthlyPay()
    {
        return BaseRate * (1 + 0.06m * Experience);
    }
}
```

```
using System;

abstract class Person
{
    public string LastName { get; }
    public DateTime BirthDate { get; }

    protected Person(string lastName, DateTime birthDate)
    {
        LastName = lastName;
        BirthDate = birthDate;
    }

    public int Age()
    {
        var today = DateTime.Today;
        int a = today.Year - BirthDate.Year;
        if (BirthDate.Date > today.AddYears(-a)) a--;
        return a;
    }

    public virtual void Show()
    {
```

```
        Console.WriteLine("{0}, возраст: {1}", LastName,
Age());
    }
}
```

```
using System;

class Student : Person
{
    public string Faculty { get; }
    public int Course { get; }

    public Student(string lastName, DateTime birthDate, string
faculty, int course)
        : base(lastName, birthDate)
    {
        Faculty = faculty;
        Course = course;
    }

    public override void Show()
    {
        Console.WriteLine("Студент: {0}, факультет: {1}, курс:
{2}, возраст: {3}", LastName, Faculty, Course, Age());
    }
}
```

```
using System;

class Teacher : Person, IEmployee
{
    public string Faculty { get; }
    public string Rank { get; }
    public int Experience { get; }
    public decimal BaseRate { get; }
    public string Department => Faculty;
    public string Position => Rank;

    public Teacher(string lastName, DateTime birthDate, string
faculty, string rank, int experience, decimal baseRate)
        : base(lastName, birthDate)
```

```
    {
        Faculty = faculty;
        Rank = rank;
        Experience = experience;
        BaseRate = baseRate;
    }

    public override void Show()
    {
        Console.WriteLine("Преподаватель: {0}, факультет: {1},
должность: {2}, стаж: {3} лет, возраст: {4}", LastName,
Faculty, Rank, Experience, Age());
    }

    public decimal GetMonthlyPay()
    {
        decimal mult = Rank.ToLower().Contains("проф") ?
1.4m : Rank.ToLower().Contains("доцент") ? 1.2m : 1.0m;
        return BaseRate * mult * (1 + 0.04m * Experience);
    }
}
```

Тесты:

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main(string[] args)
    {
        var people = new List<Person>
        {
            new Administrator("Иванова", new DateTime(1985, 3,
12), "Лаборатория сетей", 8, 900m),
            new Student("Петров", new DateTime(2005, 11, 2),
"ФКТИ", 2),
            new Teacher("Сидоров", new DateTime(1979, 6, 25),
"ФКТИ", "Доцент", 12, 1400m),
            new Manager("Кузнецова", new DateTime(1990, 1, 5),
"ФКТИ", "Менеджер проектов", 6, 1100m),
```

```
            new Student("Орлова", new DateTime(2003, 4, 17),
"ФПМИ", 4)
        };

        foreach (var p in people) p.Show();

        Console.WriteLine();
        int from = 20, to = 40;
        var inRange = people.Where(p => p.Age() >= from &&
p.Age() <= to);
        foreach (var p in inRange) p.Show();

        Console.WriteLine();
        foreach (var e in people.OfType<IEmployee>())
            Console.WriteLine("{0}: {1}, подразделение: {2},
оклад: {3:F2}", e.GetType().Name, e.Position, e.Department,
e.GetMonthlyPay());
    }
}
```

Пример:

```
 labs/lab9/LearningCenter  main  !? ❯
dotnet run                                   dotnet .NET  14:59
Администратор: Иванова, лаб.: Лаборатория сетей, стаж: 8 лет, возраст: 40
Студент: Петров, факультет: ФКТИ, курс: 2, возраст: 19
Преподаватель: Сидоров, факультет: ФКТИ, должность: Доцент, стаж: 12 лет, возраст: 46
Менеджер: Кузнецова, факультет: ФКТИ, должность: Менеджер проектов, стаж: 6 лет, возраст: 35
Студент: Орлова, факультет: ФПМИ, курс: 4, возраст: 22

Администратор: Иванова, лаб.: Лаборатория сетей, стаж: 8 лет, возраст: 40
Менеджер: Кузнецова, факультет: ФКТИ, должность: Менеджер проектов, стаж: 6 лет, возраст: 35
Студент: Орлова, факультет: ФПМИ, курс: 4, возраст: 22

Administrator: Администратор, подразделение: Лаборатория сетей, оклад: 1260.00
Teacher: Доцент, подразделение: ФКТИ, оклад: 2486.40
Manager: Менеджер проектов, подразделение: ФКТИ, оклад: 1496.00
 labs/lab9/LearningCenter  main  !? ❯
```

Вывод.

В ходе проделанной работы я научился работать с делегатами и событиями и применять их.

Code review. (by zzzcode.ai)

Резюме

Код представляет собой реализацию игры с использованием классов для игроков и игральных костей. Основные функции включают генерацию случайных значений и обработку событий. В целом, код хорошо структурирован, но требует некоторых улучшений в области стиля и читаемости.

Ошибка

В методе `random()` класса `IgralnayaKost` название метода не соответствует стандартам именования в C#. Рекомендуется переименовать его в `Roll()` для лучшего понимания.

Стиль кода

Код в целом следует стандартам C#, однако есть некоторые моменты, которые можно улучшить: Имена переменных и методов должны быть написаны в стиле `PascalCase`. Необходимо использовать явные модификаторы доступа (например, `public`, `private`) для всех членов класса.

Структура кода

Структура классов логична, однако стоит рассмотреть возможность разделения на отдельные файлы для каждого класса, чтобы улучшить организацию проекта и упростить навигацию.

Читаемость

Читаемость кода можно улучшить, добавив комментарии к методам и классам, а также следуя единообразному стилю именования. Например, использование `seans` как имени переменной может быть неочевидным для других разработчиков.

Производительность

Код работает эффективно для текущих задач, однако стоит обратить внимание на создание экземпляра `Random` в каждом вызове. Рекомендуется создать один экземпляр `Random` и использовать его повторно.

Масштабируемость

Код легко масштабируется, так как добавление новых типов игроков или игральных костей не требует значительных изменений в

существующей логике. Однако стоит учитывать возможность добавления новых функций и их влияние на текущую архитектуру.

Безопасность

Код не содержит явных уязвимостей, однако стоит учитывать обработку исключений, особенно в методах, которые могут вызывать ошибки, например, при работе с пользовательским вводом.

Обработка ошибок

В текущем коде отсутствует обработка ошибок. Рекомендуется добавить обработку исключений, чтобы предотвратить сбои программы в случае неожиданных ситуаций.

Заключение

Код представляет собой хорошую основу для игры, но требует некоторых улучшений в области стиля, читаемости и обработки ошибок. Рекомендуется внести предложенные изменения для повышения качества и удобства работы с кодом.