**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Синергия |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | Информационные системы и программирование |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по лабораторной работе №5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  | Линейные программы | | |
|  |  | (наименование темы) | | |
|  |  |  | | |
| **по дисциплине** | | |  | Разработка модулей ПО для компьютерных система |
|  | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Старчик Артемий Филиппович |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ДКИП-206ПРОГ |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | Сибирев Иван Валерьевич |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва 2024 г.**

Лабораторная работа 4. Простейшие классы.

Вариант 1

Описать класс, реализующий десятичный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения, - и свойство, позволяющее получить его текущее состояние. При выходе за границы диапазона выбрасываются исключения. Написать программу, демонстрирующую все разработанные элементы класса

Листинг кода:

using System;

class DecimalCounter

{

private int \_currentValue;

private readonly int \_minValue;

private readonly int \_maxValue;

// Конструктор по умолчанию (0-100)

public DecimalCounter() : this(0, 0, 100) { }

// Основной конструктор

public DecimalCounter(int initialValue, int minValue, int maxValue)

{

if (minValue > maxValue)

throw new ArgumentException("Минимальное значение не может быть больше максимального");

if (initialValue < minValue || initialValue > maxValue)

throw new ArgumentOutOfRangeException(nameof(initialValue),

"Начальное значение должно быть в пределах диапазона");

\_currentValue = initialValue;

\_minValue = minValue;

\_maxValue = maxValue;

}

// Свойство для получения текущего значения

public int CurrentValue => \_currentValue;

// Метод для увеличения значения на 1

public void Increment()

{

if (\_currentValue >= \_maxValue)

throw new InvalidOperationException("Достигнуто максимальное значение счетчика");

\_currentValue++;

}

// Метод для уменьшения значения на 1

public void Decrement()

{

if (\_currentValue <= \_minValue)

throw new InvalidOperationException("Достигнуто минимальное значение счетчика");

\_currentValue--;

}

// Переопределение ToString для удобного вывода

public override string ToString()

{

return $"Текущее значение: {\_currentValue} (Диапазон: {\_minValue}-{\_maxValue})";

}

}

class Program

{

static void Main()

{

Console.WriteLine("Демонстрация работы десятичного счетчика\n");

try

{

// 1. Создание счетчика с параметрами по умолчанию (0-100)

Console.WriteLine("1. Счетчик с настройками по умолчанию:");

var defaultCounter = new DecimalCounter();

Console.WriteLine(defaultCounter);

// 2. Увеличение счетчика по умолчанию

Console.WriteLine("\nУвеличиваем счетчик 5 раз:");

for (int i = 0; i < 5; i++)

{

defaultCounter.Increment();

Console.WriteLine(defaultCounter);

}

// 3. Создание счетчика с произвольными параметрами

Console.WriteLine("\n2. Счетчик с произвольными параметрами (5, 0-10):");

var customCounter = new DecimalCounter(5, 0, 10);

Console.WriteLine(customCounter);

// 4. Уменьшение произвольного счетчика

Console.WriteLine("\nУменьшаем счетчик 3 раза:");

for (int i = 0; i < 3; i++)

{

customCounter.Decrement();

Console.WriteLine(customCounter);

}

// 5. Попытка выйти за границы диапазона

Console.WriteLine("\nПытаемся выйти за границы диапазона:");

try

{

for (int i = 0; i < 15; i++)

{

customCounter.Increment();

Console.WriteLine(customCounter);

}

}

catch (InvalidOperationException ex)

{

Console.WriteLine($"\nОшибка: {ex.Message}");

}

// 6. Попытка создать невалидный счетчик

Console.WriteLine("\nПытаемся создать счетчик с невалидными параметрами:");

try

{

var invalidCounter = new DecimalCounter(15, 10, 20);

invalidCounter = new DecimalCounter(5, 10, 20); // Это вызовет исключение

}

catch (ArgumentException ex)

{

Console.WriteLine($"Ошибка создания счетчика: {ex.Message}");

}

}

catch (Exception ex)

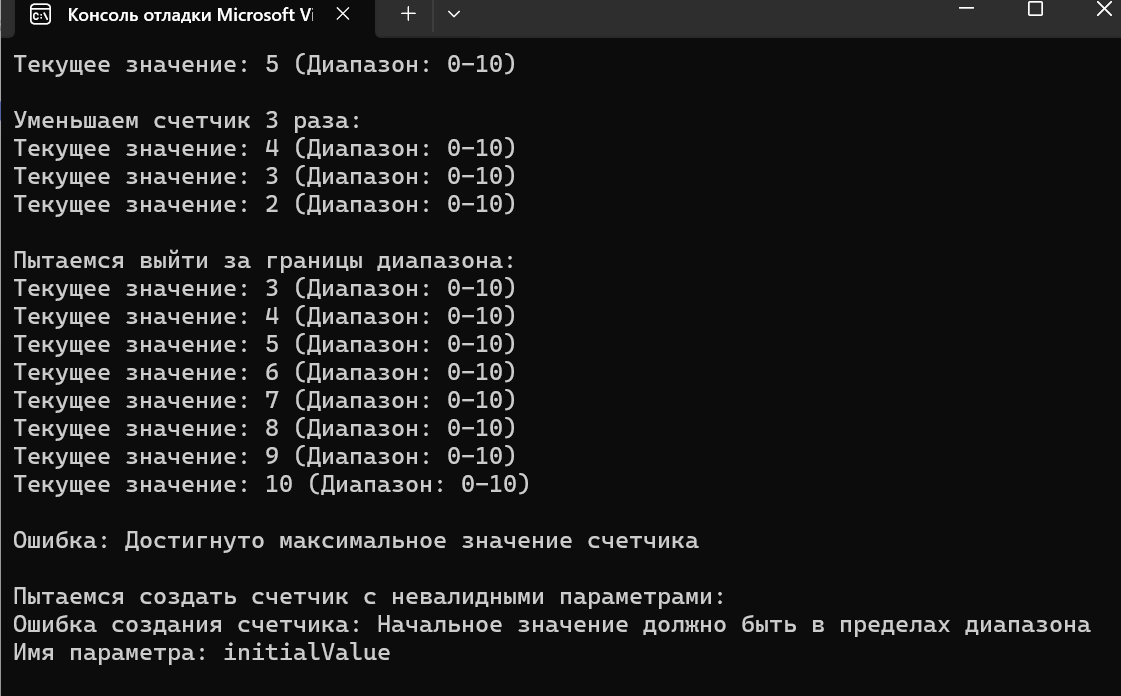
{

Console.WriteLine($"Неожиданная ошибка: {ex.Message}");

}

}

}



Лабораторная работа 8. Классы и операции.

Вариант 1

Описать класс для работы с одномерным массивом целых чисел (вектором). Обеспечить следующие возможности:

* задание произвольных целых границ индексов при создании объекта;
* обращение к отдельному элементу массива с контролем выхода за пределы массива;
* выполнение операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов;
* выполнение операций умножения и деления всех элементов массива на скаляр;
* вывод на экран элемента массива по заданному индексу и всего массива. Написать программу, демонстрирующую все разработанные элементы класса

листинг кода:

using System;

class IntegerVector

{

private readonly int[] \_data;

private readonly int \_lowerBound;

// Конструктор с произвольными границами индексов

public IntegerVector(int lowerBound, int upperBound)

{

if (lowerBound > upperBound)

throw new ArgumentException("Нижняя граница не может быть больше верхней");

\_lowerBound = lowerBound;

\_data = new int[upperBound - lowerBound + 1];

}

// Индексатор с контролем границ

public int this[int index]

{

get

{

CheckIndex(index);

return \_data[index - \_lowerBound];

}

set

{

CheckIndex(index);

\_data[index - \_lowerBound] = value;

}

}

// Проверка индекса

private void CheckIndex(int index)

{

if (index < \_lowerBound || index >= \_lowerBound + \_data.Length)

throw new IndexOutOfRangeException($"Индекс {index} выходит за границы массива");

}

// Поэлементное сложение векторов

public static IntegerVector operator +(IntegerVector a, IntegerVector b)

{

if (a == null || b == null)

throw new ArgumentNullException("Вектор не может быть null");

if (a.\_lowerBound != b.\_lowerBound || a.\_data.Length != b.\_data.Length)

throw new ArgumentException("Векторы должны иметь одинаковые границы индексов");

var result = new IntegerVector(a.\_lowerBound, a.\_lowerBound + a.\_data.Length - 1);

for (int i = 0; i < a.\_data.Length; i++)

{

result.\_data[i] = a.\_data[i] + b.\_data[i];

}

return result;

}

// Поэлементное вычитание векторов

public static IntegerVector operator -(IntegerVector a, IntegerVector b)

{

if (a == null || b == null)

throw new ArgumentNullException("Вектор не может быть null");

if (a.\_lowerBound != b.\_lowerBound || a.\_data.Length != b.\_data.Length)

throw new ArgumentException("Векторы должны иметь одинаковые границы индексов");

var result = new IntegerVector(a.\_lowerBound, a.\_lowerBound + a.\_data.Length - 1);

for (int i = 0; i < a.\_data.Length; i++)

{

result.\_data[i] = a.\_data[i] - b.\_data[i];

}

return result;

}

// Умножение вектора на скаляр

public static IntegerVector operator \*(IntegerVector vector, int scalar)

{

if (vector == null)

throw new ArgumentNullException("Вектор не может быть null");

var result = new IntegerVector(vector.\_lowerBound, vector.\_lowerBound + vector.\_data.Length - 1);

for (int i = 0; i < vector.\_data.Length; i++)

{

result.\_data[i] = vector.\_data[i] \* scalar;

}

return result;

}

// Деление вектора на скаляр

public static IntegerVector operator /(IntegerVector vector, int scalar)

{

if (vector == null)

throw new ArgumentNullException("Вектор не может быть null");

if (scalar == 0)

throw new DivideByZeroException("Деление на ноль невозможно");

var result = new IntegerVector(vector.\_lowerBound, vector.\_lowerBound + vector.\_data.Length - 1);

for (int i = 0; i < vector.\_data.Length; i++)

{

result.\_data[i] = vector.\_data[i] / scalar;

}

return result;

}

// Вывод элемента по индексу

public void PrintElement(int index)

{

try

{

Console.WriteLine($"Элемент с индексом {index}: {this[index]}");

}

catch (IndexOutOfRangeException ex)

{

Console.WriteLine(ex.Message);

}

}

// Вывод всего массива

public void PrintAll()

{

Console.WriteLine("Содержимое вектора:");

for (int i = \_lowerBound; i < \_lowerBound + \_data.Length; i++)

{

Console.WriteLine($"[{i}] = {this[i]}");

}

}

}

class Program

{

static void Main()

{

Console.WriteLine("Демонстрация работы класса IntegerVector\n");

try

{

// 1. Создание векторов с разными границами

Console.WriteLine("1. Создаем векторы:");

var vector1 = new IntegerVector(1, 5);

var vector2 = new IntegerVector(1, 5);

var vector3 = new IntegerVector(10, 12);

// 2. Заполнение векторов значениями

for (int i = 1; i <= 5; i++)

{

vector1[i] = i \* 2;

vector2[i] = i \* 3;

}

for (int i = 10; i <= 12; i++)

{

vector3[i] = i;

}

// 3. Вывод векторов

Console.WriteLine("\nВектор 1:");

vector1.PrintAll();

Console.WriteLine("\nВектор 2:");

vector2.PrintAll();

Console.WriteLine("\nВектор 3:");

vector3.PrintAll();

// 4. Поэлементные операции

Console.WriteLine("\n4. Сложение векторов 1 и 2:");

var sum = vector1 + vector2;

sum.PrintAll();

Console.WriteLine("\nВычитание векторов 1 и 2:");

var diff = vector1 - vector2;

diff.PrintAll();

// 5. Операции со скалярами

Console.WriteLine("\n5. Умножение вектора 1 на 3:");

var mult = vector1 \* 3;

mult.PrintAll();

Console.WriteLine("\nДеление вектора 2 на 2:");

var div = vector2 / 2;

div.PrintAll();

// 6. Проверка границ

Console.WriteLine("\n6. Попытка обращения к несуществующему индексу:");

vector1.PrintElement(0);

vector1.PrintElement(6);

// 7. Попытка операций с разными границами

Console.WriteLine("\n7. Попытка сложения векторов с разными границами:");

try

{

var invalid = vector1 + vector3;

}

catch (ArgumentException ex)

{

Console.WriteLine(ex.Message);

}

}

catch (Exception ex)

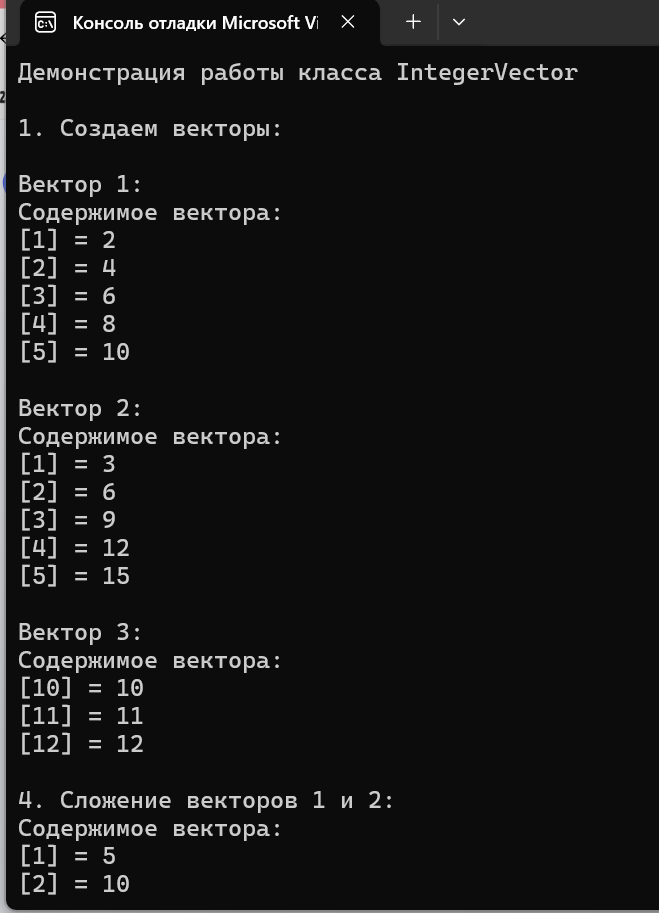
{

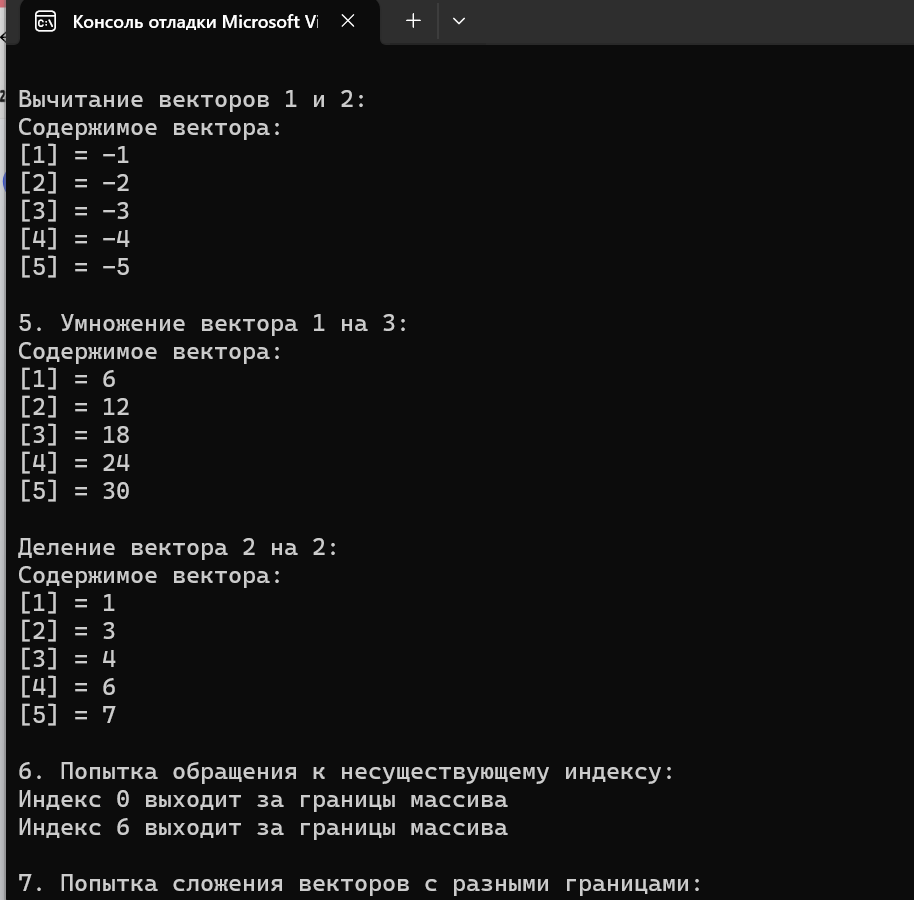
Console.WriteLine($"Неожиданная ошибка: {ex.Message}");

}

}

}





Лабораторная работа 9. Наследование

Вариант 1

Создать класс Point (точка). Нa еro основе создать классы ColoredPoint и Line (линия). На основе класса Line создать классы ColoredLine и PolyLine (многоугольник). В классах описать следующие элементы:

* конструкторы с параметрами и конструкторы по умолчанию;
* свойства для установки и получения значений всех координат, а также для изменения цвета и получения текущего цвета;
* для линий - методы изменения угла поворота линий относительно первой точки;
* для многоугольника - метод масштабирования

листинг кода:

using System;

// Базовый класс Point

public class Point

{

public double X { get; set; }

public double Y { get; set; }

// Конструктор по умолчанию

public Point() : this(0, 0) { }

// Конструктор с параметрами

public Point(double x, double y)

{

X = x;

Y = y;

}

// Метод для вывода информации о точке

public virtual void Print()

{

Console.WriteLine($"Точка: ({X}, {Y})");

}

}

// Класс ColoredPoint - наследник Point

public class ColoredPoint : Point

{

public ConsoleColor Color { get; set; }

// Конструктор по умолчанию

public ColoredPoint() : this(0, 0, ConsoleColor.White) { }

// Конструктор с параметрами

public ColoredPoint(double x, double y, ConsoleColor color) : base(x, y)

{

Color = color;

}

// Переопределенный метод вывода

public override void Print()

{

Console.ForegroundColor = Color;

Console.Write($"Цветная точка: ({X}, {Y}) ");

Console.ResetColor();

Console.WriteLine($"(Цвет: {Color})");

}

}

// Класс Line

public class Line

{

public Point Start { get; set; }

public Point End { get; set; }

// Конструктор по умолчанию

public Line() : this(new Point(), new Point()) { }

// Конструктор с параметрами

public Line(Point start, Point end)

{

Start = start ?? throw new ArgumentNullException(nameof(start));

End = end ?? throw new ArgumentNullException(nameof(end));

}

// Длина линии

public double Length => Math.Sqrt(Math.Pow(End.X - Start.X, 2) + Math.Pow(End.Y - Start.Y, 2));

// Метод поворота линии относительно начальной точки

public void Rotate(double angleDegrees)

{

double angleRadians = angleDegrees \* Math.PI / 180;

double cosTheta = Math.Cos(angleRadians);

double sinTheta = Math.Sin(angleRadians);

// Перенос конечной точки относительно начала координат

double translatedX = End.X - Start.X;

double translatedY = End.Y - Start.Y;

// Поворот точки

double rotatedX = translatedX \* cosTheta - translatedY \* sinTheta;

double rotatedY = translatedX \* sinTheta + translatedY \* cosTheta;

// Возвращение точки на место

End.X = Start.X + rotatedX;

End.Y = Start.Y + rotatedY;

}

// Метод для вывода информации о линии

public virtual void Print()

{

Console.WriteLine($"Линия от ({Start.X}, {Start.Y}) до ({End.X}, {End.Y})");

Console.WriteLine($"Длина: {Length:F2}");

}

}

// Класс ColoredLine - наследник Line

public class ColoredLine : Line

{

public ConsoleColor Color { get; set; }

// Конструктор по умолчанию

public ColoredLine() : this(new Point(), new Point(), ConsoleColor.White) { }

// Конструктор с параметрами

public ColoredLine(Point start, Point end, ConsoleColor color) : base(start, end)

{

Color = color;

}

// Переопределенный метод вывода

public override void Print()

{

Console.ForegroundColor = Color;

Console.Write($"Цветная линия от ({Start.X}, {Start.Y}) до ({End.X}, {End.Y}) ");

Console.ResetColor();

Console.WriteLine($"(Цвет: {Color})");

Console.WriteLine($"Длина: {Length:F2}");

}

}

// Класс PolyLine

public class PolyLine : Line

{

public Point[] AdditionalPoints { get; set; }

// Конструктор по умолчанию

public PolyLine() : this(new Point(), new Point(), Array.Empty<Point>()) { }

// Конструктор с параметрами

public PolyLine(Point start, Point end, Point[] additionalPoints) : base(start, end)

{

AdditionalPoints = additionalPoints ?? throw new ArgumentNullException(nameof(additionalPoints));

}

// Метод масштабирования многоугольника

public void Scale(double factor)

{

if (factor <= 0)

throw new ArgumentException("Масштабный коэффициент должен быть положительным");

// Находим центр масс

double centerX = (Start.X + End.X) / 2;

double centerY = (Start.Y + End.Y) / 2;

foreach (var point in AdditionalPoints)

{

centerX += point.X;

centerY += point.Y;

}

centerX /= (AdditionalPoints.Length + 2);

centerY /= (AdditionalPoints.Length + 2);

// Масштабируем все точки относительно центра

ScalePoint(Start, centerX, centerY, factor);

ScalePoint(End, centerX, centerY, factor);

foreach (var point in AdditionalPoints)

{

ScalePoint(point, centerX, centerY, factor);

}

}

private void ScalePoint(Point point, double centerX, double centerY, double factor)

{

point.X = centerX + (point.X - centerX) \* factor;

point.Y = centerY + (point.Y - centerY) \* factor;

}

// Переопределенный метод вывода

public override void Print()

{

Console.WriteLine("Многоугольник с точками:");

Console.WriteLine($"1. ({Start.X}, {Start.Y})");

Console.WriteLine($"2. ({End.X}, {End.Y})");

for (int i = 0; i < AdditionalPoints.Length; i++)

{

Console.WriteLine($"{i + 3}. ({AdditionalPoints[i].X}, {AdditionalPoints[i].Y})");

}

}

}

class Program

{

static void Main()

{

Console.WriteLine("Демонстрация иерархии классов геометрических фигур\n");

try

{

// 1. Создание и демонстрация точек

Console.WriteLine("1. Точки:");

var point1 = new Point(3, 4);

point1.Print();

var coloredPoint = new ColoredPoint(5, 7, ConsoleColor.Red);

coloredPoint.Print();

// 2. Создание и демонстрация линий

Console.WriteLine("\n2. Линии:");

var line1 = new Line(new Point(1, 1), new Point(4, 5));

line1.Print();

Console.WriteLine("\nПоворот линии на 45 градусов:");

line1.Rotate(45);

line1.Print();

var coloredLine = new ColoredLine(new Point(0, 0), new Point(3, 3), ConsoleColor.Blue);

coloredLine.Print();

// 3. Создание и демонстрация многоугольника

Console.WriteLine("\n3. Многоугольник:");

var polyLine = new PolyLine(

new Point(1, 1),

new Point(4, 1),

new Point[] { new Point(4, 4), new Point(1, 4) });

polyLine.Print();

Console.WriteLine("\nМасштабирование многоугольника в 2 раза:");

polyLine.Scale(2);

polyLine.Print();

}

catch (Exception ex)

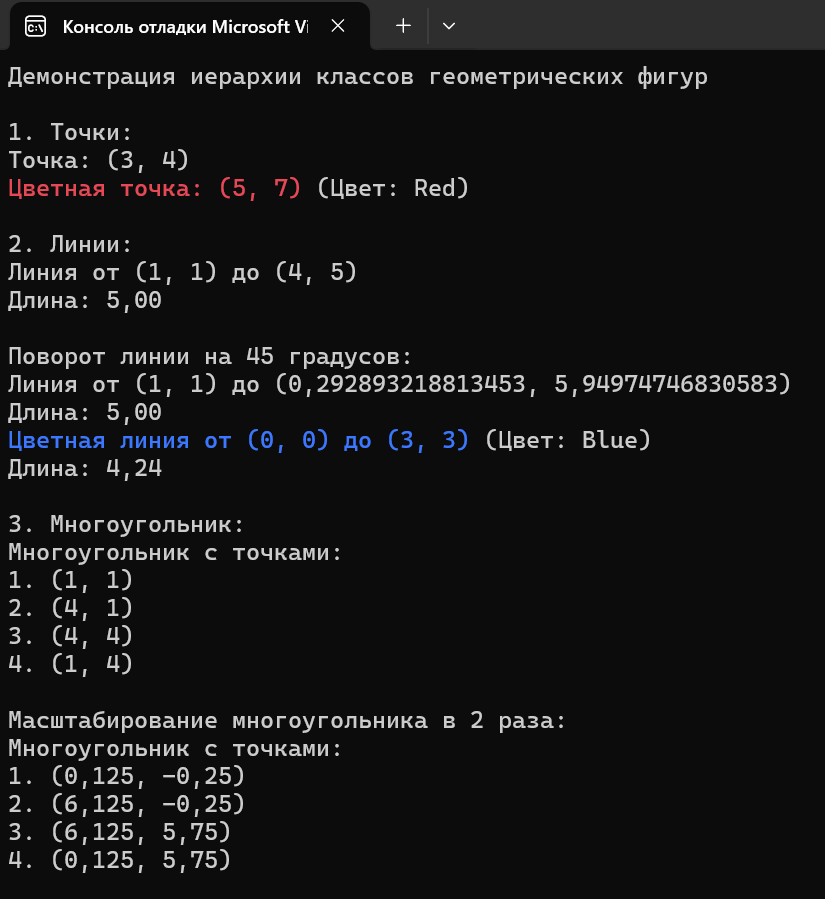
{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

}



Лабораторная работа 10. Структуры

Вариант 1

Описать структуру с именем STUDENT, содержащую следующие поля:

* фамилия и инициалы;
* номер групны;
* усневаемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

* ввод с клавиатуры дапных в массив, состоящий из десяти структур типа STUDENT (записи должны быть упорядочены по возрастанию номера групны);
* вывод на экран фамилий и номеров групи для всех студентов, включенных в массив, если средний балл студента больше 4,0 (если таких студентов нет, вывести соответствующее сообщение)

листинг кода:

using System;

using System.Linq;

// Определение структуры STUDENT

struct STUDENT

{

public string LastNameInitials; // Фамилия и инициалы

public int GroupNumber; // Номер группы

public int[] AcademicPerformance; // Успеваемость (массив из 5 оценок)

// Метод для вычисления среднего балла

public double GetAverageScore()

{

return AcademicPerformance.Average();

}

// Метод для вывода информации о студенте

public void PrintInfo()

{

Console.WriteLine($"Студент: {LastNameInitials}, Группа: {GroupNumber}, Средний балл: {GetAverageScore():F2}");

}

}

class Program

{

static void Main()

{

const int studentCount = 10; // Количество студентов

STUDENT[] students = new STUDENT[studentCount];

Console.WriteLine($"Введите данные для {studentCount} студентов:");

// Ввод данных о студентах

for (int i = 0; i < studentCount; i++)

{

Console.WriteLine($"\nСтудент #{i + 1}:");

Console.Write("Фамилия и инициалы: ");

students[i].LastNameInitials = Console.ReadLine();

Console.Write("Номер группы: ");

while (!int.TryParse(Console.ReadLine(), out students[i].GroupNumber) || students[i].GroupNumber <= 0)

{

Console.Write("Некорректный ввод. Введите положительное число для номера группы: ");

}

students[i].AcademicPerformance = new int[5];

Console.WriteLine("Введите 5 оценок (через Enter):");

for (int j = 0; j < 5; j++)

{

while (!int.TryParse(Console.ReadLine(), out students[i].AcademicPerformance[j])

|| students[i].AcademicPerformance[j] < 1

|| students[i].AcademicPerformance[j] > 5)

{

Console.Write("Некорректная оценка. Введите число от 1 до 5: ");

}

}

}

// Сортировка студентов по номеру группы (по возрастанию)

students = students.OrderBy(s => s.GroupNumber).ToArray();

Console.WriteLine("\nОтсортированный список студентов по номеру группы:");

foreach (var student in students)

{

student.PrintInfo();

}

// Поиск студентов со средним баллом > 4.0

var topStudents = students.Where(s => s.GetAverageScore() > 4.0).ToArray();

Console.WriteLine("\nСтуденты со средним баллом больше 4.0:");

if (topStudents.Length == 0)

{

Console.WriteLine("Таких студентов нет.");

}

else

{

foreach (var student in topStudents)

{

Console.WriteLine($"Фамилия: {student.LastNameInitials}, Группа: {student.GroupNumber}, Средний балл: {student.GetAverageScore():F2}");

}

}

}

}

