Министерство образования Республики Беларусь

Оршанский колледж ВГУ имени П.М. Машерова

Отчет

По модулю №3

«Задания по делегатам»

по учебной практике по программированию

Выполнил учащийся Войтеховская К.В.

группа 3ПОИС23 29.09.2025 г.

Проверил Алейников М.А.

30.09.2025 г.

Орша, 2025

**Ход работы.**

**Выполнение практических заданий.**

**Выполнение задания 1.** Создайте базовый класс "Фигура" с методом для вычисления площади. Затем создайте производные классы для разных геометрических фигур (круг, прямоугольник, треугольник) и используйте делегат для динамического вызова метода вычисления площади.

Листинг 1. Код программы

using System;

// Делегат для вычисления площади

public delegate double CalculateAreaDelegate();

// Базовый класс Фигура

abstract class Shape

{

public abstract double CalculateArea();

}

// Класс Круг

class Circle : Shape

{

public double Radius { get; set; }

public Circle(double radius)

{

Radius = radius;

}

public override double CalculateArea()

{

return Math.PI \* Radius \* Radius;

}

}

// Класс Прямоугольник

class Rectangle : Shape

{

public double Width { get; set; }

public double Height { get; set; }

public Rectangle(double width, double height)

{

Width = width;

Height = height;

}

public override double CalculateArea()

{

return Width \* Height;

}

}

// Класс Треугольник

class Triangle : Shape

{

public double Base { get; set; }

public double Height { get; set; }

public Triangle(double triangleBase, double height)

{

Base = triangleBase;

Height = height;

}

public override double CalculateArea()

{

return 0.5 \* Base \* Height;

}

}

class Program

{

static void Main()

{

// Создание фигур

Circle circle = new Circle(5);

Rectangle rectangle = new Rectangle(4, 6);

Triangle triangle = new Triangle(3, 4);

// Создание делегатов для каждой фигуры

CalculateAreaDelegate circleArea = circle.CalculateArea;

CalculateAreaDelegate rectangleArea = rectangle.CalculateArea;

CalculateAreaDelegate triangleArea = triangle.CalculateArea;

// Массив делегатов

CalculateAreaDelegate[] areaDelegates = { circleArea, rectangleArea, triangleArea };

Shape[] shapes = { circle, rectangle, triangle };

string[] shapeNames = { "Круг", "Прямоугольник", "Треугольник" };

Console.WriteLine("Вычисление площадей фигур с использованием делегатов:");

// Вызов делегатов

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

{

double area = areaDelegates[i]();

Console.WriteLine($"{shapeNames[i]}: Площадь = {area:F2}");

}

}

}

На рисунке 1 представлен результат выполнения программы.

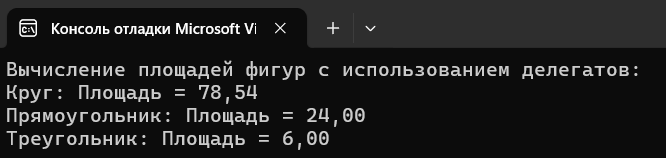


Рисунок 1 – Вывод параметров фигур

**Выполнение задания 2.** Реализуйте систему событий для мобильного приложения. Создайте класс "Уведомление" с событиями для отправки уведомлений (сообщения, звонки, электронные письма). Зарегистрируйте обработчики событий для разных типов уведомлений.

Листинг 2. Код программы с событиями для мобильного приложения

using System;

// Класс для аргументов события уведомления

public class NotificationEventArgs : EventArgs

{

public string Message { get; }

public string Sender { get; }

public DateTime Timestamp { get; }

public NotificationEventArgs(string message, string sender)

{

Message = message;

Sender = sender;

Timestamp = DateTime.Now;

}

}

// Класс Уведомление

class NotificationManager

{

// События для разных типов уведомлений

public event EventHandler<NotificationEventArgs> MessageReceived;

public event EventHandler<NotificationEventArgs> CallReceived;

public event EventHandler<NotificationEventArgs> EmailReceived;

// Методы для генерации событий

public void SendMessage(string message, string sender)

{

Console.WriteLine($"Отправка сообщения от {sender}: {message}");

MessageReceived?.Invoke(this, new NotificationEventArgs(message, sender));

}

public void MakeCall(string message, string caller)

{

Console.WriteLine($"Входящий звонок от {caller}: {message}");

CallReceived?.Invoke(this, new NotificationEventArgs(message, caller));

}

public void SendEmail(string message, string sender)

{

Console.WriteLine($"Отправка email от {sender}: {message}");

EmailReceived?.Invoke(this, new NotificationEventArgs(message, sender));

}

}

// Класс-обработчик уведомлений

class NotificationHandler

{

private string handlerName;

public NotificationHandler(string name)

{

handlerName = name;

}

// Обработчики событий

public void OnMessageReceived(object sender, NotificationEventArgs e)

{

Console.WriteLine($"[{handlerName}] СООБЩЕНИЕ получено в {e.Timestamp:HH:mm:ss}");

Console.WriteLine($" От: {e.Sender}");

Console.WriteLine($" Текст: {e.Message}");

Console.WriteLine(" Действие: Показать всплывающее уведомление\n");

}

public void OnCallReceived(object sender, NotificationEventArgs e)

{

Console.WriteLine($"[{handlerName}] ЗВОНОК получен в {e.Timestamp:HH:mm:ss}");

Console.WriteLine($" От: {e.Sender}");

Console.WriteLine($" Тема: {e.Message}");

Console.WriteLine(" Действие: Воспроизвести рингтон\n");

}

public void OnEmailReceived(object sender, NotificationEventArgs e)

{

Console.WriteLine($"[{handlerName}] EMAIL получен в {e.Timestamp:HH:mm:ss}");

Console.WriteLine($" От: {e.Sender}");

Console.WriteLine($" Тема: {e.Message}");

Console.WriteLine(" Действие: Добавить в папку 'Входящие'\n");

}

}

class Program

{

static void Main()

{

// Создание менеджера уведомлений

NotificationManager notificationManager = new NotificationManager();

// Создание обработчиков

NotificationHandler mainHandler = new NotificationHandler("Основной обработчик");

NotificationHandler backupHandler = new NotificationHandler("Резервный обработчик");

// Подписка на события

notificationManager.MessageReceived += mainHandler.OnMessageReceived;

notificationManager.CallReceived += mainHandler.OnCallReceived;

notificationManager.EmailReceived += mainHandler.OnEmailReceived;

// Дополнительная подписка для демонстрации

notificationManager.MessageReceived += backupHandler.OnMessageReceived;

Console.WriteLine("Система уведомлений мобильного приложения:");

// Генерация уведомлений

notificationManager.SendMessage("Привет! Как дела?", "Анна");

notificationManager.MakeCall("Обсудить проект", "Иван Петров");

notificationManager.SendEmail("Отчет за месяц", "бухгалтерия@company.com");

notificationManager.SendMessage("Напоминание: встреча в 15:00", "Календарь");

// Отписка от события

notificationManager.MessageReceived -= backupHandler.OnMessageReceived;

Console.WriteLine("После отписки резервного обработчика:");

notificationManager.SendMessage("Это сообщение увидят только основные обработчики", "Система");

}

}

На рисунке 2 представлен результат выполнения программы.

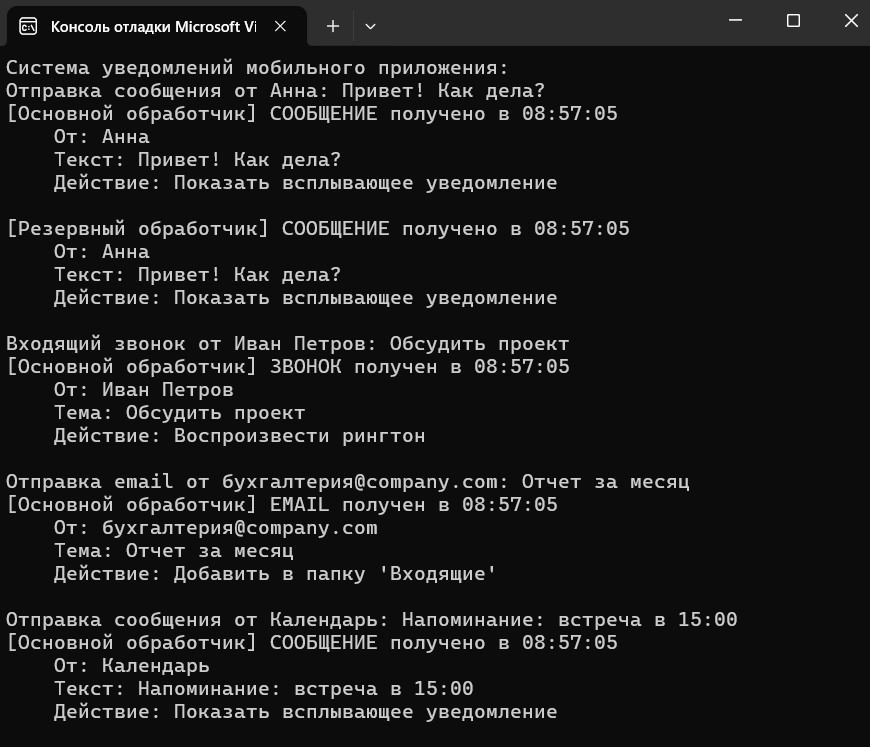


Рисунок 2 – Вывод событий

**Выполнение задания 3.** Создайте приложение для управления задачами с использованием делегатов. Пользователь должен иметь возможность добавлять задачи и выбирать делегата для выполнения каждой задачи (например, отправка уведомления или запись в журнал).

Листинг 3. Код программы

using System;

using System.Collections.Generic;

// Делегат для выполнения задачи

public delegate void TaskDelegate(string taskDescription);

// Класс Задача

class Task

{

public string Description { get; set; }

public TaskDelegate TaskHandler { get; set; }

public DateTime CreatedAt { get; set; }

public Task(string description, TaskDelegate handler)

{

Description = description;

TaskHandler = handler;

CreatedAt = DateTime.Now;

}

public void Execute()

{

TaskHandler?.Invoke(Description);

}

}

// Класс Менеджер задач

class TaskManager

{

private List<Task> tasks = new List<Task>();

// Методы-обработчики задач

public static void SendNotification(string taskDescription)

{

Console.WriteLine($"[УВЕДОМЛЕНИЕ] Отправка уведомления для задачи: {taskDescription}");

Console.WriteLine(" Статус: Уведомление отправлено успешно\n");

}

public static void LogToJournal(string taskDescription)

{

string logEntry = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} - {taskDescription}";

Console.WriteLine($"[ЖУРНАЛ] Запись в журнал: {taskDescription}");

Console.WriteLine($" Запись: {logEntry}\n");

}

public static void SendEmail(string taskDescription)

{

Console.WriteLine($"[EMAIL] Отправка email для задачи: {taskDescription}");

Console.WriteLine(" Статус: Email отправлен на почту\n");

}

public static void BackupData(string taskDescription)

{

Console.WriteLine($"[БЭКАП] Резервное копирование для задачи: {taskDescription}");

Console.WriteLine(" Статус: Данные сохранены в backup\n");

}

// Добавление задачи

public void AddTask(string description, TaskDelegate handler)

{

Task task = new Task(description, handler);

tasks.Add(task);

Console.WriteLine($"Задача добавлена: {description}");

}

// Выполнение всех задач

public void ExecuteAllTasks()

{

Console.WriteLine("\nВыполнение всех задач:");

foreach (var task in tasks)

{

Console.WriteLine($"Задача: {task.Description}");

task.Execute();

}

tasks.Clear();

Console.WriteLine("Все задачи выполнены и очищены из списка.");

}

// Выполнение задач по фильтру

public void ExecuteTasksByFilter(Predicate<Task> filter)

{

Console.WriteLine("\nВыполнение отфильтрованных задач:");

var filteredTasks = tasks.FindAll(filter);

foreach (var task in filteredTasks)

{

task.Execute();

}

}

}

class Program

{

static void Main()

{

TaskManager taskManager = new TaskManager();

Console.WriteLine("Система управления задачами с делегатами:");

// Создание делегатов

TaskDelegate notifyDelegate = TaskManager.SendNotification;

TaskDelegate logDelegate = TaskManager.LogToJournal;

TaskDelegate emailDelegate = TaskManager.SendEmail;

TaskDelegate backupDelegate = TaskManager.BackupData;

// Комбинированный делегат

TaskDelegate multiTaskDelegate = TaskManager.SendNotification;

multiTaskDelegate += TaskManager.LogToJournal;

multiTaskDelegate += TaskManager.SendEmail;

// Добавление задач

taskManager.AddTask("Проверить систему безопасности", notifyDelegate);

taskManager.AddTask("Обновить базу данных", logDelegate);

taskManager.AddTask("Отправить отчет руководству", emailDelegate);

taskManager.AddTask("Создать резервную копию", backupDelegate);

taskManager.AddTask("Комплексная задача: аудит системы", multiTaskDelegate);

// Выполнение всех задач

taskManager.ExecuteAllTasks();

// Добавление новых задач для демонстрации фильтрации

Console.WriteLine("\nДобавление новых задач для демонстрации фильтрации:");

taskManager.AddTask("Срочное уведомление", notifyDelegate);

taskManager.AddTask("Логирование ошибок", logDelegate);

taskManager.AddTask("Рассылка новостей", emailDelegate);

taskManager.AddTask("Резервное копирование логов", backupDelegate);

// Фильтрация задач по ключевым словам

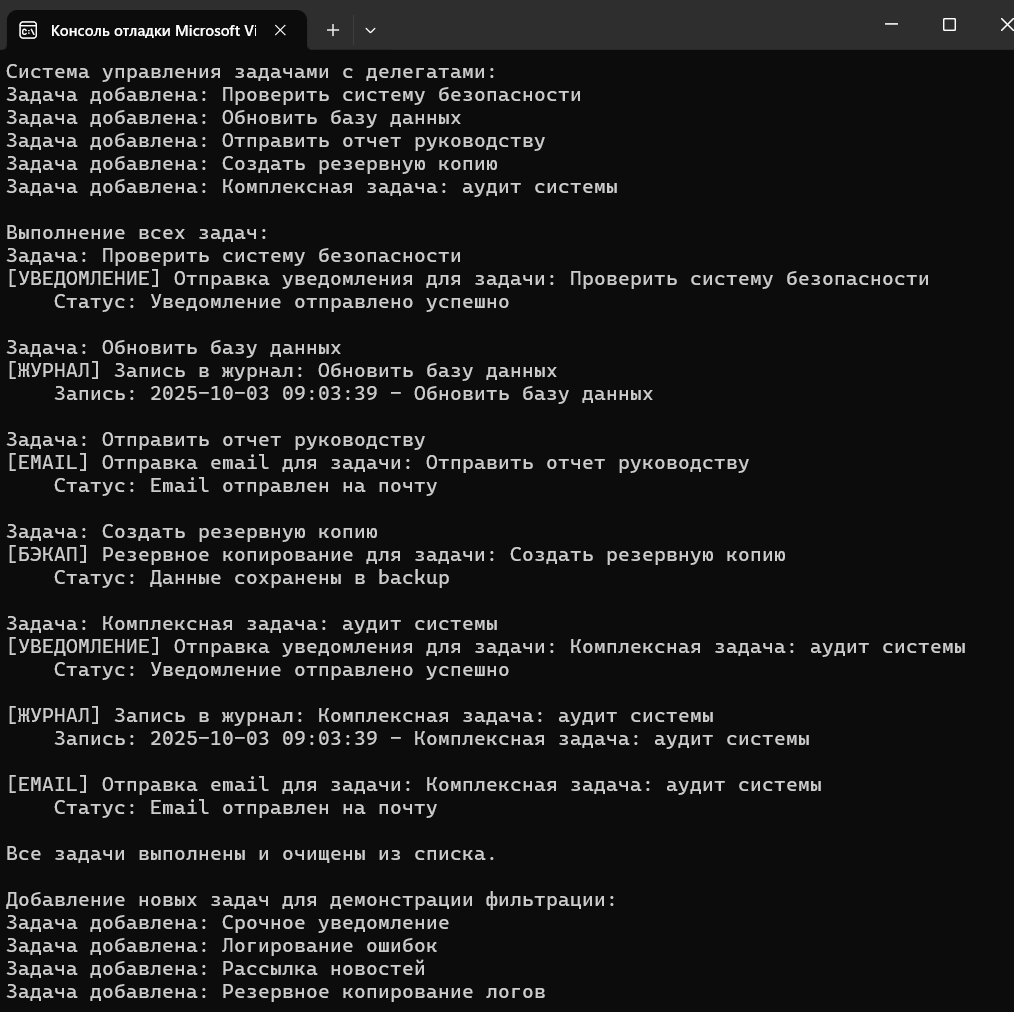
taskManager.ExecuteTasksByFilter(task => task.Description.ToLower().Contains("уведомление"));

taskManager.ExecuteTasksByFilter(task => task.Description.ToLower().Contains("резерв"));

}

}

На рисунке 3 представлен результат выполнения программы.



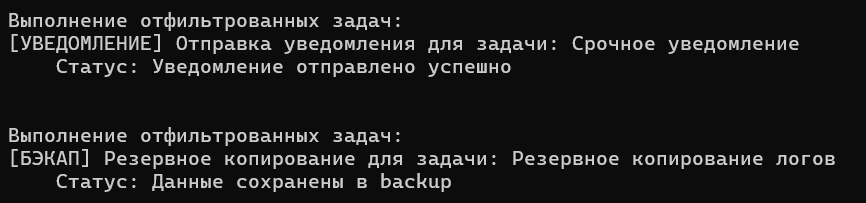


Рисунок 3 – Приложение для управления делегатами

**Выполнение задания 4.** Разработайте систему фильтрации данных с использованием делегатов. Пользователь должен иметь возможность выбрать фильтр для списка данных (например, фильтр по дате или по ключевым словам).

Листинг 4. Код программы

using System;

using System.Collections.Generic;

using System.Linq;

// Делегат для фильтрации

public delegate bool FilterDelegate<T>(T item);

// Класс для работы с данными

class DataManager<T>

{

private List<T> data;

public DataManager(List<T> initialData)

{

data = initialData;

}

// Метод фильтрации с использованием делегата

public List<T> FilterData(FilterDelegate<T> filter)

{

return data.Where(item => filter(item)).ToList();

}

// Вывод данных

public void DisplayData(string title, List<T> dataToDisplay = null)

{

var displayData = dataToDisplay ?? data;

Console.WriteLine(title);

foreach (var item in displayData)

{

Console.WriteLine(item);

}

Console.WriteLine();

}

}

// Класс для представления записи

class DataRecord

{

public string Title { get; set; }

public DateTime Date { get; set; }

public string[] Keywords { get; set; }

public int Priority { get; set; }

public DataRecord(string title, DateTime date, string[] keywords, int priority)

{

Title = title;

Date = date;

Keywords = keywords;

Priority = priority;

}

public override string ToString()

{

return $"{Date:yyyy-MM-dd} | Приоритет: {Priority} | {Title} [Ключевые слова: {string.Join(", ", Keywords)}]";

}

}

class Program

{

static void Main()

{

// Создание тестовых данных

var records = new List<DataRecord>

{

new DataRecord("Отчет за январь", new DateTime(2024, 1, 15), new[] { "отчет", "финансы" }, 2),

new DataRecord("Важная встреча", new DateTime(2024, 2, 1), new[] { "встреча", "срочно" }, 1),

new DataRecord("Презентация проекта", new DateTime(2024, 1, 20), new[] { "презентация", "проект" }, 3),

new DataRecord("Бюджет на квартал", new DateTime(2024, 2, 10), new[] { "бюджет", "финансы" }, 1),

new DataRecord("Обучение сотрудников", new DateTime(2024, 1, 25), new[] { "обучение", "кадры" }, 2),

new DataRecord("Срочный ремонт", new DateTime(2024, 2, 5), new[] { "ремонт", "срочно" }, 1)

};

DataManager<DataRecord> dataManager = new DataManager<DataRecord>(records);

Console.WriteLine("Система фильтрации данных с делегатами:");

// Отображение всех данных

dataManager.DisplayData("ВСЕ ДАННЫХ:");

// Различные фильтры с использованием делегатов

// Фильтр по дате (за последний месяц)

FilterDelegate<DataRecord> dateFilter = record => record.Date >= new DateTime(2024, 2, 1);

var filteredByDate = dataManager.FilterData(dateFilter);

dataManager.DisplayData("ДАННЫЕ ЗА ФЕВРАЛЬ 2024:", filteredByDate);

// Фильтр по ключевым словам

FilterDelegate<DataRecord> keywordFilter = record => record.Keywords.Any(kw => kw == "срочно");

var filteredByKeywords = dataManager.FilterData(keywordFilter);

dataManager.DisplayData("СРОЧНЫЕ ЗАДАЧИ:", filteredByKeywords);

// Фильтр по приоритету

FilterDelegate<DataRecord> priorityFilter = record => record.Priority == 1;

var highPriority = dataManager.FilterData(priorityFilter);

dataManager.DisplayData("ВЫСОКИЙ ПРИОРИТЕТ (1):", highPriority);

// Комбинированный фильтр с использованием лямбда-выражения

var complexFilter = dataManager.FilterData(record =>

record.Date.Month == 1 && record.Keywords.Any(kw => kw == "финансы"));

dataManager.DisplayData("ФИНАНСОВЫЕ ОТЧЕТЫ ЗА ЯНВАРЬ:", complexFilter);

// Динамическое создание фильтров

Console.WriteLine("ДИНАМИЧЕСКАЯ ФИЛЬТРАЦИЯ:");

// Пользовательский ввод фильтра (имитация)

string searchKeyword = "проект";

FilterDelegate<DataRecord> dynamicFilter = record =>

record.Title.ToLower().Contains(searchKeyword.ToLower()) ||

record.Keywords.Any(kw => kw.ToLower().Contains(searchKeyword.ToLower()));

var dynamicResults = dataManager.FilterData(dynamicFilter);

dataManager.DisplayData($"РЕЗУЛЬТАТЫ ПОИСКА ПО '{searchKeyword}':", dynamicResults);

// Многоуровневая фильтрация

Console.WriteLine("МНОГОУРОВНЕВАЯ ФИЛЬТРАЦИЯ:");

// Цепочка фильтров

var multiFiltered = records

.Where(record => record.Priority <= 2)

.Where(record => record.Date >= new DateTime(2024, 1, 20))

.ToList();

dataManager.DisplayData("ЗАДАЧИ ПРИОРИТЕТА 1-2 ПОСЛЕ 20 ЯНВАРЯ:", multiFiltered);

}

}

На рисунке 4 представлен результат выполнения программы.

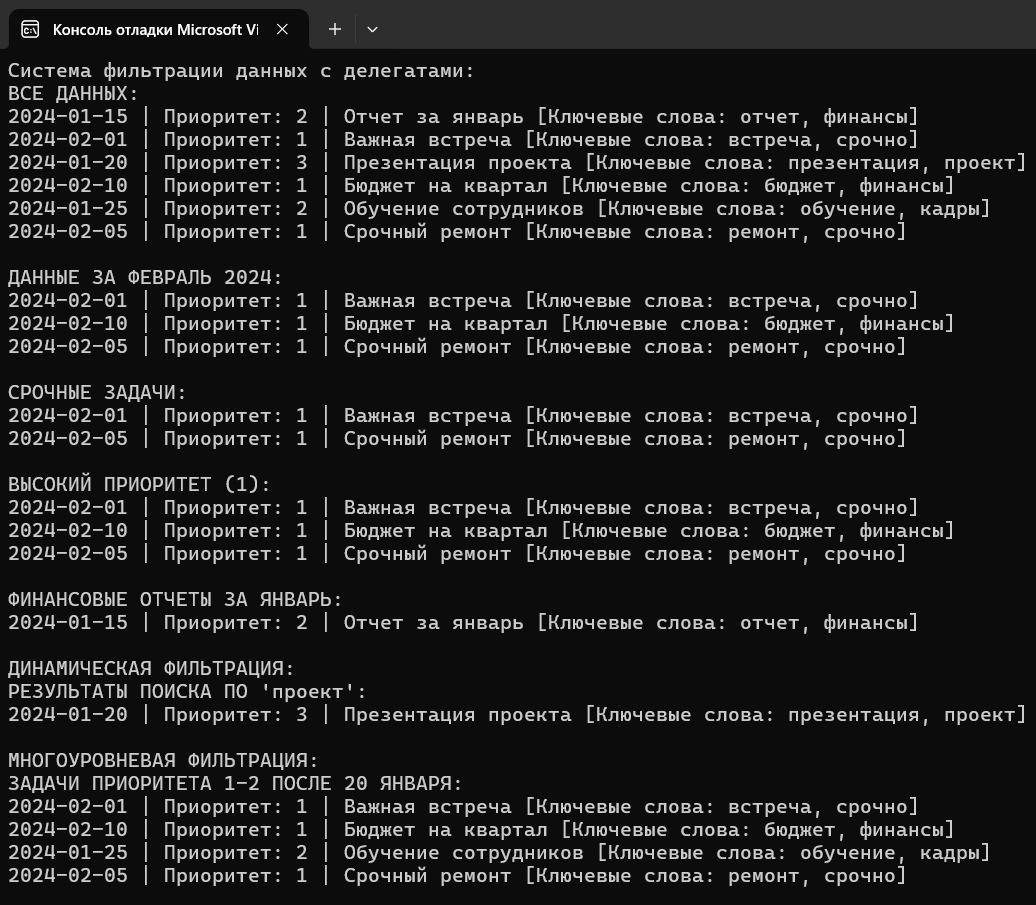


Рисунок 4 – Система фильтрации данных

**Выполнение задания 5.** 5.Создайте приложение для сортировки числовых данных. Пользователь должен иметь возможность выбрать метод сортировки (например, сортировка пузырьком или быстрая сортировка) с помощью делегатов.

Листинг 5. Код программы

using System;

using System.Collections.Generic;

// Делегат для сортировки

public delegate void SortDelegate(List<int> data);

// Класс для управления сортировкой

class SortManager

{

// Метод пузырьковой сортировки

public static void BubbleSort(List<int> data)

{

int n = data.Count;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (data[j] > data[j + 1])

{

// Обмен элементов

int temp = data[j];

data[j] = data[j + 1];

data[j + 1] = temp;

}

}

}

}

// Метод быстрой сортировки

public static void QuickSort(List<int> data)

{

if (data.Count <= 1) return;

QuickSortRecursive(data, 0, data.Count - 1);

}

private static void QuickSortRecursive(List<int> data, int left, int right)

{

if (left < right)

{

int pivotIndex = Partition(data, left, right);

QuickSortRecursive(data, left, pivotIndex - 1);

QuickSortRecursive(data, pivotIndex + 1, right);

}

}

private static int Partition(List<int> data, int left, int right)

{

int pivot = data[right];

int i = left - 1;

for (int j = left; j < right; j++)

{

if (data[j] <= pivot)

{

i++;

Swap(data, i, j);

}

}

Swap(data, i + 1, right);

return i + 1;

}

private static void Swap(List<int> data, int i, int j)

{

int temp = data[i];

data[i] = data[j];

data[j] = temp;

}

// Метод сортировки выбором

public static void SelectionSort(List<int> data)

{

int n = data.Count;

for (int i = 0; i < n - 1; i++)

{

int minIndex = i;

for (int j = i + 1; j < n; j++)

{

if (data[j] < data[minIndex])

{

minIndex = j;

}

}

if (minIndex != i)

{

Swap(data, i, minIndex);

}

}

}

// Метод сортировки вставками

public static void InsertionSort(List<int> data)

{

for (int i = 1; i < data.Count; i++)

{

int key = data[i];

int j = i - 1;

while (j >= 0 && data[j] > key)

{

data[j + 1] = data[j];

j--;

}

data[j + 1] = key;

}

}

// Сортировка по убыванию

public static void SortDescending(List<int> data)

{

data.Sort((a, b) => b.CompareTo(a));

}

// Генерация случайных данных

public static List<int> GenerateRandomData(int count, int min = 1, int max = 1000)

{

Random random = new Random();

List<int> data = new List<int>();

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

{

data.Add(random.Next(min, max));

}

return data;

}

// Вывод данных

public static void DisplayData(List<int> data, string title, int maxDisplay = 20)

{

Console.WriteLine(title);

Console.Write("[");

int displayCount = Math.Min(data.Count, maxDisplay);

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

{

Console.Write(data[i]);

if (i < displayCount - 1) Console.Write(", ");

}

if (data.Count > maxDisplay)

{

Console.Write(", ...");

}

Console.WriteLine($"] (всего элементов: {data.Count})");

}

}

class Program

{

static void Main()

{

Console.WriteLine("Приложение для сортировки числовых данных");

// Создание делегатов для разных методов сортировки

var sortMethods = new Dictionary<string, SortDelegate>

{

{ "1", SortManager.BubbleSort },

{ "2", SortManager.QuickSort },

{ "3", SortManager.SelectionSort },

{ "4", SortManager.InsertionSort },

{ "5", SortManager.SortDescending }

};

bool continueRunning = true;

while (continueRunning)

{

Console.Clear();

DisplayMenu();

string choice = Console.ReadLine();

if (choice == "6")

{

continueRunning = false;

Console.WriteLine("Выход из программы...");

continue;

}

if (sortMethods.ContainsKey(choice))

{

ProcessUserChoice(choice, sortMethods);

}

else

{

Console.WriteLine("Неверный выбор! Пожалуйста, выберите от 1 до 6.");

WaitForUser();

}

}

}

static void DisplayMenu()

{

Console.WriteLine("ВЫБЕРИТЕ МЕТОД СОРТИРОВКИ:");

Console.WriteLine("1. Пузырьковая сортировка");

Console.WriteLine("2. Быстрая сортировка");

Console.WriteLine("3. Сортировка выбором");

Console.WriteLine("4. Сортировка вставками");

Console.WriteLine("5. Сортировка по убыванию");

Console.WriteLine("6. Выход");

Console.Write("\nВаш выбор: ");

}

static void ProcessUserChoice(string choice, Dictionary<string, SortDelegate> sortMethods)

{

// Выбор размера данных

Console.Write("\nВведите количество элементов для сортировки (по умолчанию 100): ");

string sizeInput = Console.ReadLine();

int size = string.IsNullOrEmpty(sizeInput) ? 100 : int.Parse(sizeInput);

// Генерация данных

List<int> data = SortManager.GenerateRandomData(size);

Console.WriteLine($"\nСгенерировано {size} случайных чисел.");

SortManager.DisplayData(data, "Исходные данные:");

// Выполнение сортировки

string methodName = GetMethodName(choice);

Console.WriteLine($"\nВыполняется {methodName}...");

// Создаем копию для сортировки

List<int> dataToSort = new List<int>(data);

sortMethods[choice](dataToSort);

SortManager.DisplayData(dataToSort, $"После {methodName}:");

WaitForUser();

}

static string GetMethodName(string choice)

{

return choice switch

{

"1" => "Пузырьковой сортировки",

"2" => "Быстрой сортировки",

"3" => "Сортировки выбором",

"4" => "Сортировки вставками",

"5" => "Сортировки по убыванию",

\_ => "Неизвестный метод"

};

}

static void WaitForUser()

{

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

На рисунке 5 представлен результат выполнения программы.

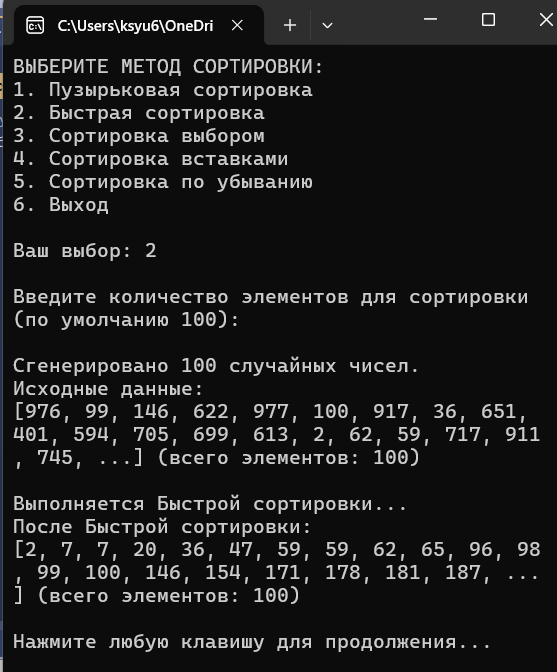


Рисунок 5 – Вывод классов