МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”


Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по домашнему заданию

**Выполнил:**
Студент группы ИУ5Ц-54Б
Цурин А.П.
**Преподаватель:**
Гапанюк Ю.Е.

Москва 2025

## 1. Задания для выполнения

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).
2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.
3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.
4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).
5. В случае создания проекта необходимо детально комментировать код.
6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.
7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

## 2. Листинг программы

**Program.cs**

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace TaskManager
{
    // Перечисление для статусов задачи
    public enum TaskStatusEnum
    {
        Pending,
        InProgress,
        Completed,
        Cancelled
    }

    // Перечисление для типов уведомлений
    public enum NotificationType
    {
        Info,
        Warning,
        Success,
        Error
    }

    // Класс задачи
    public class TaskItem
    {
        public Guid Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public TaskStatusEnum Status { get; set; }
```

```
        public DateTime CreatedAt { get; set; }
        public DateTime? CompletedAt { get; set; }
        public int Priority { get; set; }

        public TaskItem(string title, string description, int priority = 1)
        {
            Id = Guid.NewGuid();
            Title = title ?? throw new ArgumentNullException(nameof(title));
            Description = description ?? string.Empty;
            Status = TaskStatusEnum.Pending;
            CreatedAt = DateTime.UtcNow;
            Priority = priority;
        }

        public void Start()
        {
            if (Status == TaskStatusEnum.Pending)
            {
                Status = TaskStatusEnum.InProgress;
            }
        }

        public void Complete()
        {
            if (Status == TaskStatusEnum.InProgress || Status ==
TaskStatusEnum.Pending)
            {
                Status = TaskStatusEnum.Completed;
                CompletedAt = DateTime.UtcNow;
            }
        }

        public override string ToString()
        {
            return $"[{Status}] {Title} (Приоритет: {Priority})";
        }
    }

    // Интерфейс репозитория
    public interface ITaskRepository
    {
        Task<TaskItem> GetByIdAsync(Guid id);
        Task<List<TaskItem>> GetAllAsync();
        Task<TaskItem> AddAsync(TaskItem task);
        Task<TaskItem> UpdateAsync(TaskItem task);
        Task<bool> DeleteAsync(Guid id);
        Task<List<TaskItem>> GetByStatusAsync(TaskStatusEnum status);
    }

    // Интерфейс уведомлений
    public interface INotificationService
    {
        Task SendAsync(string message, NotificationType type);
```

```
}

// Реализация репозитория в памяти
public class InMemoryTaskRepository : ITaskRepository
{
    private readonly List<TaskItem> _tasks = new List<TaskItem>();
    private readonly object _lock = new object();

    public Task<TaskItem> GetByIdAsync(Guid id)
    {
        lock (_lock)
        {
            var task = _tasks.Find(t => t.Id == id);
            return Task.FromResult(task);
        }
    }

    public Task<List<TaskItem>> GetAllAsync()
    {
        lock (_lock)
        {
            return Task.FromResult(new List<TaskItem>(_tasks));
        }
    }

    public Task<TaskItem> AddAsync(TaskItem task)
    {
        lock (_lock)
        {
            _tasks.Add(task);
            return Task.FromResult(task);
        }
    }

    public Task<TaskItem> UpdateAsync(TaskItem task)
    {
        lock (_lock)
        {
            var index = _tasks.FindIndex(t => t.Id == task.Id);
            if (index >= 0)
            {
                _tasks[index] = task;
                return Task.FromResult(task);
            }
            throw new ArgumentException($"Задача с ID {task.Id} не найдена");
        }
    }

    public Task<bool> DeleteAsync(Guid id)
    {
        lock (_lock)
        {
            var task = _tasks.Find(t => t.Id == id);
```

```
                if (task != null)
                {
                    return Task.FromResult(_tasks.Remove(task));
                }
                return Task.FromResult(false);
            }
        }

        public Task<List<TaskItem>> GetByStatusAsync(TaskStatusEnum status)
        {
            lock (_lock)
            {
                var result = _tasks.FindAll(t => t.Status == status);
                return Task.FromResult(result);
            }
        }
    }

    // Реализация уведомлений в консоль
    public class ConsoleNotificationService : INotificationService
    {
        public Task SendAsync(string message, NotificationType type)
        {
            ConsoleColor color = type switch
            {
                NotificationType.Info => ConsoleColor.Blue,
                NotificationType.Warning => ConsoleColor.Yellow,
                NotificationType.Success => ConsoleColor.Green,
                NotificationType.Error => ConsoleColor.Red,
                _ => ConsoleColor.White
            };

            Console.ForegroundColor = color;
            Console.WriteLine($"[{DateTime.Now:HH:mm:ss}] {type}: {message}");
            Console.ResetColor();

            return Task.CompletedTask;
        }
    }

    // Сервис для работы с задачами
    public class TaskService
    {
        private readonly ITaskRepository _repository;
        private readonly INotificationService _notificationService;

        public TaskService(ITaskRepository repository, INotificationService
notificationService)
        {
            _repository = repository ?? throw new
ArgumentNullException(nameof(repository));
            _notificationService = notificationService ?? throw new
ArgumentNullException(nameof(notificationService));
```

```
        }

        public async Task<TaskItem> CreateTaskAsync(string title, string description,
int priority)
        {
            var task = new TaskItem(title, description, priority);
            var createdTask = await _repository.AddAsync(task);

            await _notificationService.SendAsync(
                $"Создана новая задача: {createdTask.Title}",
                NotificationType.Info
            );

            return createdTask;
        }

        public async Task<TaskItem> CompleteTaskAsync(Guid taskId)
        {
            var task = await _repository.GetByIdAsync(taskId);
            if (task == null)
                throw new ArgumentException($"Задача с ID {taskId} не найдена");

            task.Complete();
            var updatedTask = await _repository.UpdateAsync(task);

            await _notificationService.SendAsync(
                $"Задача выполнена: {updatedTask.Title}",
                NotificationType.Success
            );

            return updatedTask;
        }

        public async Task<List<TaskItem>> GetHighPriorityTasksAsync()
        {
            var allTasks = await _repository.GetAllAsync();
            var highPriorityTasks = new List<TaskItem>();

            foreach (var task in allTasks)
            {
                if (task.Priority >= 8 && task.Status != TaskStatusEnum.Completed)
                {
                    highPriorityTasks.Add(task);
                }
            }

            return highPriorityTasks;
        }

        public async Task<List<TaskItem>> GetTasksByStatusAsync(TaskStatusEnum status)
        {
            return await _repository.GetByStatusAsync(status);
        }
```

```
    }

    // Основная программа
    class Program
    {
        static async Task Main(string[] args)
        {
            Console.WriteLine("=== Система управления задачами на C# ===");
            Console.WriteLine();

            // Создаем зависимости
            ITaskRepository repository = new InMemoryTaskRepository();
            INotificationService notificationService = new
ConsoleNotificationService();

            // Создаем сервис
            var taskService = new TaskService(repository, notificationService);

            bool running = true;

            while (running)
            {
                Console.WriteLine("\nВыберите действие:");
                Console.WriteLine("1. Создать задачу");
                Console.WriteLine("2. Показать все задачи");
                Console.WriteLine("3. Выполнить задачу");
                Console.WriteLine("4. Показать задачи в работе");
                Console.WriteLine("5. Показать задачи с высоким приоритетом");
                Console.WriteLine("0. Выход");
                Console.Write("> ");

                string choice = Console.ReadLine();

                switch (choice)
                {
                    case "1":
                        await CreateTaskAsync(taskService);
                        break;
                    case "2":
                        await ShowAllTasksAsync(taskService);
                        break;
                    case "3":
                        await CompleteTaskAsync(taskService);
                        break;
                    case "4":
                        await ShowInProgressTasksAsync(taskService);
                        break;
                    case "5":
                        await ShowHighPriorityTasksAsync(taskService);
                        break;
                    case "0":
                        running = false;
                        break;
```

```
                default:
                    Console.WriteLine("Неверный выбор");
                    break;
            }
        }

        Console.WriteLine("\nДо свидания!");
    }

    static async Task CreateTaskAsync(TaskService service)
    {
        Console.Write("Введите название задачи: ");
        string title = Console.ReadLine();

        Console.Write("Введите описание: ");
        string description = Console.ReadLine();

        Console.Write("Введите приоритет (1-10): ");
        if (int.TryParse(Console.ReadLine(), out int priority) && priority >= 1 &&
priority <= 10)
        {
            var task = await service.CreateTaskAsync(title, description, priority);
            Console.WriteLine($"Задача создана: {task}");
        }
        else
        {
            Console.WriteLine("Неверный приоритет (должен быть от 1 до 10)");
        }
    }

    static async Task ShowAllTasksAsync(TaskService service)
    {
        var tasks = await service.GetTasksByStatusAsync(TaskStatusEnum.Pending);
        Console.WriteLine("\n=== Все задачи ===");

        if (tasks.Count == 0)
        {
            Console.WriteLine("Задач нет");
            return;
        }

        foreach (var task in tasks)
        {
            Console.WriteLine(task);
        }
    }

    static async Task CompleteTaskAsync(TaskService service)
    {
        Console.Write("Введите ID задачи для выполнения: ");
        if (Guid.TryParse(Console.ReadLine(), out Guid taskId))
        {
            try
```

```
                {
                    var task = await service.CompleteTaskAsync(taskId);
                    Console.WriteLine($"Задача выполнена: {task}");
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Ошибка: {ex.Message}");
                }
            }
            else
            {
                Console.WriteLine("Неверный формат ID");
            }
        }

        static async Task ShowInProgressTasksAsync(TaskService service)
        {
            var tasks = await service.GetTasksByStatusAsync(TaskStatusEnum.InProgress);
            Console.WriteLine("\n=== Задачи в работе ===");

            if (tasks.Count == 0)
            {
                Console.WriteLine("Задач в работе нет");
                return;
            }

            foreach (var task in tasks)
            {
                Console.WriteLine(task);
            }
        }

        static async Task ShowHighPriorityTasksAsync(TaskService service)
        {
            var tasks = await service.GetHighPriorityTasksAsync();
            Console.WriteLine("\n=== Задачи с высоким приоритетом ===");

            if (tasks.Count == 0)
            {
                Console.WriteLine("Задач с высоким приоритетом нет");
                return;
            }

            foreach (var task in tasks)
            {
                Console.WriteLine(task);
            }
        }
    }
}
```

## 3. Результаты работы программы

PS C:\Users\student \project> dotnet.exe run
=== Система управления задачами на C# ===

Выберите действие:
1. Создать задачу
2. Показать все задачи
3. Выполнить задачу
4. Показать задачи в работе
5. Показать задачи с высоким приоритетом
0. Выход
> 1
Введите название задачи: Установка новой ОС
Введите описание: Скачать образ ОС, переместить его на флешку, перезапустить ПК с флешки, пройти шаги по установке ОС
Введите приоритет (1-10): 10
[23:17:52] Info: Создана новая задача: Установка новой ОС
Задача создана: [Pending] Установка новой ОС (Приоритет: 10)

Выберите действие:
1. Создать задачу
2. Показать все задачи
3. Выполнить задачу
4. Показать задачи в работе
5. Показать задачи с высоким приоритетом
0. Выход
> 1
Введите название задачи: Купить книги
Введите описание: Зайти в любой книжный магазин и купить нужные книжки
Введите приоритет (1-10): 6
[23:18:29] Info: Создана новая задача: Купить книги
Задача создана: [Pending] Купить книги (Приоритет: 6)

Выберите действие:
1. Создать задачу
2. Показать все задачи
3. Выполнить задачу
4. Показать задачи в работе
5. Показать задачи с высоким приоритетом
0. Выход
> 2

=== Все задачи ===
[Pending] Установка новой ОС (Приоритет: 10)
[Pending] Купить книги (Приоритет: 6)

Выберите действие:
1. Создать задачу
2. Показать все задачи
3. Выполнить задачу
4. Показать задачи в работе
5. Показать задачи с высоким приоритетом
0. Выход

> 5

=== Задачи с высоким приоритетом ===
[Pending] Установка новой ОС (Приоритет: 10)

Выберите действие:
1. Создать задачу
2. Показать все задачи
3. Выполнить задачу
4. Показать задачи в работе
5. Показать задачи с высоким приоритетом
0. Выход
> 0

До свидания!