

Министерство науки и высшего образования РФ
ФГАОУ ВПО
Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

Отчет по контрольной работе №1
по дисциплине «Объектно-Оrientированное Программирование»

Выполнил:
студент группы БИВТ-24-5

Черных Богдан

Проверил:
Стучилин В. В.

Москва, 2025

ВАРИАНТ 4

Вариант = 27(по журналу) + 2(сдвиг) = 4

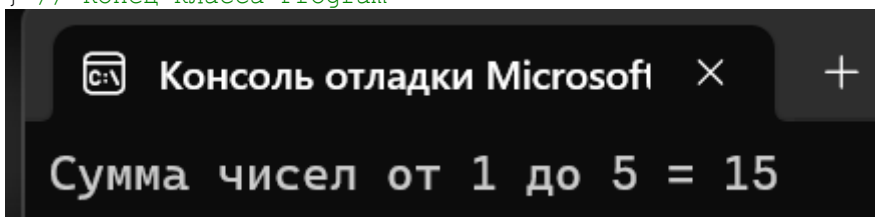
Блок №1.

4.а) Структурный и объектный подходы в программировании. Примеры.

Структурный подход основывается на разбиении задачи на последовательность инструкций, функций и процедур. Программа представляется набором блоков кода, где данные и алгоритмы (функции) зачастую разделены. Главный упор делается на последовательное выполнение команд и управление потоком выполнения (циклы, условия, переходы). Такой подход удобен для небольших программ, но при росте сложности может приводить к избыточной связанности кода и трудностям в поддержке.

Пример:

```
using System; // Подключаем пространство имен System для работы с консолью
class Program // Объявляем класс Program, содержащий точку входа в программу
{ // Начало тела класса Program
    static void Main() // Объявляем метод Main, точку входа программы
    { // Начало метода Main
        int n = 5; // Инициализируем переменную n значением 5
        int result = SumNumbers(n); // Вызываем метод SumNumbers для вычисления
        // суммы чисел от 1 до n
        Console.WriteLine("Сумма чисел от 1 до " + n + " = " + result); // Выводим
        // результат на экран
    } // Конец метода Main
    static int SumNumbers(int limit) // Объявляем метод SumNumbers с параметром
    limit
    { // Начало метода SumNumbers
        int sum = 0; // Инициализируем переменную sum значением 0
        for (int i = 1; i <= limit; i++) // Запускаем цикл от 1 до limit
        // включительно
        { // Начало цикла for
            sum += i; // Прибавляем текущее значение i к переменной sum
        } // Конец цикла for
        return sum; // Возвращаем итоговую сумму
    } // Конец метода SumNumbers
} // Конец класса Program
```



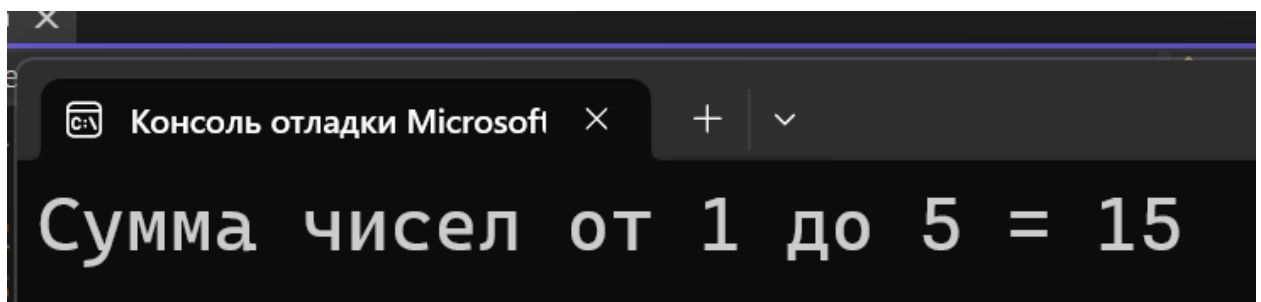
Объектный подход строится вокруг концепции объектов и классов. Класс задаёт «шаблон» – объединяет данные (например свойства, поля) и методы (например функции, процедуры), которые работают с этими данными.

Главные принципы ООП – инкапсуляция, наследование и полиморфизм:

1. Инкапсуляция позволяет скрыть внутреннюю реализацию класса, предоставляя интерфейс для взаимодействия.
2. Наследование позволяет создавать новые классы на основе уже существующих, повторно используя их код.
3. Полиморфизм обеспечивает возможность обработки объектов разных классов через общий интерфейс.

Пример:

```
using System; // Подключаем пространство имен System для работы с консолью
class SumCalculator // Объявляем класс SumCalculator для инкапсуляции логики
вычисления суммы
{ // Начало тела класса SumCalculator
    public int Limit; // Объявляем публичное поле Limit для хранения предельного
значения
    public int CalculateSum() // Объявляем метод CalculateSum для вычисления суммы
чисел от 1 до Limit
    { // Начало метода CalculateSum
        int sum = 0; // Инициализируем переменную sum значением 0
        for (int i = 1; i <= Limit; i++) // Запускаем цикл от 1 до Limit
включительно
        { // Начало цикла for
            sum += i; // Прибавляем текущее значение i к переменной sum
        } // Конец цикла for
        return sum; // Возвращаем итоговую сумму
    } // Конец метода CalculateSum
} // Конец класса SumCalculator
class Program // Объявляем класс Program для запуска приложения
{ // Начало тела класса Program
    static void Main() // Объявляем метод Main, точку входа программы
    { // Начало метода Main
        SumCalculator calculator = new SumCalculator(); // Создаем объект
calculator класса SumCalculator
        calculator.Limit = 5; // Устанавливаем значение поля Limit равным 5
        int result = calculator.CalculateSum(); // Вызываем метод CalculateSum и
сохраняем результат
        Console.WriteLine("Сумма чисел от 1 до " + calculator.Limit + " = " +
result); // Выводим результат на экран
    } // Конец метода Main
} // Конец класса Program
```



4.6) Классы в C#. Объявление классов. Пример.

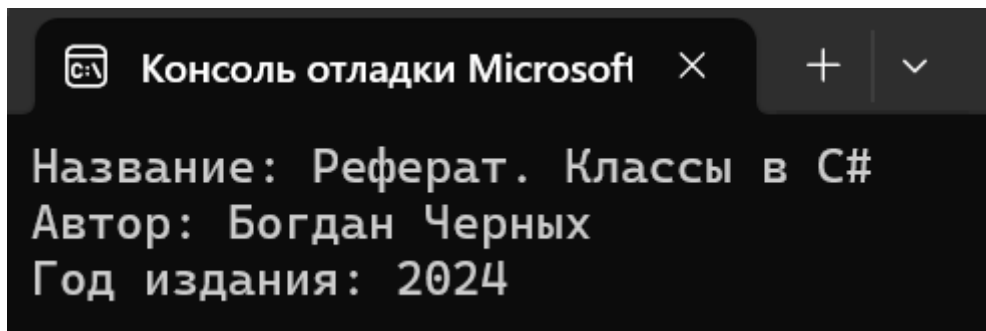
В C# **класс** – это **пользовательский тип данных**, который задаёт структуру объектов (данные) и их поведение (методы). Объявление класса производится с использованием ключевого слова `Class`.

Основные моменты объявления класса:

1. Модификаторы доступа. Обычно классы объявляются с модификатором `public`, чтобы их можно было использовать в других частях программы.
2. Поля. Переменные, описывающие состояние объекта.
3. Методы. Функции, определяющие поведение объекта.
4. Конструкторы. Специальные методы для инициализации нового экземпляра класса.

Пример:

```
using System; // Подключаем пространство имен System для работы с консолью
class Book // Объявляем класс Book для представления книги
{ // Начало тела класса Book
    public string Title; // Объявляем публичное поле Title для хранения названия
    книги
    public string Author; // Объявляем публичное поле Author для хранения имени
    автора
    public int PublicationYear; // Объявляем публичное поле PublicationYear для
    хранения года издания
    public void DisplayBookInfo() // Объявляем метод DisplayBookInfo для вывода
    информации о книге
    { // Начало метода DisplayBookInfo
        Console.WriteLine("Название: " + Title); // Выводим название книги на
        экран
        Console.WriteLine("Автор: " + Author); // Выводим имя автора на экран
        Console.WriteLine("Год издания: " + PublicationYear); // Выводим год
        издания книги на экран
    } // Конец метода DisplayBookInfo
} // Конец класса Book
class Program // Объявляем класс Program для демонстрации работы класса Book
{ // Начало тела класса Program
    static void Main() // Объявляем метод Main, точку входа программы
    { // Начало метода Main
        Book myBook = new Book(); // Создаем объект myBook класса Book
        myBook.Title = "Реферат. Классы в C#"; // Присваиваем объекту значение
        названия книги
        myBook.Author = "Богдан Черных"; // Присваиваем объекту значение имени
        автора
        myBook.PublicationYear = 2024; // Присваиваем объекту значение года
        издания
        myBook.DisplayBookInfo(); // Вызываем метод DisplayBookInfo для вывода
        информации о книге
    } // Конец метода Main
} // Конец класса Program
```



4.в) *Использование классов в консольных приложениях C#.* *Примеры.*

В консольных приложениях на C# классы используются для организации логики программы, разделения кода на модули и реализации принципов ООП.

Пример:

```
using System; // Подключаем пространство имен System для работы с консолью
using System.Collections.Generic; // Подключаем пространство имен для работы с
коллекциями
class Book // Объявляем класс Book для представления книги
{ // Начало тела класса Book
    public string Title; // Объявляем публичное поле Title для хранения названия
книги
    public string Author; // Объявляем публичное поле Author для хранения имени
автора
    public int PublicationYear; // Объявляем публичное поле PublicationYear для
хранения года издания
    public void PrintInfo() // Объявляем метод PrintInfo для вывода информации о
книге
    { // Начало метода PrintInfo
        Console.WriteLine("Название: " + Title); // Выводим название книги на
экран
        Console.WriteLine("Автор: " + Author); // Выводим имя автора на экран
        Console.WriteLine("Год издания: " + PublicationYear); // Выводим год
издания книги на экран
    } // Конец метода PrintInfo
} // Конец класса Book
class Program // Объявляем класс Program для запуска консольного приложения
{ // Начало тела класса Program
    static void Main() // Объявляем метод Main, точку входа программы
    { // Начало метода Main
        List<Book> library = new List<Book>(); // Создаем список library для
хранения объектов Book
        Book book1 = new Book(); // Создаем объект book1 класса Book
        book1.Title = "Реферат, Классы в C#"; // Устанавливаем название книги для
book1
        book1.Author = "Богдан Черных"; // Устанавливаем имя автора для book1
        book1.PublicationYear = 2024; // Устанавливаем год издания для book1
        library.Add(book1); // Добавляем объект book1 в список library
        Book book2 = new Book(); // Создаем объект book2 класса Book
        book2.Title = "Доклад, Русско-Японская война"; // Устанавливаем название
книги для book2
        book2.Author = "Черных Богдан"; // Устанавливаем имя автора для book2
        book2.PublicationYear = 1905; // Устанавливаем год издания для book2
        library.Add(book2); // Добавляем объект book2 в список library
    }
}
```

```

        foreach (Book book in library) // Начинаем цикл для перебора каждого
        объекта Book в списке library
        { // Начало цикла foreach
            book.PrintInfo(); // Вызываем метод PrintInfo для вывода информации о
            текущей книге
            Console.WriteLine("-----"); // Выводим разделительную
            линию между записями
        } // Конец цикла foreach
    } // Конец метода Main
} // Конец класса Program

```

```

Консоль отладки Microsoft
Название: Реферат, Классы в C#
Автор: Богдан Черных
Год издания: 2024
-----
Название: Доклад, Русско-Японская война
Автор: Черных Богдан
Год издания: 1905
-----

```

Блок №2.

4.a) Использование статических методов в Си#. Примеры.

Статические методы принадлежат самому классу, а не его экземплярам. Они удобны для реализации «утилитарных» функций (например, математических расчётов, форматирования данных), когда не требуется состояние объекта. Такие методы можно вызывать без создания объекта класса.

Примеры:

```

using System; // Подключаем пространство имен System для работы с консолью
class FootballUtils // Объявляем класс FootballUtils для футбольных операций со
статическими методами
{ // Начало тела класса FootballUtils
    public static int CalculateTotalGoals(int[] goals) // Объявляем статический
метод CalculateTotalGoals, принимающий массив голов
    { // Начало метода CalculateTotalGoals
        int total = 0; // Инициализируем переменную total для подсчета общего
числа голов
        for (int i = 0; i < goals.Length; i++) // Запускаем цикл для перебора
элементов массива goals
        { // Начало цикла for

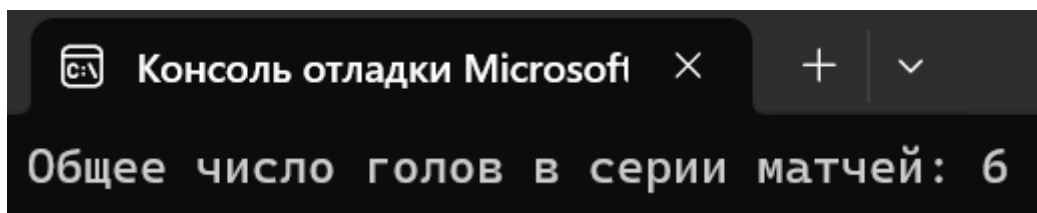
```

```

        total += goals[i]; // Прибавляем количество голов текущего матча к
total
    } // Конец цикла for
    return total; // Возвращаем общее число голов
} // Конец метода CalculateTotalGoals
} // Конец класса FootballUtils
class Program // Объявляем класс Program для запуска приложения
{ // Начало тела класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        int[] matchGoals = new int[] { 2, 3, 1 }; // Инициализируем массив
matchGoals с голами в трех матчах
        int totalGoals = FootballUtils.CalculateTotalGoals(matchGoals); //
Вызываем статический метод CalculateTotalGoals для подсчета общего числа голов
        Console.WriteLine("Общее число голов в серии матчей: " + totalGoals); //
Выводим общее число голов на экран
    } // Конец метода Main
} // Конец класса Program

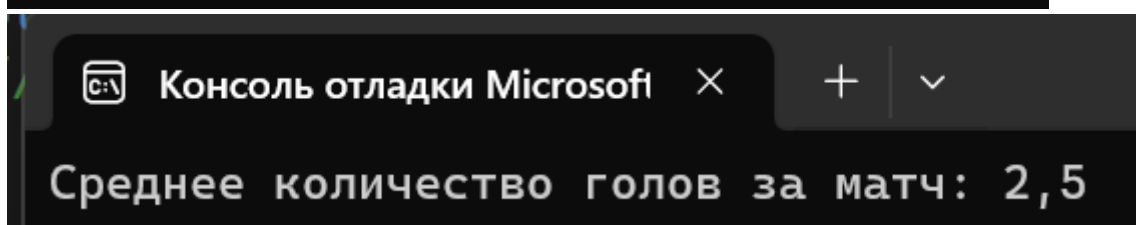
using System; // Подключаем пространство имен System для работы с консолью
class FootballStats // Объявляем класс FootballStats для вычисления статистики
{ // Начало тела класса FootballStats
    public static double CalculateAverageGoals(int[] goals) // Объявляем
статический метод для расчёта среднего числа голов
    { // Начало метода CalculateAverageGoals
        int sum = 0; // Инициализируем переменную sum для подсчёта суммы голов
        for (int i = 0; i < goals.Length; i++) // Запускаем цикл для перебора
массива голов
        { // Начало цикла for
            sum += goals[i]; // Прибавляем количество голов текущего матча к сумме
        } // Конец цикла for
        return (double)sum / goals.Length; // Возвращаем среднее число голов,
приводя сумму к типу double
    } // Конец метода CalculateAverageGoals
} // Конец класса FootballStats
class Program // Объявляем класс Program для запуска приложения
{ // Начало класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        int[] goalsArray = new int[] { 1, 4, 2, 3 }; // Инициализируем массив с
количеством голов за несколько матчей
        double average = FootballStats.CalculateAverageGoals(goalsArray); //
Вызываем статический метод для расчёта среднего количества голов
        Console.WriteLine("Среднее количество голов за матч: " + average); //
Выводим результат расчёта на экран
    } // Конец метода Main
} // Конец класса Program

```



Консоль отладки Microsoft

Общее число голов в серии матчей: 6



Консоль отладки Microsoft

Среднее количество голов за матч: 2,5

4.б) Обработка исключений. Примеры.

Обработка исключений в C# – это механизм, позволяющий перехватывать и корректно реагировать на ошибки (исключения), которые возникают во время выполнения программы, не приводя к её аварийному завершению.

Основные элементы обработки исключений:

1. `try` – блок, в котором размещается код, потенциально способный вызвать исключение.
2. `catch` – блок, который перехватывает и обрабатывает возникшее исключение. Можно использовать несколько блоков `catch` для обработки разных типов исключений.
3. `finally` – блок, содержащий код, который выполнится независимо от того, произошло исключение или нет (например, для освобождения ресурсов).

Примеры:

```
using System; // Подключаем пространство имен System для работы с консолью
class Program // Объявляем класс Program для демонстрации обработки исключений
{ // Начало класса Program
    static void Main() // Точка входа в программу – метод Main
    { // Начало метода Main
        try // Начало блока try для выполнения потенциально опасного кода
        { // Начало блока try
            int totalGoals = 5; // Инициализируем общее число голов
            int matches = 0; // Инициализируем число матчей равным 0 для имитации
            деления на ноль
            int averageGoals = totalGoals / matches; // Пытаемся вычислить среднее
            число голов (возникнет деление на ноль)
            Console.WriteLine("Среднее число голов: " + averageGoals); // Выводим
            результат, если исключения не возникло
        } // Конец блока try
        catch (DivideByZeroException ex) // Ловим исключение деления на ноль
        { // Начало блока catch
            Console.WriteLine("Ошибка: деление на ноль при расчёте среднего
            количества голов."); // Выводим сообщение об ошибке деления на ноль
            Console.WriteLine("Подробности ошибки: " + ex.Message); // Выводим
            подробности возникшего исключения
        } // Конец блока catch
        finally // Блок finally, который выполняется в любом случае
        { // Начало блока finally
            Console.WriteLine("Завершение расчёта статистики команды."); //
            Выводим сообщение о завершении расчёта
        } // Конец блока finally
    } // Конец метода Main
} // Конец класса Program

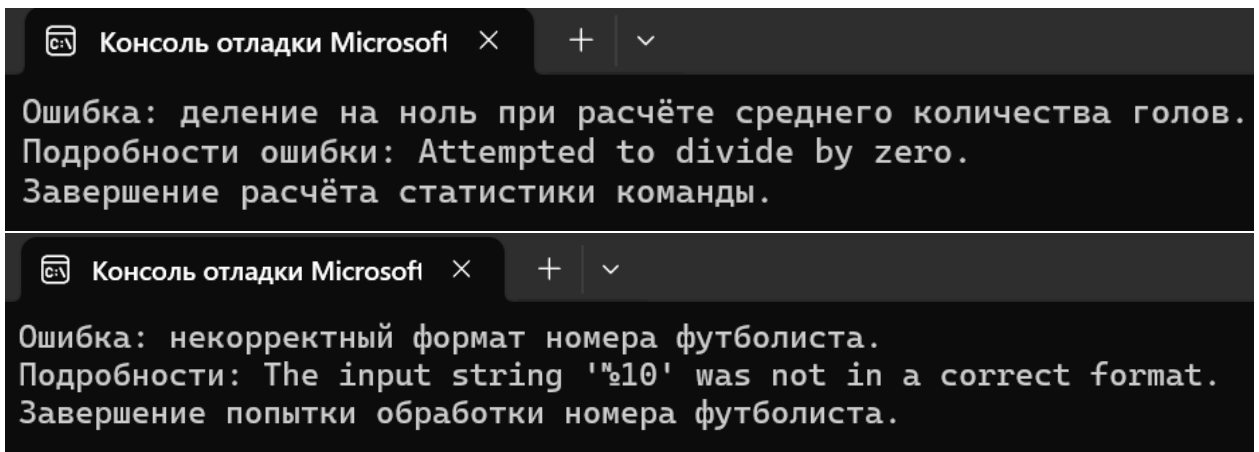
using System; // Подключаем пространство имен System для работы с консолью
class Program // Объявляем класс Program для демонстрации обработки исключений
{ // Начало класса Program
    static void Main() // Точка входа в программу – метод Main
    { // Начало метода Main
        string input = "N10"; // Инициализируем строку input с некорректным
        форматом номера футболиста
        try // Начало блока try для попытки преобразования строки в число
```



```

        { // Начало блока try
            int jerseyNumber = int.Parse(input); // Пытаемся преобразовать строку
input в целое число
            Console.WriteLine("Номер футболиста: " + jerseyNumber); // Выводим
номер футболиста, если преобразование успешно
        } // Конец блока try
        catch (FormatException ex) // Ловим исключение FormatException при
неверном формате строки
        { // Начало блока catch
            Console.WriteLine("Ошибка: некорректный формат номера футболиста.");
// Выводим сообщение о неправильном формате
            Console.WriteLine("Подробности: " + ex.Message); // Выводим
подробности возникшего исключения
        } // Конец блока catch
        finally // Блок finally, который выполняется в любом случае
        { // Начало блока finally
            Console.WriteLine("Завершение попытки обработки номера футболиста.");
// Выводим сообщение о завершении обработки
        } // Конец блока finally
    } // Конец метода Main
} // Конец класса Program

```



4.в) Разработайте приложение, демонстрирующее использование модификаторов доступа для методов класса в C#.

Пример:

```

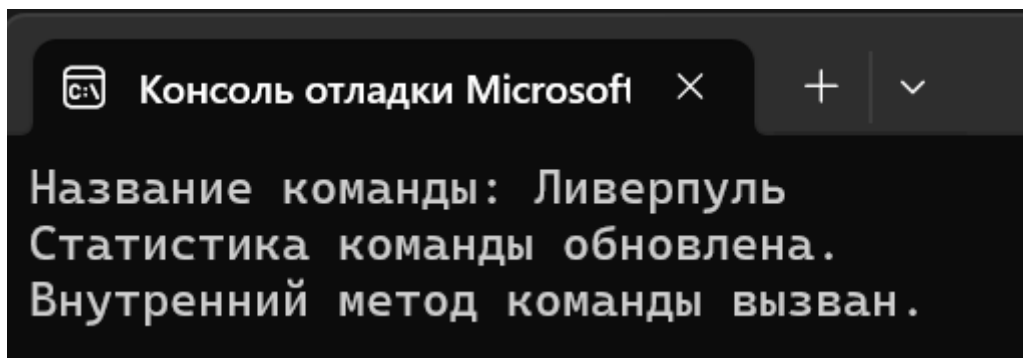
using System; // Подключаем пространство имен System для работы с консолью
class FootballTeam // Объявляем класс FootballTeam для представления футбольной
команды
{ // Начало тела класса FootballTeam
    public string TeamName; // Объявляем публичное поле TeamName для хранения
названия команды
    public void ShowTeamName() // Объявляем публичный метод ShowTeamName для
вывода названия команды
    { // Начало метода ShowTeamName
        Console.WriteLine("Название команды: " + TeamName); // Выводим название
команды на экран
    } // Конец метода ShowTeamName
    private void CalculateBudget() // Объявляем приватный метод CalculateBudget
для расчёта бюджета команды
    { // Начало метода CalculateBudget

```

```

        Console.WriteLine("Расчёт бюджета команды выполнен."); // Выводим
        сообщение о расчёте бюджета
    } // Конец метода CalculateBudget
    protected void UpdateStatistics() // Объявляем защищённый метод
UpdateStatistics для обновления статистики
    { // Начало метода UpdateStatistics
        Console.WriteLine("Статистика команды обновлена."); // Выводим сообщение
        об обновлении статистики
    } // Конец метода UpdateStatistics
    internal void InternalMethod() // Объявляем внутренний метод InternalMethod,
        доступный внутри сборки
    { // Начало метода InternalMethod
        Console.WriteLine("Внутренний метод команды вызван."); // Выводим
        сообщение из внутреннего метода
    } // Конец метода InternalMethod
} // Конец класса FootballTeam
class PremierLeagueTeam : FootballTeam // Объявляем класс PremierLeagueTeam,
наследующий от FootballTeam
{ // Начало тела класса PremierLeagueTeam
    public void AccessProtectedMethod() // Объявляем публичный метод для доступа к
защищённому методу базового класса
    { // Начало метода AccessProtectedMethod
        UpdateStatistics(); // Вызываем защищённый метод UpdateStatistics из
        базового класса
    } // Конец метода AccessProtectedMethod
} // Конец класса PremierLeagueTeam
class Program // Объявляем класс Program для запуска приложения
{ // Начало класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        PremierLeagueTeam team = new PremierLeagueTeam(); // Создаем объект team
        класса PremierLeagueTeam
        team.TeamName = "Ливерпуль"; // Устанавливаем название команды "Ливерпуль"
        team.ShowTeamName(); // Вызываем публичный метод для вывода названия
        команды
        team.AccessProtectedMethod(); // Вызываем метод, который обращается к
        защищённому методу UpdateStatistics
        team.InternalMethod(); // Вызываем внутренний метод InternalMethod для
        демонстрации доступа внутри сборки
        // team.CalculateBudget(); // Попытка вызова приватного метода
        CalculateBudget (недоступен вне класса) - строка закомментирована
    } // Конец метода Main
} // Конец класса Program

```



Блок №3.

4.а) Оператор is. Примеры использования.

Оператор is используется для проверки, соответствует ли объект указанному типу (то есть является ли он экземпляром данного типа или его наследником). Оператор также поддерживает синтаксис «pattern matching», позволяющий не только проверить тип, но и сразу выполнить приведение.

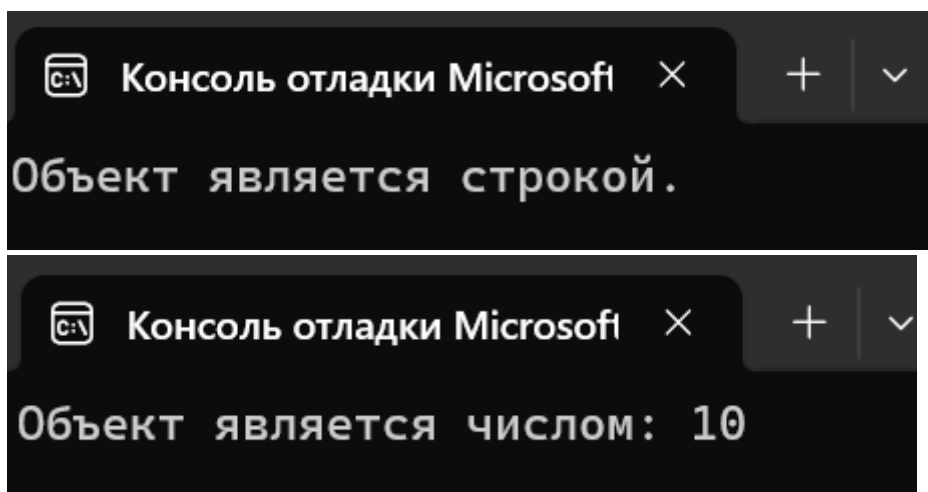
Примеры:

Проверка типа объекта с использованием оператора is

```
using System; // Подключаем пространство имен System для работы с консолью
class Program // Объявляем класс Program для демонстрации оператора is
{ // Начало класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        object obj = "Футбол"; // Инициализируем объект строкой "Футбол"
        if (obj is string) // Проверяем, является ли obj объектом типа string
        { // Начало блока if
            Console.WriteLine("Объект является строкой."); // Выводим сообщение,
// если obj - строка
        } // Конец блока if
    } // Конец метода Main
} // Конец класса Program
```

Использование оператора is с pattern matching для извлечения значения

```
using System; // Подключаем пространство имен System для работы с консолью
class Program // Объявляем класс Program для демонстрации pattern matching
{ // Начало класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        object data = 10; // Инициализируем объект числом 10
        if (data is int number) // Используем оператор is с pattern matching для
// проверки типа и извлечения значения
        { // Начало блока if
            Console.WriteLine("Объект является числом: " + number); // Выводим
// сообщение с извлечённым числом
        } // Конец блока if
    } // Конец метода Main
} // Конец класса Program
```

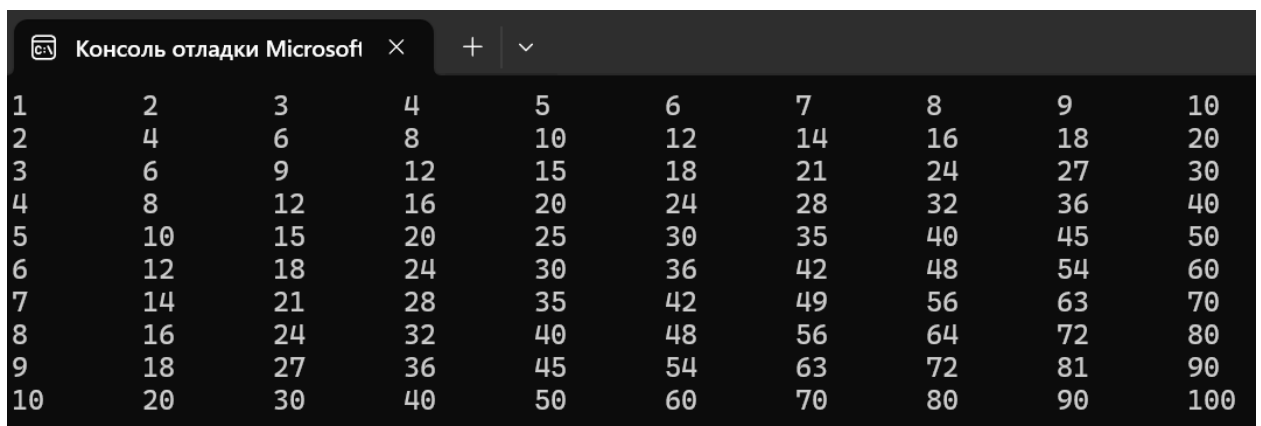


4.б) Создайте программу для заполнения двумерного массива 10×10 элементов в соответствии с «таблицей умножения». Т.е. каждый элемент массива должен содержать значение таблицы

умножения (например, элемент $[2,2] = 4$).

Программа:

```
using System; // Подключаем пространство имен System для работы с консолью
class Program // Объявляем класс Program для создания таблицы умножения
{ // Начало класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        int[,] multiplicationTable = new int[10, 10]; // Создаем двумерный массив
        10x10 для хранения значений таблицы умножения
        for (int i = 0; i < 10; i++) // Запускаем внешний цикл по строкам массива
        { // Начало внешнего цикла
            for (int j = 0; j < 10; j++) // Запускаем внутренний цикл по столбцам
                массива
            { // Начало внутреннего цикла
                multiplicationTable[i, j] = (i + 1) * (j + 1); // Заполняем
                элемент массива произведением (i+1) и (j+1)
            } // Конец внутреннего цикла
        } // Конец внешнего цикла
        for (int i = 0; i < 10; i++) // Запускаем внешний цикл для вывода строк
            таблицы
            { // Начало цикла вывода строк
                for (int j = 0; j < 10; j++) // Запускаем внутренний цикл для вывода
                    элементов строки
                { // Начало цикла вывода элементов
                    Console.Write(multiplicationTable[i, j] + "\t"); // Выводим
                    элемент с символом табуляции для выравнивания
                } // Конец цикла вывода элементов
                Console.WriteLine(); // Переходим на новую строку после вывода текущей
                строки
            } // Конец цикла вывода строк
        } // Конец метода Main
    } // Конец класса Program
```



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

4.в) Режимы открытия файла (FileMode) в классе FileStream. Примеры.

Класс **FileStream** использует перечисление **FileMode**, которое определяет, как будет открываться файл. Основные режимы:

1. CreateNew – Создает новый файл. Если файл уже существует, генерируется исключение.

2. Create – Создает новый файл. Если файл существует, он перезаписывается.
3. Open – Открывает существующий файл. Если файл не существует, генерируется исключение.
4. OpenOrCreate – Открывает файл, если он существует, или создает новый, если его нет.
5. Truncate – Открывает существующий файл и усекает его до нуля (удаляет содержимое). Если файла не существует, генерируется исключение.
6. Append – Открывает файл для добавления данных. Если файл не существует, он создается. Только операция записи доступна, при этом указатель автоматически перемещается в конец файла.

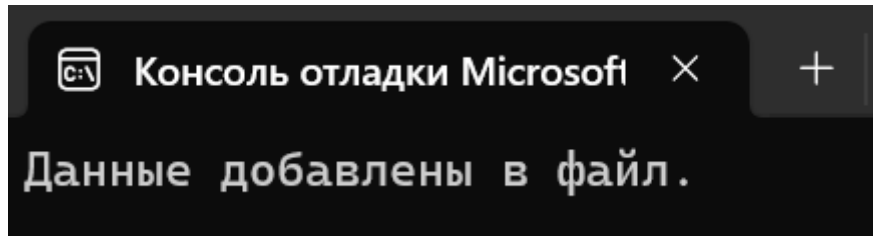
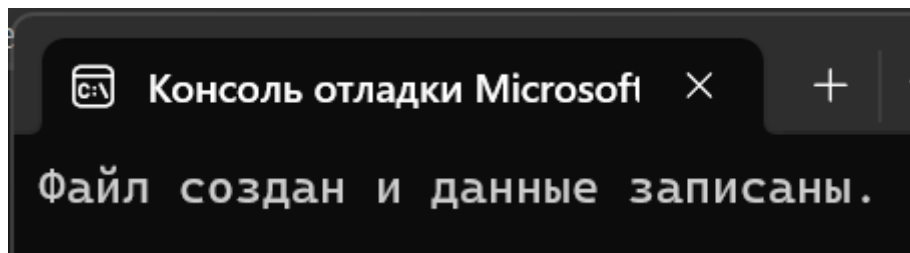
Примеры:

Создание нового файла с использованием FileMode.Create

```
using System; // Подключаем пространство имен System для работы с консолью
using System.IO; // Подключаем пространство имен System.IO для работы с файлами
class Program // Объявляем класс Program для демонстрации работы с FileStream
{ // Начало класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        string filePath = "output.txt"; // Задаем путь к файлу output.txt
        using (FileStream fs = new FileStream(filePath, FileMode.Create)) //
        Открываем файл с режимом Create для создания нового файла
        { // Начало блока using для FileStream
            byte[] data = System.Text.Encoding.UTF8.GetBytes("Запись в файл с
режимом Create."); // Преобразуем строку в массив байтов
            fs.Write(data, 0, data.Length); // Записываем массив байтов в файл
        } // Конец блока using для FileStream, файл автоматически закрывается
        Console.WriteLine("Файл создан и данные записаны."); // Выводим сообщение
        о том, что файл создан и данные записаны
    } // Конец метода Main
} // Конец класса Program
```

Добавление данных в существующий файл с использованием FileMode.Append

```
using System; // Подключаем пространство имен System для работы с консолью
using System.IO; // Подключаем пространство имен System.IO для работы с файлами
class Program // Объявляем класс Program для демонстрации добавления данных в файл
{ // Начало класса Program
    static void Main() // Точка входа в программу - метод Main
    { // Начало метода Main
        string filePath = "output.txt"; // Задаем путь к файлу output.txt
        using (FileStream fs = new FileStream(filePath, FileMode.Append)) //
        Открываем файл с режимом Append для добавления данных в конец файла
        { // Начало блока using для FileStream
            byte[] data = System.Text.Encoding.UTF8.GetBytes(" Добавление данных с
режимом Append."); // Преобразуем строку в массив байтов
            fs.Write(data, 0, data.Length); // Записываем массив байтов в конец
файла
        } // Конец блока using для FileStream, файл автоматически закрывается
        Console.WriteLine("Данные добавлены в файл."); // Выводим сообщение о том,
что данные добавлены в файл
    } // Конец метода Main
} // Конец класса Program
```



Конец работы.