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 №2
по дисциплине «Объектно-Ориентированное Программирование»

Выполнил:
студент группы БИВТ-24-5

Черных Богдан

Проверил:
Стучилин В. В.

Москва, 2025

ВАРИАНТ 3

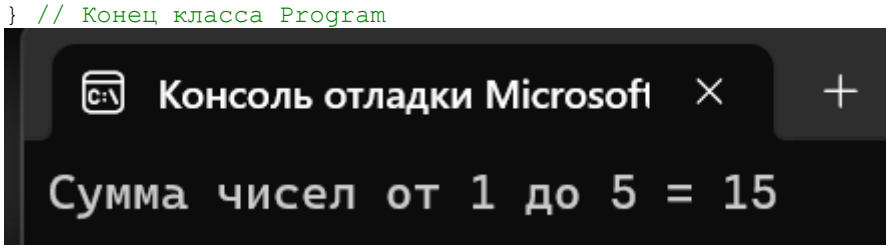
Блок №1.

3.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
```

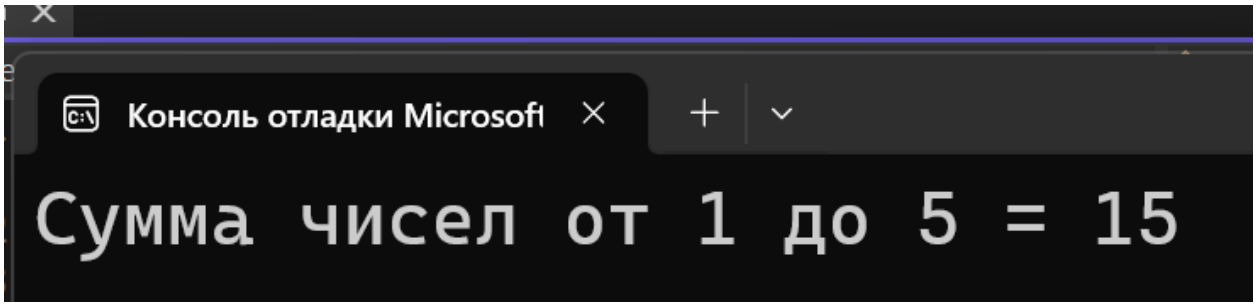


Объектный подход строится вокруг концепции объектов и классов. Класс задаёт «шаблон» – объединяет данные (например свойства, поля) и методы (например функции, процедуры), которые работают с этими данными. Главные принципы ООП – инкапсуляция, наследование и полиморфизм:

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
```



3.б) Классы в C#. Объявление классов. Пример.

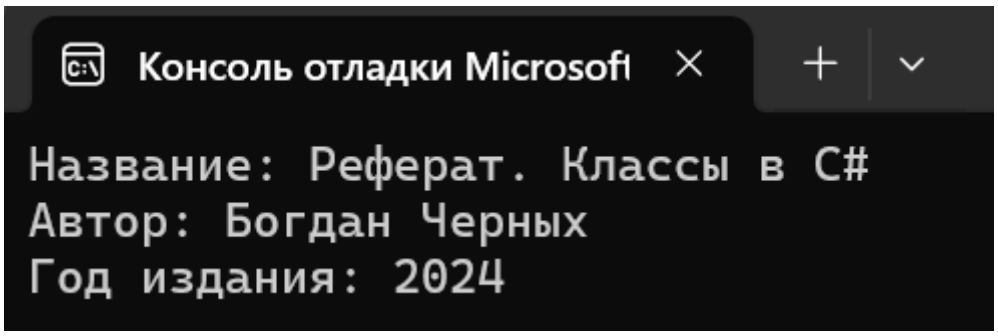
В C# **класс** – это **пользовательский тип данных**, который задаёт структуру объектов (данные) и их поведение (методы). Объявление класса производится с использованием ключевого слова Class.

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
```



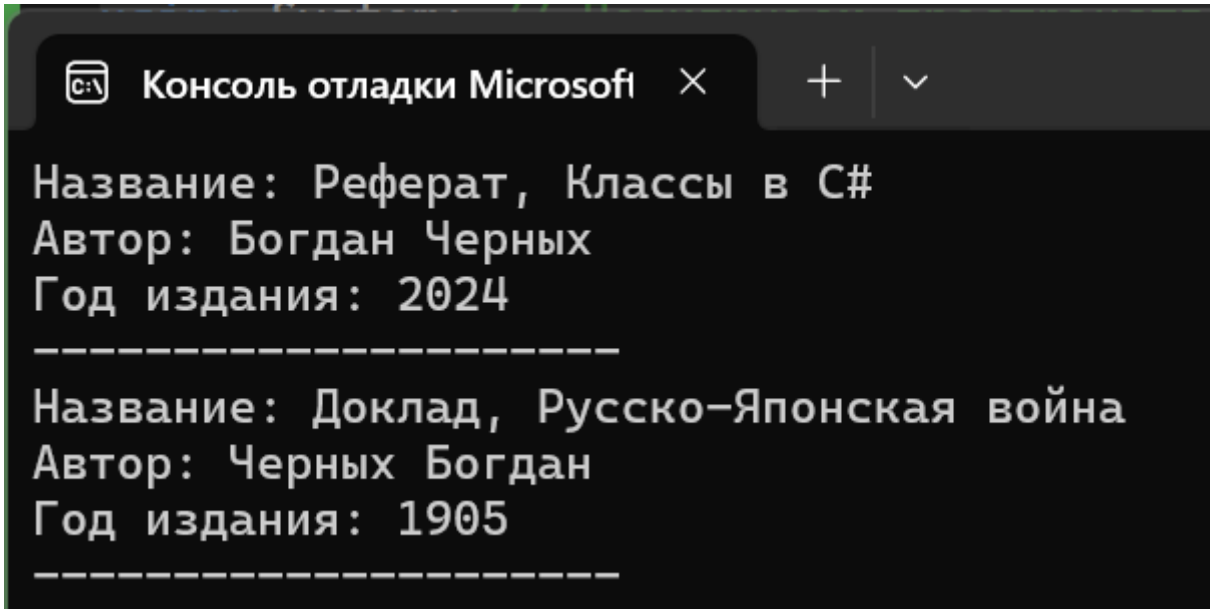
3.в) Использование классов в консольных приложениях Си#. Примеры.

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
```



Блок №2.

3 а) Статические и простые методы в Си#. Примеры.

Статические методы

Статический метод отмечается ключевым словом static и принадлежит типу, а не конкретному объекту.
Такие методы можно вызывать через имя класса без использования оператора new.
Статические методы не имеют доступа к нестатическим (instance) полям или методам класса, они взаимодействуют лишь со статическими членами.

Простые методы

Простой метод объявляется без ключевого слова static и считается членом конкретного экземпляра класса.
Чтобы вызвать инстансный метод, необходимо сначала создать объект класса с помощью оператора new.

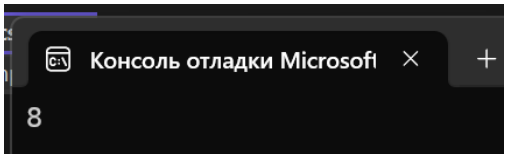
Instance-методы могут обращаться к полям и другим методам того же экземпляра, включая как статические, так и нестатические члены.

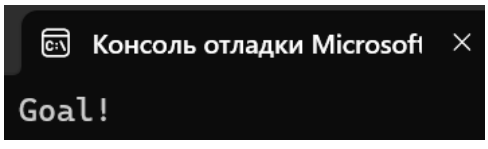
Пример статистического:

```
using System; // Аналог <iostream> для работы с консолью основными функциями

public class Calculator // объявление класса Calculator
{
    public static int Add(int a, int b) // статический метод Add
    {
        return a + b; // возвращаем сумму параметров
    }
}

public class Program // класс Program с точкой входа
{
    public static void Main() // статический метод Main
    {
        int result = Calculator.Add(5, 3); // вызов статического метода Add
        Console.WriteLine(result); // вывод результата на консоль
    }
}
```

Пример простого:

```
using System; // Аналог <iostream> для работы с консолью основными функциями
public class FootballPlayer // класс FootballPlayer
{
    public void Kick() // простой (instance) метод Kick
    {
        Console.WriteLine("Goal!"); // выводим сообщение о голе
    }
}

public class Program // класс Program
{
    public static void Main() // статический метод Main
    {
        FootballPlayer player = new FootballPlayer(); // создаём экземпляр игрока
        player.Kick(); // вызываем метод Kick для удара по мячу
    }
}
```





3.б) Механизм полиморфизма в Си#. Примеры.

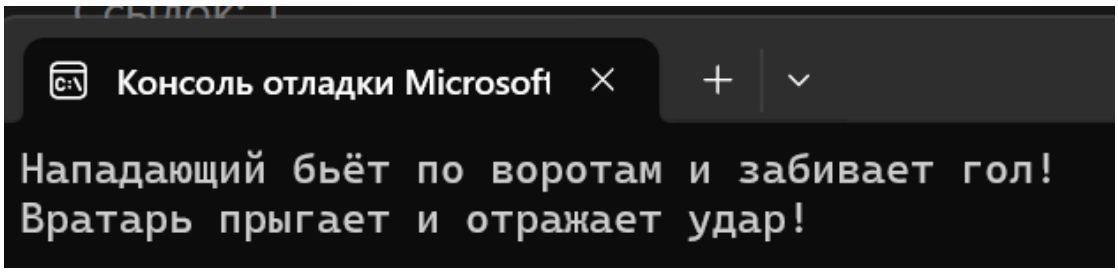
Полиморфизм в C# — это способность объектов разных классов реагировать по-разному на один и тот же вызов метода. При этом обращение к методу выполняется через ссылку на базовый тип, а конкретная реализация метода выбирается в зависимости от реального типа объекта во время выполнения.

Механизм полиморфизма в C#

1. Виртуальные методы (virtual) в базовом классе создают точку расширения.
2. Переопределение методов (override) в производных классах обеспечивает собственную реализацию.
3. Вызов через ссылку на базовый тип вызывает ту реализацию, которая соответствует реальному типу объекта (динамическая диспетчеризация).

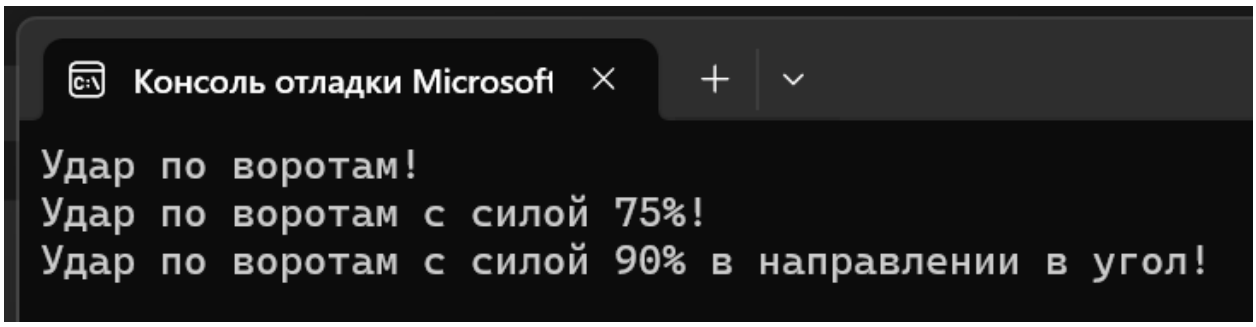
Пример:

```
using System; // Аналог <iostream> для работы с консолью и основными функциями

public class Player                // базовый класс Player
{
    public virtual void Play()     // виртуальный метод Play
    {
        Console.WriteLine("Игрок выполняет действие"); // общий вывод для любого
игрока
    }
}

public class Striker : Player      // класс Striker (нападающий) наследуется от
Player
{
    public override void Play()    // переопределяем метод Play для нападающего
    {
        Console.WriteLine("Нападающий бьёт по воротам и забивает гол!"); //
действие Striker
    }
}

public class Goalkeeper : Player // класс Goalkeeper (вратарь) наследуется от
Player
{
    public override void Play()    // переопределяем метод Play для вратаря
    {
        Console.WriteLine("Вратарь прыгает и отражает удар!"); // действие
Goalkeeper
    }
}

public class Program              // класс Program с точкой входа
{
    public static void Main()      // статический метод Main
    {
        Player player1 = new Striker();     // создаём экземпляр Striker и
присваиваем переменной типа Player
        player1.Play();                     // вызов Striker.Play() через ссылку
Player

        Player player2 = new Goalkeeper(); // создаём экземпляр Goalkeeper и
присваиваем переменной типа Player
        player2.Play();                     // вызов Goalkeeper.Play() через ссылку
Player
    }
}
```



3.в) Разработайте программу, демонстрирующее использование механизма перегрузки методов.

```
using System; // Для работы с консолью и основными функциями

public class FootballPlayer       // класс FootballPlayer
{
    // Перегруженный метод Kick без параметров
    public void Kick()
    {
        Console.WriteLine("Удар по воротам!"); // простой удар по воротам
    }

    // Перегруженный метод Kick с параметром силы удара
    public void Kick(int power)
    {
        Console.WriteLine($"Удар по воротам с силой {power}%!"); // удар с
указанной силой
    }

    // Перегруженный метод Kick с указанием силы и направления
    public void Kick(int power, string direction)
    {
        Console.WriteLine($"Удар по воротам с силой {power}% в направлении
{direction}!"); // удар с силой и направлением
    }
}

public class Program             // класс Program с точкой входа
{
    public static void Main()     // статический метод Main
    {
        FootballPlayer player = new FootballPlayer(); // создаём экземпляр игрока

        player.Kick();                                // вызываем Kick() без
параметров
        player.Kick(75);                              // вызываем Kick(int power)
        player.Kick(90, "в угол");                    // вызываем Kick(int power,
string direction)
    }
}
```



Блок №3.

3.а) Ссылочные типы в Си-шарп. Примеры.

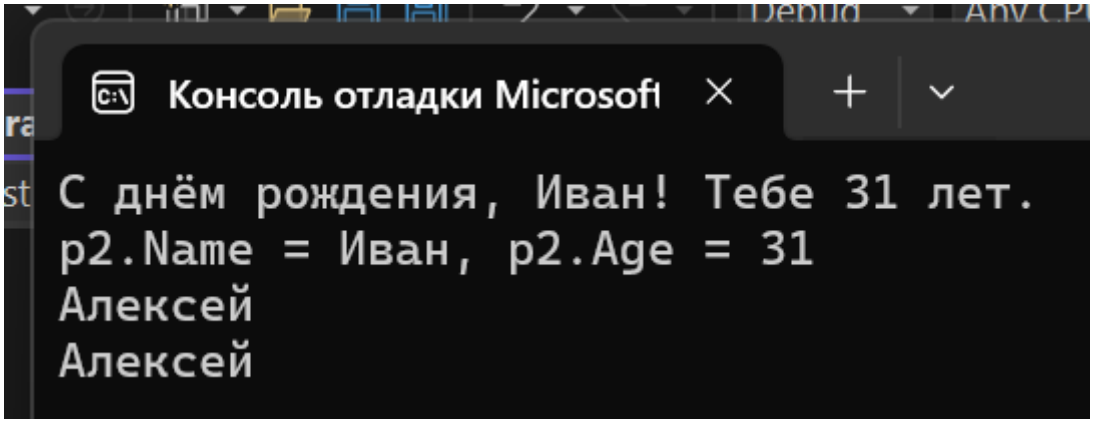
В C# все ссылочные типы представляют объекты, хранящиеся в управляемой куче (heap). Переменная такого типа хранит не само значение объекта, а ссылку на область памяти, где это значение расположено. К основным категориям ссылочных типов относятся:
1. Классы (class)
2. Интерфейсы (interface)
3. Делегаты (delegate)
4. Массивы (T[])
5. Тип string (хотя выглядит как примитив, на деле — ссылочный тип)

Ссылочные типы отличаются от значимых типов (struct, примитивы, перечисления) тем, что при копировании переменной ссылочного типа копируется не само содержимое объекта, а лишь ссылка на него.

Пример:

```
using System; // Аналог <iostream> для работы с консолью и основными функциями

public class Person             // определяем класс Person — ссылочный тип
{
    public string Name;         // поле Name хранит ссылку на строку
    public int Age;             // поле Age — значимый тип внутри объекта

    public Person(string name, int age) // конструктор
    {
        Name = name;          // присваиваем ссылку на строку
        Age = age;            // записываем значение возраста
    }

    public void HaveBirthday()  // метод изменяет состояние объекта
    {
        Age++;                  // увеличиваем возраст
        Console.WriteLine($"С днём рождения, {Name}! Тебе {Age} лет."); // выводим
сообщение
    }
}

public class Program          // класс Program с точкой входа
```

```
{
    public static void ChangeName(Person p) // метод принимает ссылку на Person
    {
        p.Name = "Алексей";     // меняем поле Name через ту же ссылку
    }

    public static void Main()  // статический метод Main
    {
        Person p1 = new Person("Иван", 30); // создаём новый объект в куче
        Person p2 = p1;       // копируем ссылку: p2 указывает на тот же объект

        p1.HaveBirthday();    // через p1: С днём рождения, Иван! Тебе 31 лет.
        Console.WriteLine($"p2.Name = {p2.Name}, p2.Age = {p2.Age}");
        // выводит: p2.Name = Иван, p2.Age = 31 — изменения видны через p2

        ChangeName(p2);       // передаём ссылку в метод
        Console.WriteLine(p1.Name); // выводит: Алексей — имя изменилось и для p1

        p2 = null;            // обнуляем ссылку p2; объект всё ещё жив, т.к. p1
ссылается
        Console.WriteLine(p1.Name); // всё ещё доступно: Алексей
    }
}
```



3.б) Класс HttpWebResponse. Примеры использования

HttpWebResponse — это класс из пространства имён System.Net, представляющий ответ на HTTP-запрос, выполненный с помощью HttpWebRequest или WebRequest. Он позволяет получить статусный код ответа, заголовки, а также содержимое тела ответа в виде потокового объекта (Stream).

Основные свойства и методы

1. StatusCode — возвращает объект HttpStatusCode, указывающий результат запроса (например, OK, NotFound, InternalServerError).
2. StatusDescription — строковое описание статусного кода.
3. Headers — коллекция заголовков ответа (WebHeaderCollection).

4. GetResponseStream() — метод, возвращающий поток (Stream) с данными тела ответа.
5. ContentType — MIME-тип содержимого (например, "application/json").
6. ContentLength — длина содержимого в байтах.

Пример:
Выполнение простого GET-запроса и чтение текста ответа

```
using System;                    // для работы с базовыми типами и консолью
using System.Net;                // для HttpWebRequest и HttpWebResponse
using System.IO;                 // для Stream и StreamReader

public class Program            // основной класс программы
{
    public static void Main()   // метод Main — точка входа
    {
        // Создаём HTTP-запрос по URL
        HttpWebRequest request =
(HttpWebRequest)WebRequest.Create("https://example.com");
        request.Method = "GET"; // задаём метод запроса GET

        // Получаем HTTP-ответ
        using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
        {
            Console.WriteLine($"Status: {(int)response.StatusCode}
{response.StatusDescription}");
            // выводим код и описание статуса

            // Читаем поток ответа
            using (Stream responseStream = response.GetResponseStream())
            using (StreamReader reader = new StreamReader(responseStream))
            {
                string content = reader.ReadToEnd(); // читаем весь текст из
потока
                Console.WriteLine("Response body:");
                Console.WriteLine(content); // выводим тело ответа
            }
        }
    }
}
```

Пример:
Скачивание файла по HTTP и сохранение на диск Csharp.

```
using System;
using System.Net;
using System.IO;

public class Downloader         // класс для загрузки файла
{
    public void DownloadFile(string url, string localPath) // метод скачивания
    {
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
        request.Method = "GET"; // задаём GET-запрос

        using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
        {
            if (response.StatusCode == HttpStatusCode.OK) // проверяем успешный
статус
            {
                using (Stream responseStream = response.GetResponseStream())
                using (FileStream fileStream = File.Create(localPath))
                {
```

```
                    responseStream.CopyTo(fileStream); // копируем данные в файл
                    Console.WriteLine($"File saved to {localPath}");
                    // выводим сообщение об успешном сохранении
                }
            }
            else
            {
                Console.WriteLine($"Error: {(int)response.StatusCode}
{response.StatusDescription}");
                // выводим ошибку при не-OK статусе
            }
        }
    }
}

public class Program
{
    public static void Main()
    {
        Downloader dl = new Downloader(); // создаём экземпляр загрузчика
        dl.DownloadFile("https://example.com/image.png", "image.png");
        // вызываем метод загрузки файла
    }
}
```

3.в) Разработайте приложение для записи данных из БД (любая СУБД).

Пример:

```
using System;                                       // Для работы с консолью и
базовыми типами
using Microsoft.Data.Sqlite;                        // Провайдер для SQLite

public class Program                                 // Основной класс приложения
{
    public static void Main()                        // Точка входа
    {
        // Путь к файлу базы данных SQLite (создастся в папке приложения)
        string connectionString = "Data Source=football.db";

        // Создаём соединение с СУБД
        using (var connection = new SqliteConnection(connectionString))
        {
            connection.Open();                      // Открываем соединение

            // Создаём таблицу Players, если её ещё нет
            var createTableCmd = connection.CreateCommand();
            createTableCmd.CommandText =
                @"CREATE TABLE IF NOT EXISTS Players (
                    Id      INTEGER PRIMARY KEY AUTOINCREMENT,  -- уникальный
идентификатор
                    Name    TEXT NOT NULL,                      -- имя игрока
                    Goals   INTEGER NOT NULL);";               // число голов
            createTableCmd.ExecuteNonQuery();       // Выполняем команду создания
таблицы

            // Запрашиваем данные у пользователя
            Console.Write("Введите имя игрока: ");
            string name = Console.ReadLine();        // Считываем имя
```

```
            Console.Write("Введите число голов: ");
            int goals = int.Parse(Console.ReadLine()); // Считываем и конвертируем
число голов

            // Формируем SQL-запрос для вставки новой записи
            var insertCmd = connection.CreateCommand();
            insertCmd.CommandText =
                @"INSERT INTO Players (Name, Goals)
                  VALUES ($name, $goals);";           // Параметризованный INSERT
            insertCmd.Parameters.AddWithValue("$name", name);
            insertCmd.Parameters.AddWithValue("$goals", goals);
            insertCmd.ExecuteNonQuery();            // Выполняем INSERT

            Console.WriteLine("Игрок успешно добавлен!");

            // Для проверки: читаем и выводим все записи из таблицы
            var selectCmd = connection.CreateCommand();
            selectCmd.CommandText = "SELECT Id, Name, Goals FROM Players;";
            using (var reader = selectCmd.ExecuteReader())
            {
                Console.WriteLine("\nТекущий список игроков:");
                while (reader.Read())                // Пока есть следующая запись
                {
                    int id = reader.GetInt32(0);      // Считываем Id
                    string playerName = reader.GetString(1); // Считываем Name
                    int playerGoals = reader.GetInt32(2);    // Считываем Goals
                    Console.WriteLine($"{id}: {playerName} — {playerGoals}
голов");
                }
            }

            connection.Close();                   // Закрываем соединение
(необязательно при using)
        }
    }
}
```

Принцип работы:

Подключение к SQLite: через SqliteConnection с указанием файла БД (football.db).
Создание таблицы: CREATE TABLE IF NOT EXISTS гарантирует, что таблица Players будет создана один раз.
Вставка данных: читаем имя и число голов из консоли, затем выполняем параметризованный запрос INSERT.
Проверка результата: выполняем SELECT и выводим все записи из таблицы для подтверждения успешной записи.

Конец работы.