Работа с файлами

1. Простая работа с файлами

Мы уже научились хранить данные в памяти с помощью переменных, массивов и объектов. Но у этих способов есть ограничение: всё хранится только **во время работы программы**. Как только программа завершает работу, данные исчезают.

Чтобы информация сохранялась между запусками программы, её нужно записывать **во внешний источник**, например просто в **файлы**.

Файл — это именованная область на диске, в которой можно хранить информацию.

Файлы бывают разных типов:

- **Текстовые** (например, `.txt`) — содержат обычный текст. Их можно открыть и прочитать любым текстовым редактором.
- **Бинарные** (например, `.jpg`, `.exe`, `.mp3`) — содержат данные в машинном формате.

Класс `File`

Для работы с файлами в .NET есть пространство имён `System.IO`. В нём есть несколько классов для работы с файловой системой.

Класс `File` предоставляет **статические методы**, с помощью которых можно **читать и записывать текстовые файлы**.

Запись текста в файл

Метод `File.WriteAllText(path, text)` — записывает строку в файл. Если файл существует, он перезаписывается.

```
using System.IO;

class Program
{
    static void Main()
    {
        string path = "data.txt"; // имя файла
        string text = "Hello, world!";

        File.WriteAllText(path, text);
        Console.WriteLine("Файл создан и записан.");
    }
}
```

После запуска программы в папке с проектом появится файл `data.txt` с содержимым:

```
Hello, world!
```

Чтение текста из файла

Метод `File.ReadAllText(path)` — читает весь текст из файла и возвращает его строкой.

```
string path = "data.txt";

if (File.Exists(path))
{
    string content = File.ReadAllText(path);
    Console.WriteLine("Содержимое файла:");
    Console.WriteLine(content);
}
else
{
    Console.WriteLine("Файл не найден.");
}
```

Дозапись в файл

Метод `File.AppendAllText(path, text)` — добавляет строку в конец файла (вместо перезаписи).

```
File.AppendAllText("data.txt", "\nНовая строка!");
```

Теперь файл `data.txt` будет содержать:

```
Hello, world!
Новая строка!
```

Работа с файлами построчно

Для простых задач, когда мы хотим записать/прочитать **много строк**, удобно использовать статические методы `File.WriteAllLines` и `File.ReadAllLines` из `System.IO`.

Основные методы

- `File.WriteAllLines(string path, string[] contents)` — записывает массив/перечисление строк в файл. Если файл уже существует — он **перезаписывается**.
- `string[] File.ReadAllLines(string path)` — читает все строки файла и возвращает их массивом `string[]`.
- `File.AppendAllLines(string path, string[] contents)` — добавляет строки в конец файла (не перезаписывает).

Когда использовать

- Когда размер файла **не большой** и удобно загрузить всё сразу в память.
- Когда удобно работать с массивом строк (например, CSV, список имён, логов небольшого размера).

Пример. Сохранение и чтение списка пользователей

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string folder = Path.Combine(Directory.GetCurrentDirectory(), "data");
        Directory.CreateDirectory(folder); // убедимся, что папка есть
        string path = Path.Combine(folder, "users.txt");
        // Массив строк — сохраняем
        string[] users =
        {
            "ivanov",
            "petrov",
            "sidorov"
        };
        File.WriteAllLines(path, users);
        Console.WriteLine("users.txt записан.");
        // Читаем обратно
        if (File.Exists(path))
        {
            string[] loaded = File.ReadAllLines(path);
            Console.WriteLine("Считанные имена:");
            foreach (var u in loaded)
                Console.WriteLine($"- {u}");
        }
    }
}
```

Пример. CSV-строки и парсинг

```
// Запись CSV
string[] csvLines = {
    "alice;25;engineer",
    "bob;30;designer",
    "carol;22;intern"
};
File.WriteAllLines("employees.csv", csvLines);

// Чтение и разбор
var lines = File.ReadAllLines("employees.csv");
foreach (var line in lines)
{
    var parts = line.Split(';'); // простой парсер CSV с ; как разделителем
    string name = parts[0];
    int age = int.Parse(parts[1]);
    string role = parts[2];
    Console.WriteLine($"{name} - {age} - {role}");
}
```

Проверка и управление файлами

При реальной работе с файлами часто нужно корректно проверять существование, удалять, копировать и перемещать файлы. Для этого в классе `File` есть методы.

- `File.Exists(path)` — проверяет, существует ли файл; возвращает `true/false`.
- `File.Delete(path)` — удаляет файл. Если файла нет — исключения **нет** (ничего не делает).
- `File.Copy(sourceFileName, destFileName, bool overwrite = false)` — копирует файл. Если `overwrite == false` и файл назначения существует — будет `IOException`. При `true` — перезапишет.
- `File.Move(sourceFileName, destFileName)` — перемещает (или переименовывает) файл. Если файл назначения существует — `IOException`.

Пример. Резервная копия отчёта

```
using System;
using System.IO;

class BackupExample
{
    static void Main()
    {
        string src = "report.txt";
        string backupFolder = Path.Combine(Directory.GetCurrentDirectory(), "backups");
        Directory.CreateDirectory(backupFolder);

        string dest = Path.Combine(backupFolder, "report_backup.txt");

        if (!File.Exists(src))
        {
            Console.WriteLine("Исходного файла нет.");
            return;
        }

        // Копируем с перезаписью, если нужно
        File.Copy(src, dest, overwrite: true);
        Console.WriteLine("Создана резервная копия: " + dest);
    }
}
```

Пример. Безопасное перемещение (переименование) с проверкой

```
string source = "temp.txt";
string dest = "archive\\temp_old.txt";
Directory.CreateDirectory(Path.GetDirectoryName(dest) ?? ".");

// Если целевой файл уже есть — удалим его сначала
if (File.Exists(dest))
{
    File.Delete(dest);
}

File.Move(source, dest);
Console.WriteLine("Файл перемещён.");
```

2. Работа с папками

Класс `Directory` позволяют управлять папками: проверять, создавать, перечислять файлы и подпапки, удалять и т.п.

Основные методы `Directory`

- `Directory.Exists(path)` — возвращает `true`, если папка существует.
- `Directory.CreateDirectory(path)` — создаёт папку (включая все отсутствующие родительские папки). Если папка уже есть — **ничего не делает** и не кидает исключение.
- `Directory.GetFiles(path)` — возвращает массив строк с путями к файлам в папке (без подпапок по умолчанию).
- `Directory.GetDirectories(path)` — возвращает массив путей к подпапкам.
- `Directory.GetFileSystemEntries(path)` — возвращает и файлы, и папки.

Пример. Создать папку, записать файл, перечислить содержимое

```
string baseDir = Path.Combine(Directory.GetCurrentDirectory(), "out");
// Создаст папку "out" (и родительские папки, если надо)
Directory.CreateDirectory(baseDir);
Console.WriteLine($"Directory created: {baseDir}");
// Записать несколько файлов (используем File для простоты)
File.WriteAllText(Path.Combine(baseDir, "a.txt"), "one");
File.WriteAllText(Path.Combine(baseDir, "b.log"), "two");
File.WriteAllText(Path.Combine(baseDir, "c.txt"), "three");
 // Перечислить файлы (только верхний уровень)
string[] files = Directory.GetFiles(baseDir);
Console.WriteLine("Files in directory:");
foreach (var f in files)
    Console.WriteLine(" - " + Path.GetFileName(f));
// Перечислить папки (пока их нет)
var dirs = Directory.GetDirectories(baseDir);
Console.WriteLine("Subdirectories: " + dirs.Length);
// Создадим подпапку и переместим файл туда
string sub = Path.Combine(baseDir, "logs");
Directory.CreateDirectory(sub);
File.Move(Path.Combine(baseDir, "b.log"), Path.Combine(sub, "b.log"));
// Рекурсивный список всех файлов
Console.WriteLine("All files recursively:");
foreach (var f in Directory.GetFiles(baseDir, "*", SearchOption.AllDirectories))
    Console.WriteLine(" > " + f);
```

Удаление папки

- `Directory.Delete(path)` — удаляет **пустую** папку.
- `Directory.Delete(path, recursive: true)` — удаляет папку и всё её содержимое (включая подпапки).

```
if (Directory.Exists(baseDir))
{
    Directory.Delete(baseDir, recursive: true);
    Console.WriteLine("Deleted " + baseDir);
}
```

3. Пути к файлам

Абсолютный vs относительный путь

- **Абсолютный путь** указывает полный путь от корня файловой системы:
  - Windows: `C:\Projects\app\data\file.txt`
  - Linux/macOS: `/home/user/app/data/file.txt`
- **Относительный путь** задаётся относительно **текущей рабочей директории (current working directory)**, например `data\file.txt` или `./data/file.txt`.

Используйте `Path.Combine` для формирования путей, а `Path.GetFullPath` чтобы получить абсолютный путь.

```
string relative = Path.Combine("data", "file.txt"); // "data/file.txt" или "data\file.txt"
string abs = Path.GetFullPath(relative);
Console.WriteLine(abs);
```

Текущая папка: `Directory.GetCurrentDirectory()`

- Возвращает текущую рабочую директорию процесса. Она может **отличаться** от места, где лежит exe.
- `Environment.CurrentDirectory` даёт то же значение и может быть изменена во время выполнения (`Environment.CurrentDirectory = "..."`).
- Так же текущую рабочую директорию можно изменить с помощью метода `Directory.SetCurrentDirectory()`

```
Console.WriteLine("CurrentDirectory: " + Directory.GetCurrentDirectory());
```

Директория запуска

- `AppDomain.CurrentDomain.BaseDirectory` возвращает путь **к каталогу, откуда загружено приложение** (обычно папка с .exe). Это чаще подходит для поиска ресурсов, расположенных вместе с приложением.
- Отличие:
  - `GetCurrentDirectory()` — возвращает путь к текущему рабочему каталогу приложения, который может меняться в процессе выполнения.
  - `BaseDirectory` — возвращает путь к каталогу, в котором установлено приложение, и это значение обычно не меняется после запуска.

```
Console.WriteLine("BaseDirectory: " + AppDomain.CurrentDomain.BaseDirectory);
```

Правильное объединение путей: `Path.Combine(...)`

Никогда не склеивайте строки вручную с `'\'` или `'/'`. Используйте `Path.Combine` — он корректно выставит разделители для текущей ОС.

```
string dataDir = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "data");
string filePath = Path.Combine(dataDir, "users.txt");
```

`Path.Combine` умеет объединять несколько сегментов: `Path.Combine(a, b, c)`.

Полезные методы `Path`

- `Path.GetFileName(path)` — имя файла с расширением.
- `Path.GetFileNameWithoutExtension(path)`
- `Path.GetDirectoryName(path)` — путь к директории.
- `Path.GetExtension(path)` — расширение файла.
- `Path.GetFullPath(path)` — получить абсолютный путь.
- `Path.GetTempPath()` — временная папка ОС.
- `Path.DirectorySeparatorChar` — системный разделитель (`\` или `/`).

Примеры практического использования

```
// Собираем путь к файлу в подпапке рядом с исполняемым файлом
string dataDir = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "data");
Directory.CreateDirectory(dataDir); // гарантируем существование
string logFile = Path.Combine(dataDir, "app.log");

File.AppendAllText(logFile, $"{DateTime.Now}: app started");
```

Практическое задание

Что нужно сделать:

1. **Продолжайте работу над проектом TodoList**
   В этом задании вы научитесь работать с файлами и директориями с помощью классов `File`, `Directory`, `Path` для сохранения состояния программы и восстановления данных при запуске.
2. **Создайте статический класс** `FileManager` для работы с файлами и папками.
   В классе должны быть методы:
   - `EnsureDataDirectory(string dirPath)` — проверяет существование папки, в которой должны располагаться файлы, если её нет — создаёт её с помощью `Directory.CreateDirectory`.
   - `SaveProfile(Profile profile, string filePath)` — сохраняет данные пользователя в `profile.txt`.
   - `LoadProfile(string filePath) : Profile` — загружает данные пользователя из `profile.txt`.
   - `SaveTodos(TodoList todos, string filePath)` — сохраняет задачи в CSV-файл `todo.csv`.
   - `LoadTodos(string filePath) : TodoList` — загружает задачи из CSV-файла.
3. **Формат CSV-файла для задач**:
   - Столбцы: `Index;Text;IsDone;LastUpdate`
   - Пример строки:
   `csharp 0;"Купить продукты";false;2025-08-21T14:30:00 1;"Сделать домашку";true;2025-08-20T10:00:00`
   - **Совет по содержимому с** `;` **и переносами строк**:
   - Обрамляйте текст кавычками `"`
   - Заменяйте переносы `\n` на специальную последовательность `\\n`
   - При чтении — делайте обратное преобразование

4. **Интеграция с командами**:
   - Все методы, которые изменяют `TodoList` (`Add`, `Done`, `Update`, `Delete` и т.п.), после выполнения должны вызывать:
     ```
     FileManager.SaveTodos(todos, todoFilePath);
     ```
   - Команда `Profile` после указание данных пользователя должна вызывать:
     ```
     FileManager.SaveProfile(profile, profileFilePath);
     ```
5. **Загрузка данных при запуске программы**:
   - Перед началом главного цикла команд проверьте существование папки с помощью `FileManager.EnsureDataDirectory`.
   - Если файлы существуют, загрузите их с помощью `LoadProfile` и `LoadTodos`.
   - Если файлов нет — создайте новые объекты `Profile` и `TodoList` и создайте файлы `profile.txt` и `todo.csv`.
6. **Обязательное использование классов `File`, `Directory`, `Path`**:
   - Для формирования путей используйте `Path.Combine`.
   - Перед чтением и записью файлов делайте проверки `File.Exists`.
7. Делайте коммиты после **каждого изменения**. Один большой коммит будет оцениваться в два раза ниже.
8. Обновите **README.md** — добавьте описание новых возможностей программы.
9. Сделайте push изменений в GitHub.

Подсказки для реализации:

1. **Экранирование текста для CSV**:

```
string EscapeCsv(string text) =>
    "\"" + text.Replace("\"", "\"\"").Replace("\n", "\\n") + "\"";

string UnescapeCsv(string text) =>
    text.Trim('"').Replace("\\n", "\n").Replace("\"\"", "\"");
```

2. **Создание папки и файлов при запуске**:

```
FileManager.EnsureDataDirectory(dataDirPath);
if (!File.Exists(profilePath)) File.WriteAllText(profilePath, "");
if (!File.Exists(todoPath)) File.WriteAllText(todoPath, "");
```