Streams, Files and Directories

File Types, Using Streams and Manipulating Files

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#csharp-advanced

Table of Contents

1. What are **Streams**?
2. **Readers** and **Writers**
3. **File Streams**
4. **File** Class
5. **Directory** Class

Streams: Basic Concepts

What are Streams?

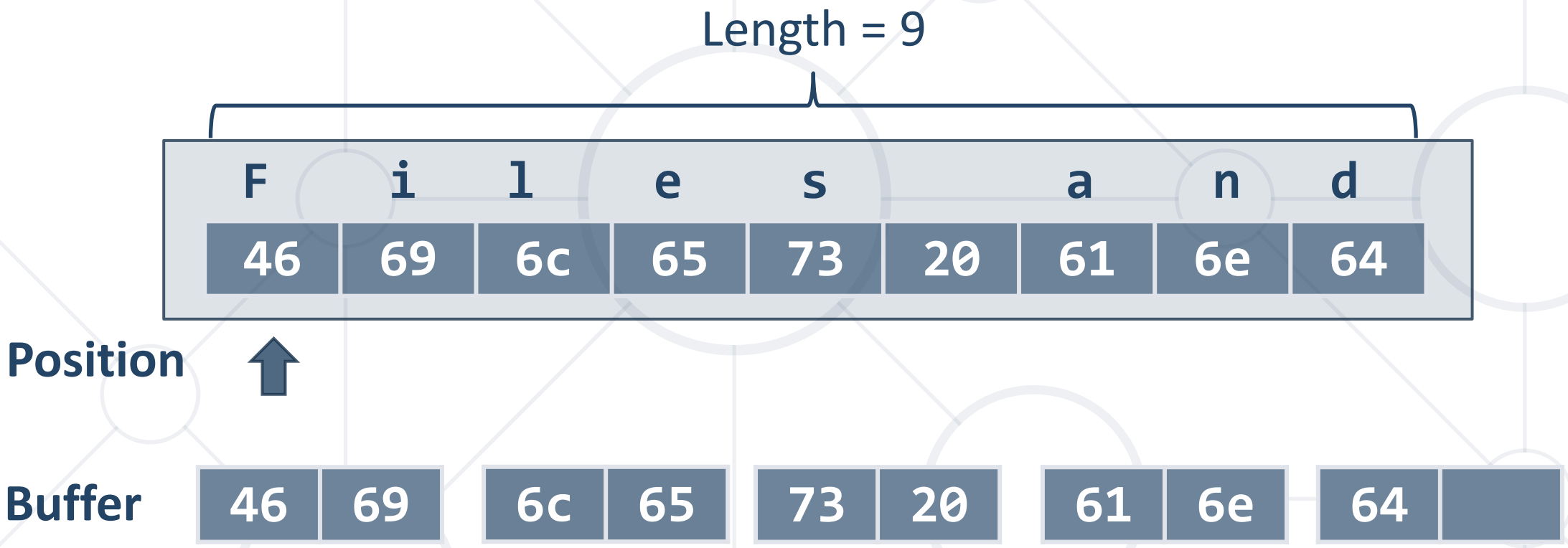
What is a Stream?

- In programming, **streams** are used to **transfer data** between two endpoints (apps / devices / programs)
- Developers use a **stream** to:
  - **Read** (receive) data
  - **Write** (send) data

Stream Basics

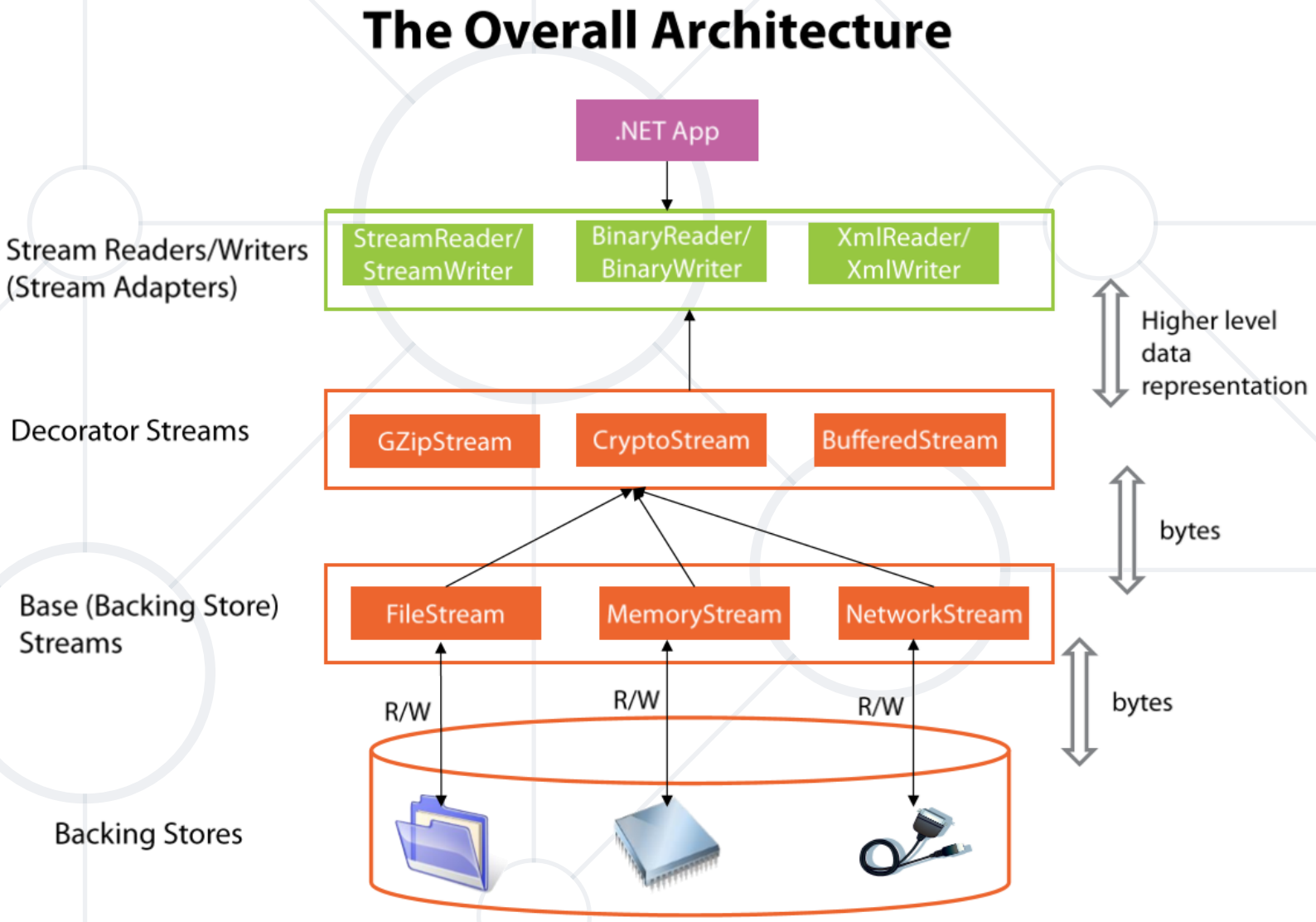
- **Streams** are means for **transferring** (reading and writing) **data**
  - Example: downloading a file from Internet uses streams
- Streams are ordered **sequences of bytes**
  - Provide **sequential** access to its elements (follow the FIFO rule)
- Different types of streams are access different data sources:
  - **File** streams, **network** streams, **memory** streams and others
- Typical use scenario: **open** a stream → **read** / **write** → **close**
- Streams use **buffering**: data is sent and comes in **chunks**

Streams and Buffering – Example



- **Position** is the current offset from the stream start
- **Buffer** keeps **n** bytes of the stream from the current position

Stream Types in .NET



Text Readers and Writers

Readers and Writers in C#

Using StreamReader

- **StreamReader** in C# reads **text** from a file / stream
- The **using(…)** statement closes properly the stream at the end

```
var reader = new StreamReader(fileName);
using (reader)
{
  // Use the reader here, e.g.
  string line = reader.ReadLine();
}
```

Example: Reading a Text File

- Read the content from a text file `input.txt`
- Print on the console each **line number** + **line text** (start from 1)

```
First line
Second Line
Third line
```

```
1. First line
2. Second Line
3. Third line
```

Example: Reading a Text File – C# Code

```
var reader = new StreamReader("../../../input.txt");
using (reader)
{
  int counter = 1;
  while (true)
  {
    string line = reader.ReadLine();
    if (line == null)
      break;
    Console.WriteLine(++counter + ". " + line);
  }
}
```

Using StreamWriter

- **StreamWriter** in C# writes **text** to a file / stream
- The **using(…)** statement closes properly the stream at the end

```
var writer = new StreamWriter(fileName);
using (writer)
{
  // Use the writer here, e.g.
  writer.WriteLine("Some text");
}
```

Problem: Odd Lines

- Read the content from a text file **input.txt**
- Print the **odd lines** in a text file **output.txt**
- Counting starts from **0**

```
Two households, both alike in dignity,
In fair Verona, where we lay our scene,
From ancient grudge break to new mutiny,
Where civil blood makes civil hands unclean.
```

```
In fair Verona, where we lay our scene,
Where civil blood makes civil hands unclean.
```

Solution: Odd Lines

```
var reader = new StreamReader("input.txt");
using (reader) {
  int counter = 0;
  string line = reader.ReadLine();
  using (var writer = new StreamWriter("output.txt")) {
    while (line != null)
      if (counter % 2 == 1)
        writer.WriteLine(line);
    counter++;
    line = reader.ReadLine();
  }
}
```

Problem: Line Numbers

- Read the text file **input.txt**
- Insert a **line number** in front of each line of the file
- Save the result in a text file **output.txt**

```
Two households, both alike in dignity,
In fair Verona, where we lay our scene,
From ancient grudge break to new mutiny,
Where civil blood makes civil hands unclean.
```

```
1. Two households, both alike in dignity,
2. In fair Verona, where we lay our scene,
3. From ancient grudge break to new mutiny,
4. Where civil blood makes civil hands unclean.
```

Solution: Line Numbers

```
using (var reader = new StreamReader("input.txt"))
{
  string line = reader.ReadLine();
  int counter = 1;
  using (var writer = new StreamWriter("output.txt"))
    while (line != null)
    {
      writer.WriteLine($"{counter}. {line}");
      line = reader.ReadLine();
      counter++;
    }
}
```

Open → Read → Close with Try-Catch-Finally

```
StreamReader reader = null;
int linesCount = 0;
try {
  reader = new StreamReader("input.txt");
  while (reader.ReadLine() != null)
    linesCount++;
  Console.WriteLine("Lines count: {0}", linesCount);
}
catch (Exception ex) {
  Console.Error.WriteLine("Error reading file: {0}", ex);
}
finally {
  if (reader != null) reader.Close();
}
```

Instead of `try-finally`, you can use `using(reader)`

Base Streams in .NET

System.IO.Stream

The System.IO.Stream Class

- The base class for all streams is **`System.IO.Stream`**
  - Provides the basic read / write functionality
  - **`Read(buffer)`**, **`Write(buffer, offset, count)`**
- Some streams do not support read, write or positioning operations
  - Properties **`CanRead`**, **`CanWrite`** and **`CanSeek`** are provided
  - Streams which support positioning, have also the properties **`Position`** and **`Length`**

Methods of System.IO.Stream Class

- `int Read(byte[] buffer, int offset, int count)`
  - Read as many as **`count`** bytes from input stream, starting from the given **`offset`** position
  - Returns the number of read bytes or 0, if end of stream is reached
  - Can freeze for undefined time while reading at least 1 byte
  - Can read less than the claimed number of bytes

F	i	l	e	s		a	n	d
46	69	6c	65	73	20	61	6e	64

Methods of System.IO.Stream Class

- `Write(byte[] buffer, int offset, int count)`
  - Writes a sequence of `count` bytes to an output stream, starting from the given `offset` position
  - Can freeze for undefined time, until it sends all bytes to their destination
- `Flush()`
  - Sends the internal buffers data to its destination (data storage, I/O device, etc.)

Methods of System.IO.Stream Class

- `Close()`
  - Calls `Flush()`
  - Closes the connection to the device (mechanism)
  - Releases the used resources
- `Seek(offset, SeekOrigin)`
  - Moves the position (if supported) with given offset towards the beginning, the end or the current position

File Streams

Reading / Writing Binary Files

File Streams

- **File streams** reads / writes sequences of bytes from a file
- **Creating** a new binary file:

```
using (var fs = new FileStream("file.bin", FileMode.Create))
{
  // Write to the file: fs.Write(byte[]) …
}
```

- **Opening** an existing file

```
using (var fs = new FileStream("file.bin", FileMode.Open))
{  // Read from file or write to the file …  }
```

Writing Text to File – Example

```
string text = "Кирилица";
var fileStream =
  new FileStream("log.txt", FileMode.Create);
using(fileStream)
{
  byte[] bytes = Encoding.UTF8.GetBytes(text);
  fileStream.Write(bytes, 0, bytes.Length);
}
```

`Encoding.UTF8.GetBytes()` returns the underlying bytes of the characters

Encrypt / Decrypt File with XOR

```
using (var fin = new FileStream("example.png", FileMode.Open))
using (var fout = new FileStream("example-encrypted.png", FileMode.Create))
{
  byte[] buffer = new byte[4096];
  while (true)
  {
    int bytesRead = fin.Read(buffer);
    if (bytesRead == 0) break;
    const byte secret = 183;
    for (int i = 0; i < bytesRead; i++)
      buffer[i] = (byte) (buffer[i] ^ secret);
    fout.Write(buffer, 0, bytesRead);
  }
}
```

Encrypting the read bytes with the constant parameter `secret` using **XOR** operator

https://gist.github.com/nakov/1d39c4513cff83b8a735d7dc883dfe18

.NET API for Easily Working with Files

File Class in .NET

Reading Text Files

- File.ReadAllText() → string - reads a text file at once

```
using System.IO;
…
string text = File.ReadAllText("file.txt");
```

- File.ReadAllLines() → string[] - reads a text file's lines

```
using System.IO;
…
string[] lines = File.ReadAllLines("file.txt");
```

Writing Text Files

- **Writing** a **string** to a text file:

```
File.WriteAllText("output.txt", "Files are fun :)");
```

- **Writing** a **sequence** of strings to a text file, at separate lines:

```
string[] names = { "peter", "irina", "george", "mary" };
File.WriteAllLines("output.txt", names);
```

- **Appending** additional text to an existing file:

```
File.AppendAllText("output.txt", "\nMore text\n");
```

Reading / Writing Binary Files

- Writing a **byte[]** to a text file:

```
using System.IO;
…
byte[] bytesToWrite = { 0, 183, 255 };
File.WriteAllBytes("output.txt", bytesToWrite);
```

- Reading a binary file into **byte[]**:

```
using System.IO;
…
byte[] bytesRead = File.ReadAllBytes("binaryFile.txt");
```

.NET API for Working with Directories

Directory Class in .NET

Basic Directory Operations

- **Creating** a directory (with all its subdirectories at the specified path), unless they already exists:

```
Directory.CreateDirectory("TestFolder");
```

- **Deleting** a directory (with its contents):

```
Directory.Delete("TestFolder", true);
```

- **Moving** a file or a directory to a new location:

```
Directory.Move("Test", "New Folder");
```

Listing Directory Contents

- **GetFiles()**
  - Returns the names of the files in the specified directory (including their paths)

```
string[] filesInDir = Directory.GetFiles("TestFolder");
```

- **GetDirectories()**
  - Returns the names of the subdirectories (including their paths) in the specified directory

```
string[] subDirs = Directory.GetDirectories("TestFolder");
```

Problem: Calculate Folder Size

- You are given a folder named **`TestFolder`**
- Calculate the **size of all files in the folder** (with its subfolders)
- Print the result in a file "**`output.txt`**" in **megabytes**

output.txt
5.16173839569092

Solution: Calculate Folder Size

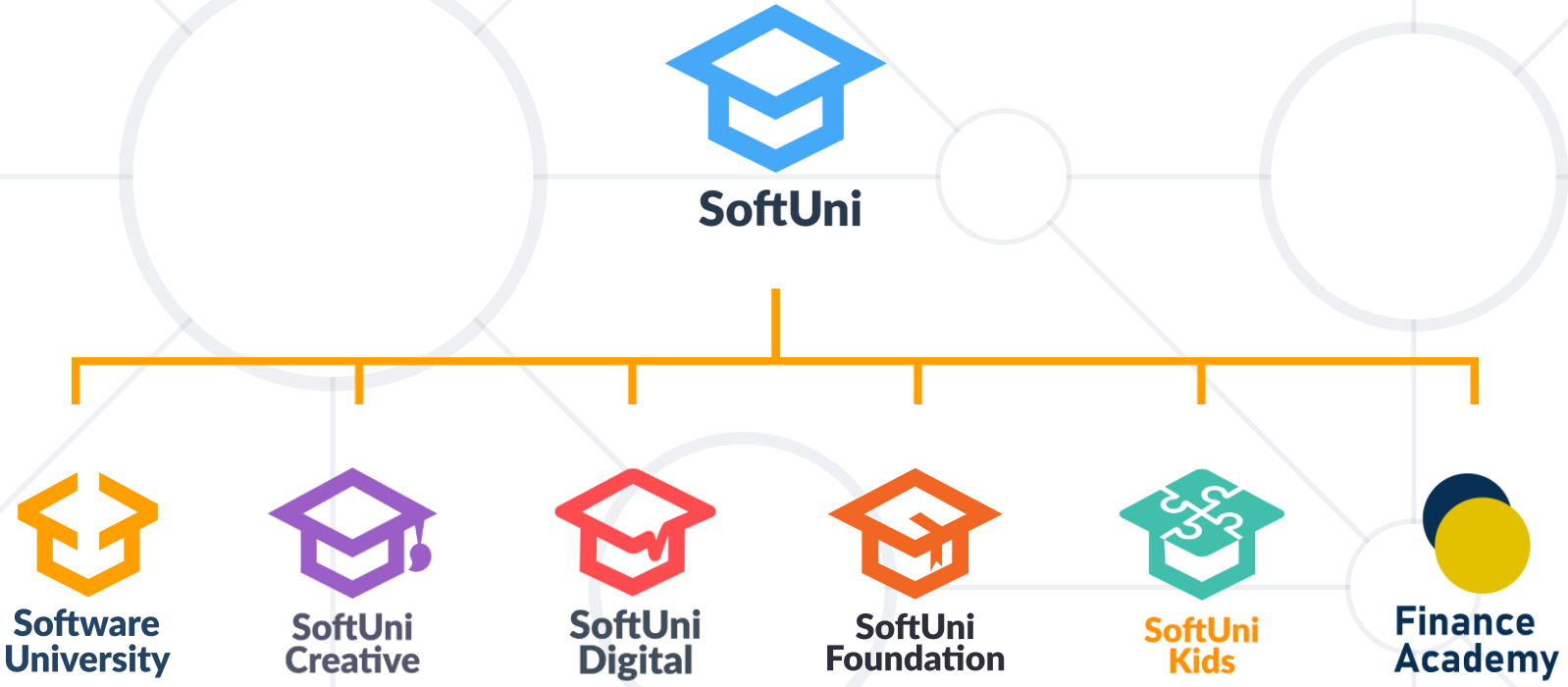
```
double sum = 0;

DirectoryInfo dir = new DirectoryInfo("TestFolder");
FileInfo[] infos = dir.GetFiles("*", SearchOption.AllDirectories);

foreach (FileInfo fileInfo in infos)
{
   sum += fileInfo.Length;
}

sum = sum / 1024 / 1024;

File.WriteAllText("output.txt", sum.ToString());
```

Gets all files from the given folder and its subfolders

Summary

- **Streams** are ordered sequences of bytes
  - Operations: **open** → **read** / **write** → **close**
  - Always close streams with **`try-finally`** or **`using(…)`**
- Use **`StreamReader`** / **`StreamWriter`** for text data
- Use **`FileStream`** to read / write binary files
- Use the **`File`** class to read / write files at once
- Use the **`Directory`** class to work with directories

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank
Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR
Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg