SDU

Lecture 3

Text & Binary Streams

Shark Book: Chapter 12, 13, 15

Agenda

1. Char Struct and String Class
   - *Methods of Char Struct*
   - *Methods of String Class*

2. Exceptions
   - *Mini Exercises (Exception Types)*

3. Text & Binary Streams
   - *Appending a Text File*
   - *Writing to a Binary File*
   - *Reading from a Binary File*

Objectives

1. Revise the pre-requisite concepts of
   - *Character and String Classes*
   - *Exceptions and Exception Handling*
   - *StreamReader & StreamWriter*
2. Build on the Revision
   - *BinaryReader & BinaryWriter*
   - *Appending to a File*

Methods in the Char Struct

- The majority of the methods in a char struct are static methods or non-instance methods
- The character methods reside in the `Char` struct
- `Char.MethodName(arguments)`

Method	Description
`IsDigit(ch)`	Returns true if the specified character is a digit.
`IsLetter(ch)`	Returns true if the specified character is a letter.
`IsLetterOrDigit(ch)`	Returns true if the specified character is a letter or digit.
`IsLower(ch)`	Returns true if the specified character is a lowercase letter.
`IsUpper(ch)`	Returns true if the specified character is an uppercase letter.
`ToLower(ch)`	Returns the lowercase of the specified character.
`ToUpper(ch)`	Returns the uppercase of the specified character.

Strings and Simple Methods for **String** Objects

- To represent a string of character, the data type `string` is used
- `string message = "Welcome to C#";`
- String provides both static and instance methods.
- `referenceVariable.MethodName(arguments).`

Method	Description
`Length`	Returns the number of characters in this string.
`Concat(s1, s2)`	Returns a new string that concatenates string `s1` with string `s2`. (Note: While `Concat` can be called with a variable number of arguments, it's often shown with two for clarity. It's a static method of the `string` class.)
`ToUpper()`	Returns a new string with all letters in uppercase.
`ToLower()`	Returns a new string with all letters in lowercase.
`Trim()`	Returns a new string with whitespace characters trimmed from both sides.

Comparing Strings

Method	Description
`Equals(s1)`	Returns true if this string is equal to string `s1`.
`CompareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or less than `s1`.
`StartsWith(prefix)`	Returns true if this string starts with the specified prefix.
`EndsWith(suffix)`	Returns true if this string ends with the specified suffix.

Comparing Strings

```
public class OrderTwoCities
{
    public static void Main(string[] args)
    {
        // Prompt user to enter two cities
        Console.Write("Enter the first city: ");
        string city1 = Console.ReadLine();
        Console.Write("Enter the second city: ");
        string city2 = Console.ReadLine();

        // Compare the cities using String.Compare() with ordinal comparison
        if (string.Compare(city1, city2, StringComparison.Ordinal) < 0)
        {
            Console.WriteLine("The cities in alphabetical order are " + city1 + " "+ city2);
        }
        else
        {
            Console.WriteLine("The cities in alphabetical order are " + city2 + " "+ city1);
        }
    }
}
```

```
Enter the first city: Odense
Enter the second city: Copenhagen
The cities in alphabetical order are Copenhagen Odense
```

Finding a Character or a Substring in a String

Method	Description
`IndexOf(char)`	Returns the zero-based index of the first occurrence of the specified character in the string. Returns `-1` if not found.
`IndexOf(char, int)`	Returns the zero-based index of the first occurrence of the specified character in the string, starting the search from the specified index. Returns `-1` if not found.
`IndexOf(string)`	Returns the zero-based index of the first occurrence of the specified substring in the string. Returns `-1` if not found.
`IndexOf(string, int)`	Returns the zero-based index of the first occurrence of the specified substring in the string, starting the search from the specified index. Returns `-1` if not found.
`LastIndexOf(char)`	Returns the zero-based index of the last occurrence of the specified character in the string. Returns `-1` if not found.
`LastIndexOf(char, int)`	Returns the zero-based index of the last occurrence of the specified character in the string, searching backward from the specified index. Returns `-1` if not found.
`LastIndexOf(string)`	Returns the zero-based index of the last occurrence of the specified substring in the string. Returns `-1` if not found.
`LastIndexOf(string, int)`	Returns the zero-based index of the last occurrence of the specified substring in the string, searching backward from the specified index. Returns `-1` if not found.

Exercise: Find Output

```
class Program
{
static void Main()
{
string sampleText = "hello world, welcome to C# programming world";

Console.WriteLine("IndexOf examples:");
Console.WriteLine("IndexOf('w') -> " + sampleText.IndexOf('w'));
Console.WriteLine("IndexOf('o', 5) -> " + sampleText.IndexOf('o', 5));
Console.WriteLine("IndexOf(\"world\") -> " + sampleText.IndexOf("world"));
Console.WriteLine("IndexOf(\"world\", 10) -> " + sampleText.IndexOf("world", 10));

Console.WriteLine("\nLastIndexOf examples:");
Console.WriteLine("LastIndexOf('w') -> " + sampleText.LastIndexOf('w'));
Console.WriteLine("LastIndexOf('o', 20) -> " + sampleText.LastIndexOf('o', 20));
Console.WriteLine("LastIndexOf(\"world\") -> " + sampleText.LastIndexOf("world"));
Console.WriteLine("LastIndexOf(\"world\", 30) -> " + sampleText.LastIndexOf("world", 30));
}
}
```

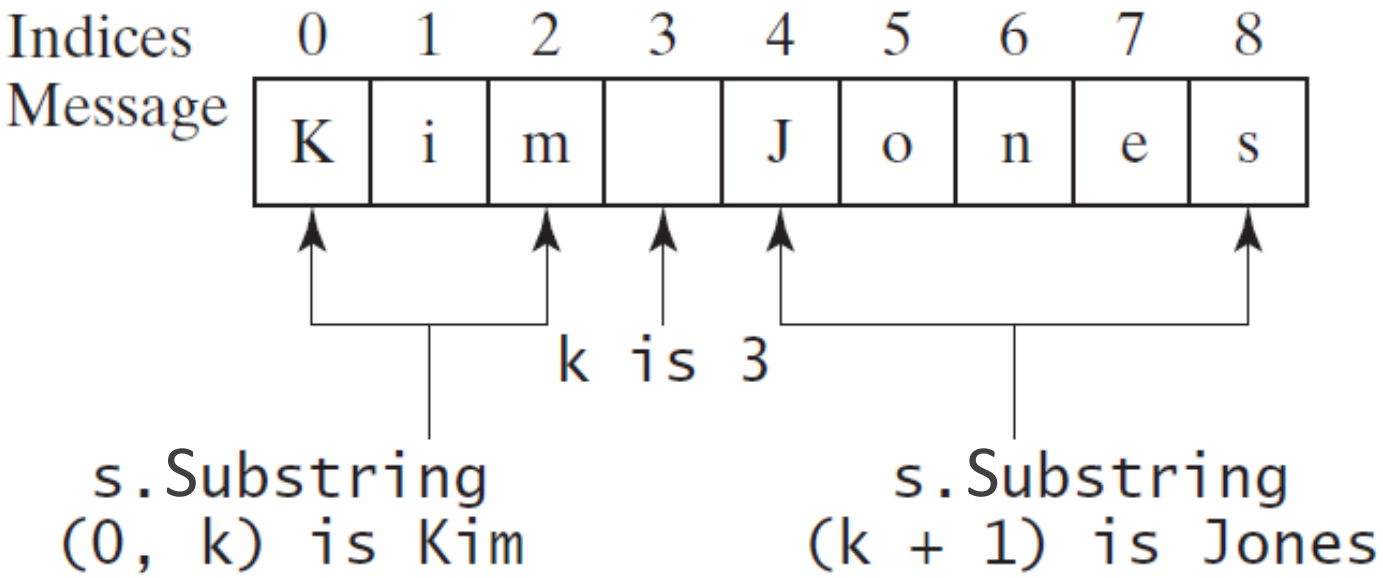
Finding a Character or a Substring in a String

start index

number of characters to extract

```
int k = s.IndexOf(' ');
String firstName = s.Substring(0, k);
String lastName =  s.Substring(k + 1);
```

What is special about Substring method???



Conversion between Strings and Numbers

Where do you need these conversions? Any idea????

- We can convert a numeric string into a number using

```
int intValue = int.Parse(intString);
double doubleValue = double.Parse(doubleString);
```

- We can also convert numbers back to string using the string concatenation operator

```
String s = number + "";
```

- **Note** that if the string is not a numeric string, the conversion would cause a runtime error.

Mini Exercise

- Show two ways to concatenate the following two strings together to get the string "Hi, mom.":

```
string hi = "Hi, ";
string mom = "mom.";
```

- How long is the string returned by the following expression? What is the string?

```
"Was it a car or a cat I saw?".Substring(9, 3)
```

- What is the output of the following code

```
string x = "10";
int y = 20;
string z = x + y;
int p = y + int.Parse(x);
System.Console.WriteLine(z);
System.Console.WriteLine(p);
```

Exceptions

https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions/

System Exception Example

What will happen on running the following code????

```
public static void Main(string[] args)
{
    int result = CalculateFactorial(5);
    Console.WriteLine($"Factorial: {result}");
}

static int CalculateFactorial(int n)
{
    return n * CalculateFactorial(n - 1);
}
```


SDU

Group activity
Mini Exercises (1-4)

Mini Exercise 1

. **Question:** Is there anything wrong with this exception handler as written? Will this code compile?

```
try {

} catch (Exception e) {

} catch (ArithmeticException a) {

}
```

Mini Exercise 2

Question: What exception types can be caught by the following handler?

```
catch (Exception e) {

}
```

What is wrong with using this type of exception handler?

Mini Exercise 3: Write Output


```
public class Test: Exception { }

public class Example
{
    public static void Main(string args)
    {
        try
        {
            throw new Test();
        }
        catch (Test t)
        {
            Console.WriteLine("Got the Test Exception");
        }
        finally
        {
            Console.WriteLine("Inside finally block");
        }
    }
}
```

What is special about Test class?

Mini Exercise 4a: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try {
            int[] a = new int[10];
            a[11] = 9;
        }
        catch (                                    ) {
            Console.WriteLine("                    ");
        }
    }
}
```

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/exceptions#214-how-exceptions-are-handled

Mini Exercise 4b: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try
        {
            int num = int.Parse("XYZ");
            Console.WriteLine(num);
        }
        catch (                    e) {
            Console.WriteLine(                                    );
        }
    }
}
```

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/exceptions#214-how-exceptions-are-handled

Mini Exercise 4c: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try
        {
            string str = "beginnersbook";
            Console.WriteLine(str.Length);
            char c = str[0];
            c = str[40];
            Console.WriteLine(c);
        }
        catch ([redacted]) {
            Console.WriteLine("[redacted]");
        }
    }
}
```

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/exceptions#214-how-exceptions-are-handled

Mini Exercise 4d: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try
        {
            string str = null;
            Console.WriteLine(str.Length);
        }
        catch (                                    ) {
            Console.WriteLine(                                    );
        }
    }
}
```

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/exceptions#214-how-exceptions-are-handled

Mini Exercise 4e: Find Output


```
try
{
    string input = null;
    int result = int.Parse(input);
}
catch (ArgumentNullException ex)
{
    Console.WriteLine("Null argument exception caught.");
}
catch (FormatException ex)
{
    Console.WriteLine("Format exception caught.");
}
catch (Exception ex)
{
    Console.WriteLine($"General exception caught: {ex.Message}");
}
```

Find output:

- when input=null
- what if input were an **empty string ("")**, what will be the output?

Break (10 Minutes)

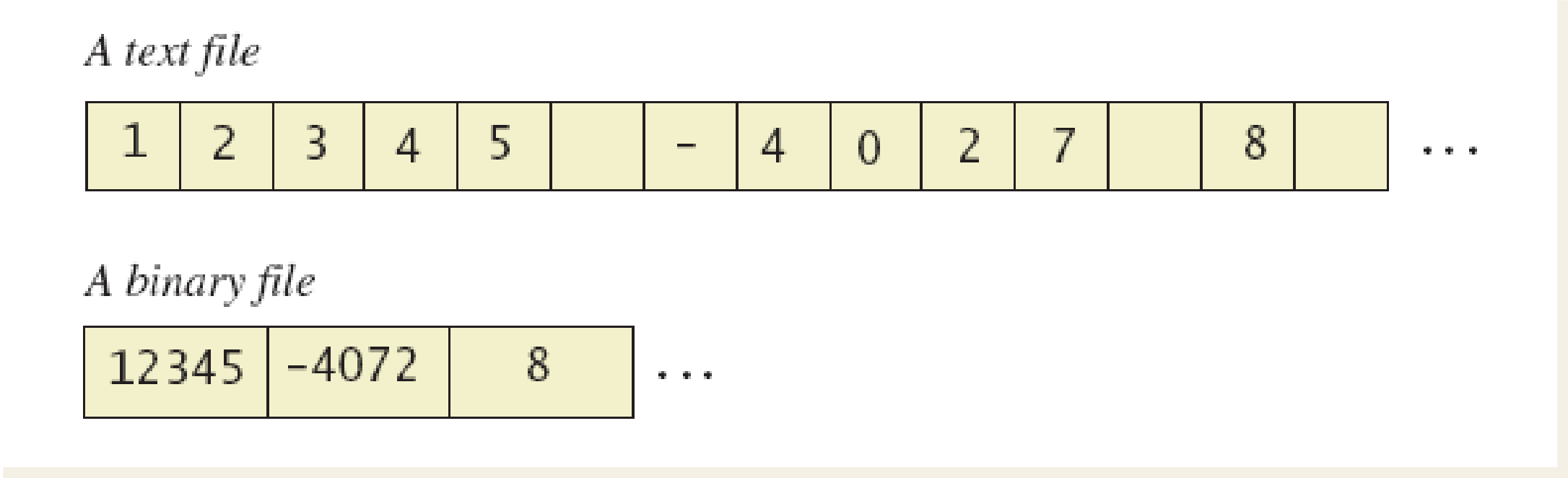
Text & Binary Streams

Files and File Types

There are two main types of files

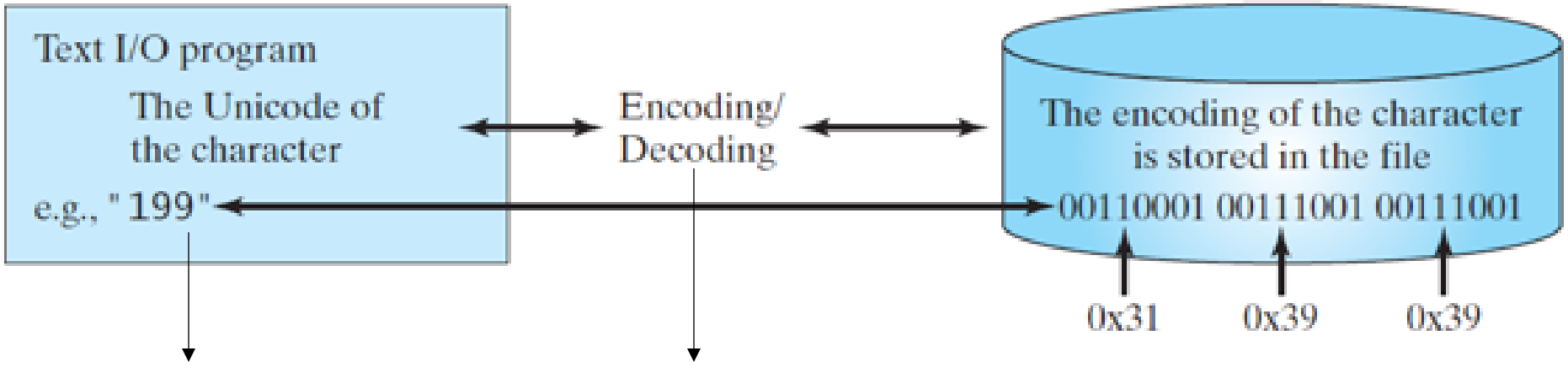
1. Text Files : These are sequence of characters that can be edited with text editors. E.g. c# source code

2. Binary Files : These are sequence of bytes that can be edited by specialized programs E.g. movies, music files



Text I/O

- Text I/O requires encoding and decoding.
- Convert a Unicode to a file specific encoding when writing a character and,
- Convert a file specific encoding to a Unicode when reading a character.

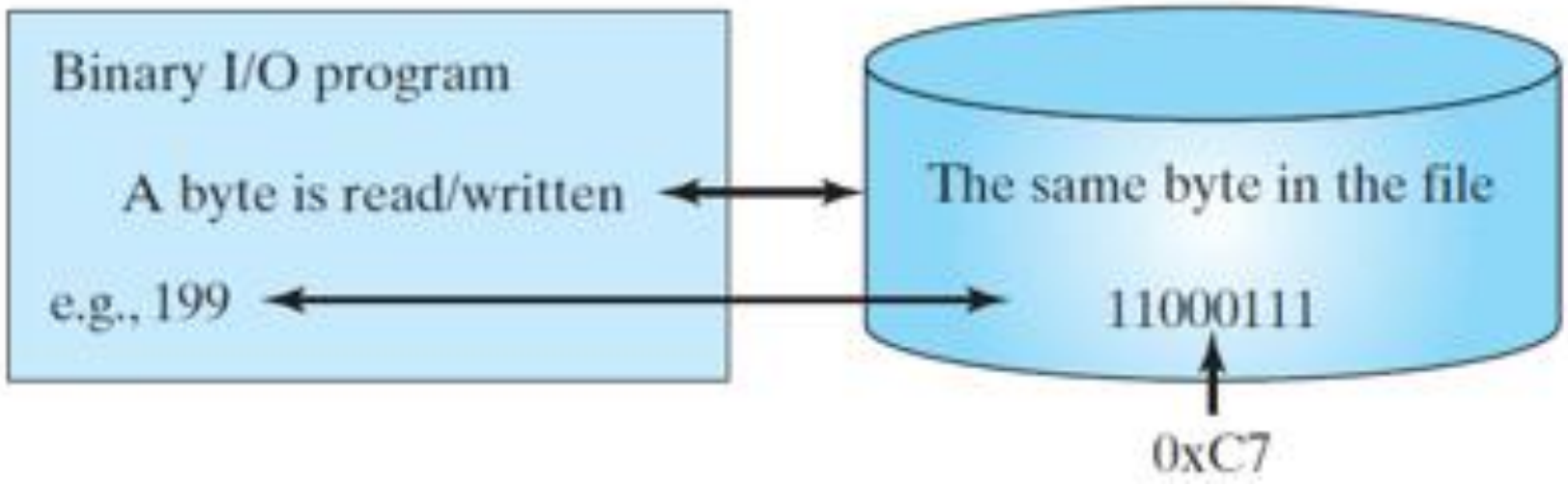


A standard that defines the characters and their code points.

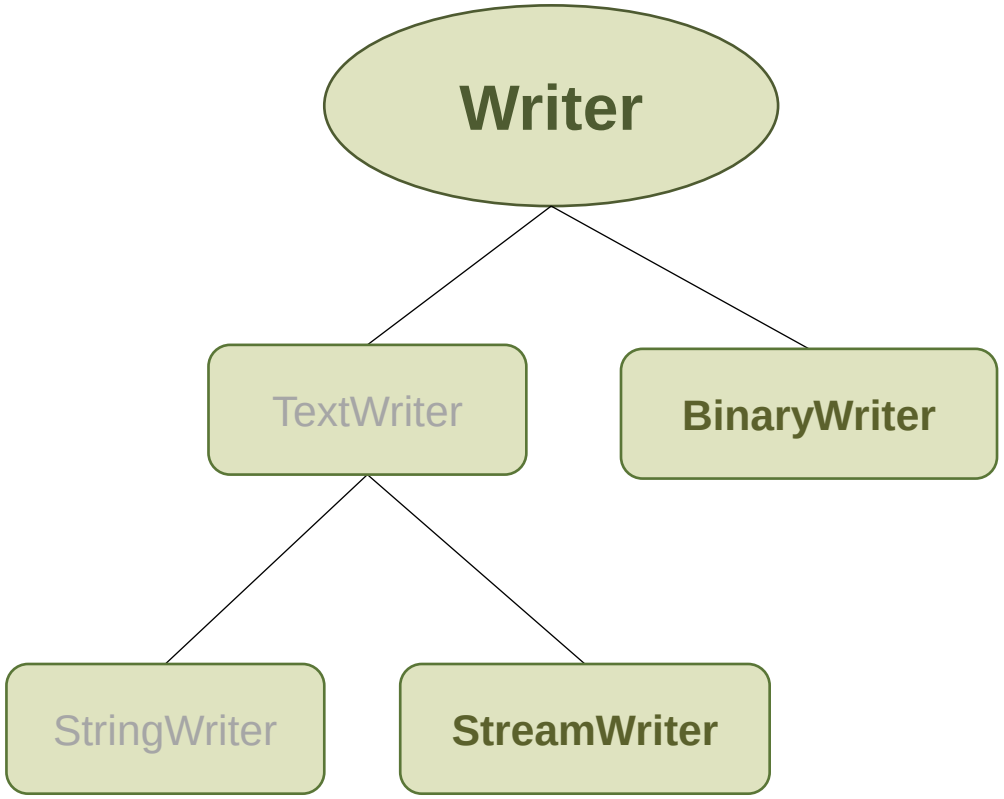
A way to represent Unicode characters as binary data or vice versa, i.e., UTF-8

Binary I/O

- Binary I/O does not require encoding and decoding.
- When you write a byte to a file, the original byte is copied into the file.
- When you read a byte from a file, the exact byte in the file is returned.



Writer Classes



Appending to Text Files: StreamWriter

- **StreamWriter** class has an appropriate constructor for appending to an existing file
- So using the command **StreamWriter sw = new StreamWriter("out.txt");**
- Will create and overwrite an empty file to **"out.txt"**
- We can use the **StreamWriter** class to append to an existing file **"out.txt"** as follows:

StreamWriter sw = new StreamWriter("out.txt", true);

- The **true** argument in the **StreamWriter** indicates that we want to append to the file.

SDU

Appending to Text Files: Example

```
public class AppendTextExample
{
    public static void Main()
    {
        string filePath = "example.txt"; // File path
        string textToAppend = "This is a new line appended to the file.";

        // Using StreamWriter in append mode (true for append)
        using (StreamWriter writer = new StreamWriter(filePath, true))
        {
            writer.WriteLine(textToAppend);
            Console.WriteLine("Text has been appended to the file.");
        }
    }
}
```

the using statement

Resource declaration and acquisition

```
// The stream is automatically closed when the using block ends.
using (StreamWriter sw = new StreamWriter("MyFile2.txt"))
{
    sw.Write("Hello, StreamWriter");
}
```

Resource usage

Resource disposal

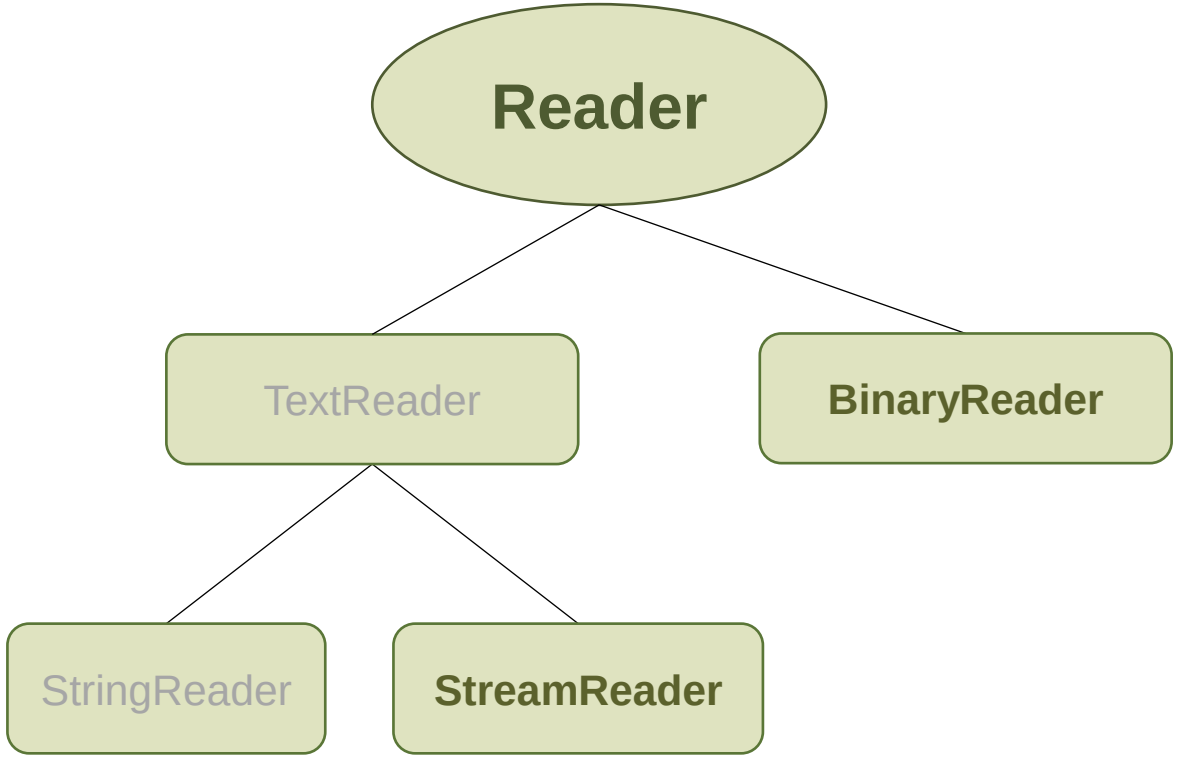
Shark book: page 437 and 438

BinaryWriter

- The BinaryWriter class is used to write binary data to a stream.
- Writes primitive types in binary to a stream and supports writing strings in a specific encoding.
- A BinaryWriter object is created by passing a FileStream object to its constructor.

```
// This code includes exception handling.
using (FileStream fs = new FileStream("data.bin", FileMode.Create))
using (BinaryWriter writer = new BinaryWriter(fs))
{
    writer.Write(123);
    writer.Write(45.67);
    writer.Write(true);
    writer.Write("I love C#");
}
```

Reader Classes



Comma Separated Files: StreamReader

Name of the csv file

```
File: Transactions.txt
SKU,Quantity,Price,Description
4039,50,0.99,SODA
9100,5,9.50,T-SHIRT
1949,30,110.00,JAVA PROGRAMMING TEXTBOOK
5199,25,1.50,COOKIE
```

File Header

Data

TO DO: Compute the total sales

SDU

Comma Separated Files

```
  string fileName = "Transactions.txt";
  using (StreamReader inputStream = new StreamReader(fileName))
{
  string line = inputStream.ReadLine();
  double total = 0;

while (!inputStream.EndOfStream)
{
  line = inputStream.ReadLine();
  string[] ary = line.Split(',');

  string SKU = ary[0];
  int quantity = int.Parse(ary[1]);
  double price = double.Parse(ary[2]);
  string description = ary[3];

  Console.WriteLine($"Sold {quantity} of {description} (SKU: {SKU}) at ${price:F2} each");
  total += quantity * price;
}
}
Console.WriteLine($"Total sales: ${total:F2}");
```

Open the input stream.

Read the header

Read lines in the file

A `string` implements a split method by a delimiter e.g. ‘, ‘ and it return a string array

We can parse the individual elements of the array

We can perform an arithmetic operation on the parsed elements.

BinaryReader

- The BinaryReader class is used to read binary data from a file.
- A BinaryReader object is created by passing a FileStream object to its constructor.

```
// This code includes exception handling (not shown).
using (FileStream fs = new FileStream("data.bin", FileMode.Open))
using (BinaryReader reader = new BinaryReader(fs))
{
    int intValue = reader.ReadInt32();
    double doubleValue = reader.ReadDouble();
    bool boolValue = reader.ReadBoolean();
    string stringValue = reader.ReadString();

    Console.WriteLine($"Integer: {intValue}");
    Console.WriteLine($"Double: {doubleValue}");
    Console.WriteLine($"Boolean: {boolValue}");
    Console.WriteLine($"String: {stringValue}");
}
```

Type inference: var keyword

```
using (var writer = new StreamWriter("output.txt"))
{
    writer.WriteLine("Hello, World!");
    writer.WriteLine("This is a simple StreamWriter example.");
} // StreamWriter is automatically closed here
```

Summary

- StreamReader and StreamWriter are designed for reading and writing text.
- They work with character-based data (strings, text files) and use an encoding (e.g., UTF-8 by default).
- StreamReader/StreamWriter converts text to bytes and vice versa, which introduces overhead compared to binary operations.
- BinaryReader and BinaryWriter are designed for reading and writing raw binary data.
- They are efficient for Reading and writing primitive data types (e.g., integers, floats, booleans).
- No encoding is involved, which makes these classes ideal for working with non-text data.

MCQs Quiz

Go to Plans -> VOP-3 -> VOP-3 (Lecture) -> Lecture-3 Test

Good Luck ☺