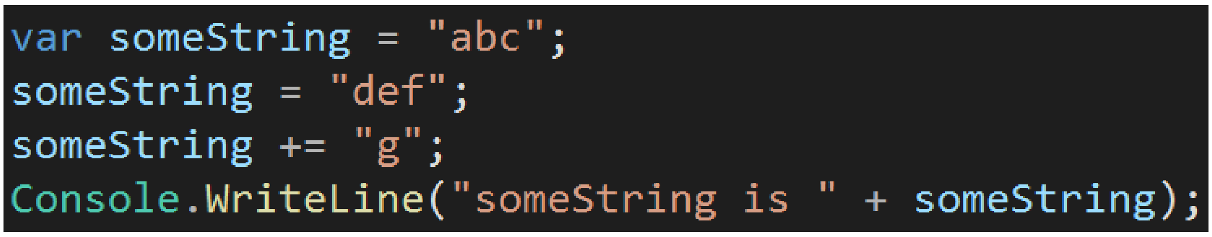
33. What is the difference between string and StringBuilder?

Brief summary: String is a type used for representing textual data. StringBuilder is a utility class created for optimal concatenation of strings.

String is a type used for representing textual data. StringBuilder is a utility class created for optimal concatenation of strings.

We all know strings. We use them all the time to represent some text. What some people don't know is that all strings are immutable, which means once created they can't be modified.

You may now be surprised. You probably mutated strings plenty of times by now. For example, this code is perfectly valid:



```
var someString = "abc";
someString = "def";
someString += "g";
Console.WriteLine("someString is " + someString);
```

Ta-dah. We modified a string. First, we changed the value from "abc" to "def" and then we added "g" to it, which resulted in the final value of "defg". So what's the fuss about the immutability?

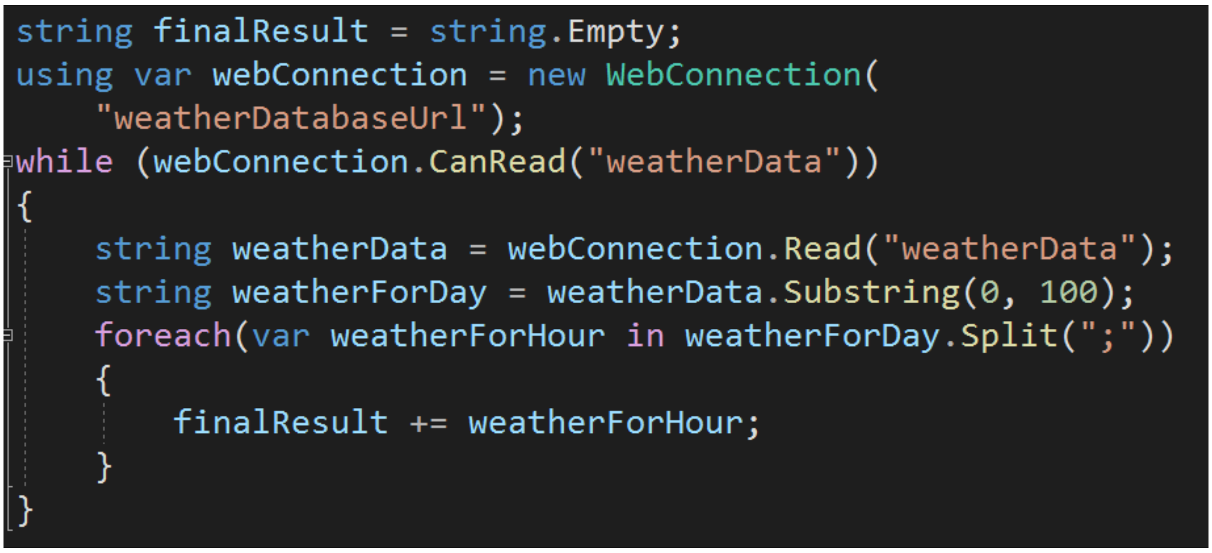
Well, we actually didn't modify the "abc" string. We created a brand new "def" string and we simply pointed the reference stored in **someString** to this new string. A similar thing happened in the next line. We created a new string by concatenating "def" with "g" and then pointed the reference to this new string. At some point in time, the Garbage Collector will see those old strings as objects to whom no reference points, and will remove them from memory.

All right. So we now know that when we "modify" a value of string the following things need to happen:

- a new object needs to be created, which involves allocating memory for it
- the variable that was pointing to the original string must be pointed to the new string

- at some point, the Garbage Collector must clean out the old string

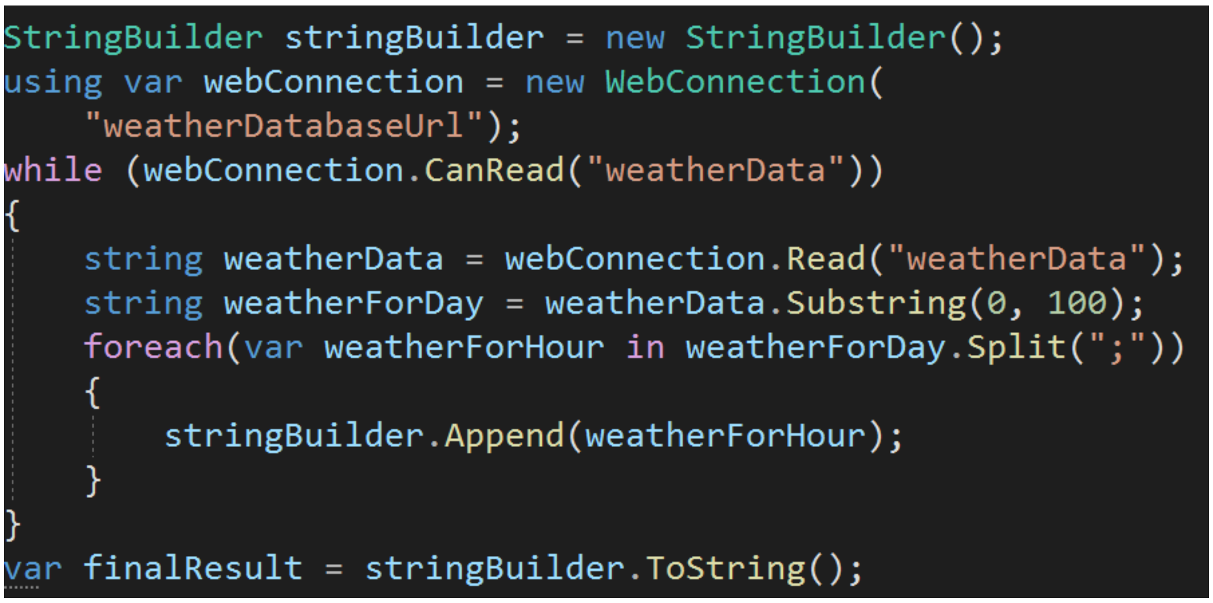
That’s relatively a lot of work. In many cases it’s ok and we don’t notice any performance impact when we add “Mr. ” to the “John Smith” string. But a need to build strings gradually from parts is pretty common. For example, imagine your application is downloading some data from the web, and it does it in chunks, as the data is pretty large. You need to create some kind of extract from this data.



```
string finalResult = string.Empty;
using var webConnection = new WebConnection(
    "weatherDatabaseUrl");
while (webConnection.CanRead("weatherData"))
{
    string weatherData = webConnection.Read("weatherData");
    string weatherForDay = weatherData.Substring(0, 100);
    foreach(var weatherForHour in weatherForDay.Split(";"))
    {
        finalResult += weatherForHour;
    }
}
```

For each chunk read (and it can be thousands of them) you need to build some pretty complex string, and then append this string to the string representing the final result. It can involve millions of concatenation operations. As the process continues the final result is growing. At some point, it can be a huge string, and still, every concatenation keeps copying it to a new (huge!) part of memory, adding some tiny part, and then replacing the reference stored in the original variable. Not to mention that behind the scenes the Garbage Collector is struggling to clean up all those large, but unused strings from memory.

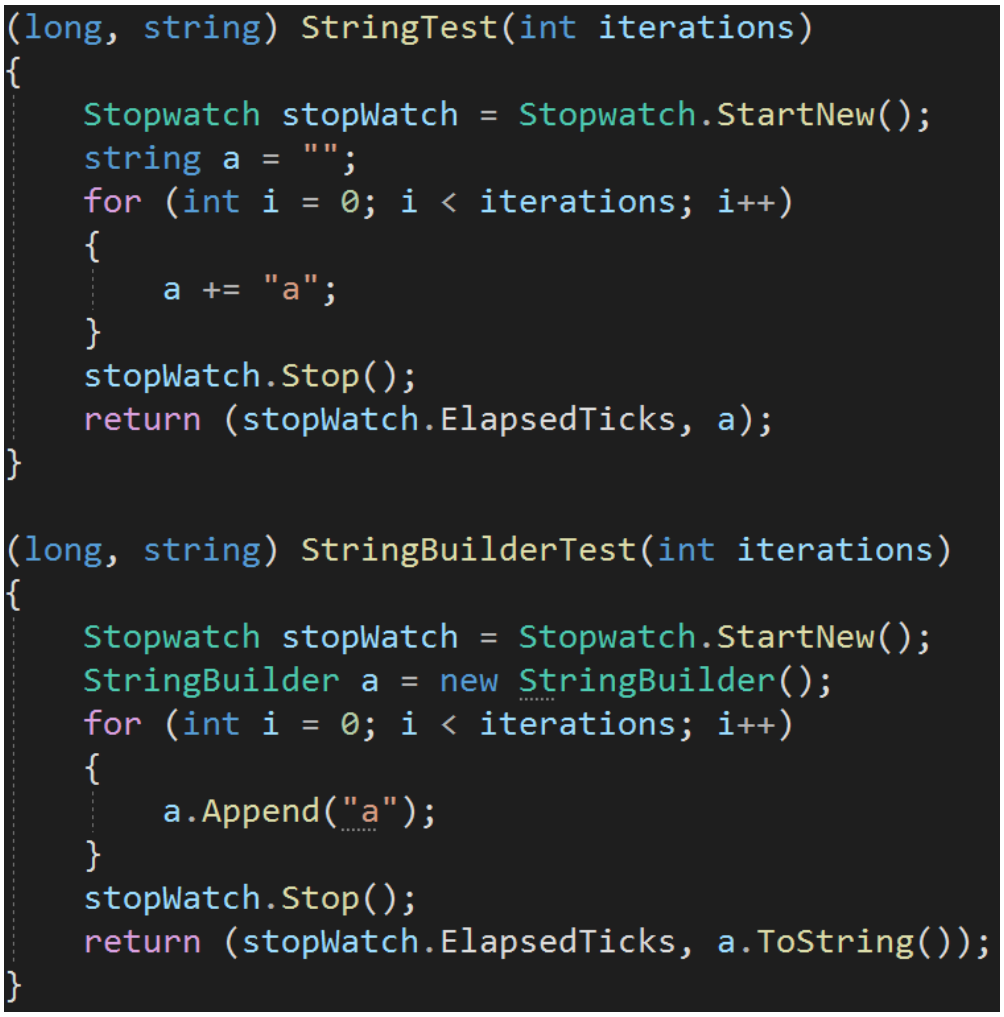
And for such use cases, the StringBuilder class has been created.



```csharp
StringBuilder stringBuilder = new StringBuilder();
using var webConnection = new WebConnection(
    "weatherDatabaseUrl");
while (webConnection.CanRead("weatherData"))
{
    string weatherData = webConnection.Read("weatherData");
    string weatherForDay = weatherData.Substring(0, 100);
    foreach(var weatherForHour in weatherForDay.Split(";"))
    {
        stringBuilder.Append(weatherForHour);
    }
}
var finalResult = stringBuilder.ToString();
```

The StringBuilder can add or remove pieces from the final result, but without this laborious copying of the old string, adding a part, and removing the old string. StringBuilder object maintains a buffer to accommodate expansions to the string. New data is appended to the buffer if there is any space in it left. Otherwise, a new, larger buffer is allocated, data from the original buffer is copied to the new buffer, and the new data is then appended to the new buffer. As you can see the only scenario when the entire result is being copied is when the buffer needs to be enlarged.

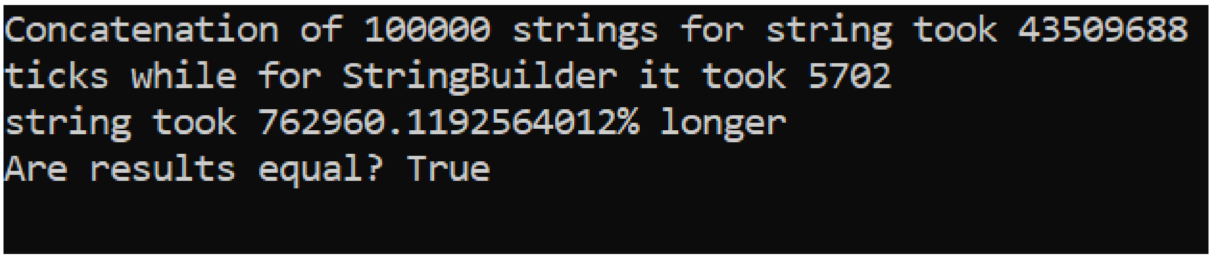
All right. Let's see a simple program that will measure the performance of string and StringBuilder:



```
(long, string) StringTest(int iterations)
{
    Stopwatch stopWatch = Stopwatch.StartNew();
    string a = "";
    for (int i = 0; i < iterations; i++)
    {
        a += "a";
    }
    stopWatch.Stop();
    return (stopWatch.ElapsedTicks, a);
}

(long, string) StringBuilderTest(int iterations)
{
    Stopwatch stopWatch = Stopwatch.StartNew();
    StringBuilder a = new StringBuilder();
    for (int i = 0; i < iterations; i++)
    {
        a.Append("a");
    }
    stopWatch.Stop();
    return (stopWatch.ElapsedTicks, a.ToString());
}
```

As you can see both methods simply build a string of letters "a" of the length given in the parameter. Let's see how they will handle 100000 iterations:

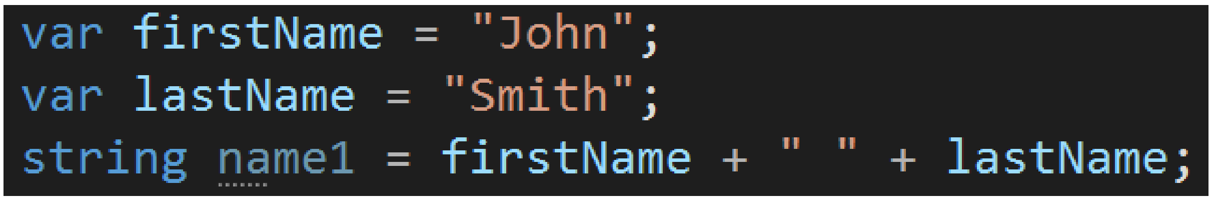


```
Concatenation of 100000 strings for string took 43509688
ticks while for StringBuilder it took 5702
string took 762960.1192564012% longer
Are results equal? True
```

Wow. StringBuilder built the result string over 7000 times faster than string.

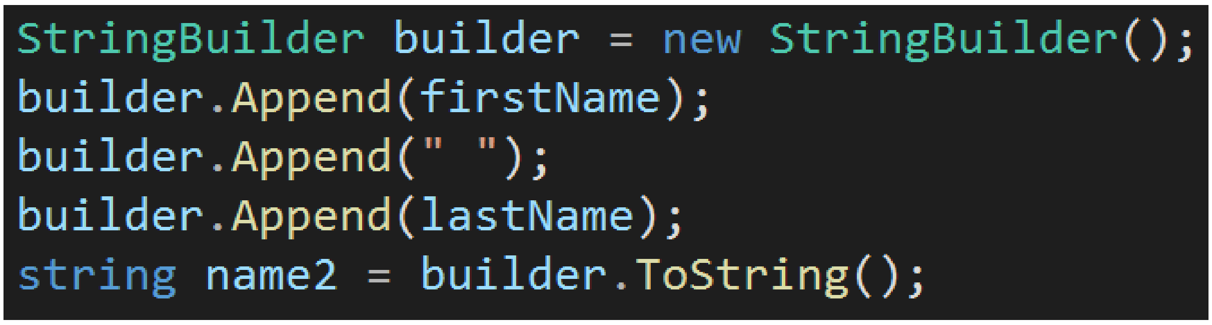
I hope I convinced you that when you implement some process of incremental building of strings, the StringBuilder should be your choice.

Of course, for simple uses like concatenating a couple of strings, using the StringBuilder is an overkill and it only complicates the code. And if you wanted to have at least a tiny performance boost, I must disappoint you. If the string is not built incrementally but is composed in a single instruction, plain old string addition actually works faster:



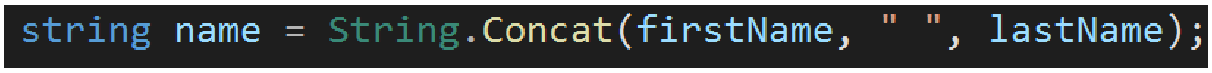
```
var firstName = "John";
var lastName = "Smith";
string name1 = firstName + " " + lastName;
```

The above is faster than the below, not to mention how much simpler it looks:



```
StringBuilder builder = new StringBuilder();
builder.Append(firstName);
builder.Append(" ");
builder.Append(lastName);
string name2 = builder.ToString();
```

The performance of the first code is better because behind the scenes this code is translated into:



```
string name = String.Concat(firstName, " ", lastName);
```

Which is actually quite efficient. Remember, this can only be done if concatenation happens in a single instruction, so it would not work if, for example, we used a loop to build a string from pieces.

All right. The most important thing you need to remember from this lecture is to use StringBuilder when incrementally building large strings, as it gives much better performance than using a simple string.

Bonus questions:

- "**What does it mean that strings are immutable?**"
  It means once a string is created, it can't be modified. When we modify a string, actually a brand-new string is created and the variable that stored it simply has a new reference to this new object.