

Data Structures and Algorithms Using Java
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 58
String Buffer Class

So, String as it is an important concept. So string, Java developer has created this string concept in a more detailed way and therefore, many more other class also have been defined. So, we have studied one class called the string to process strings, there is another one class the StringBuffer which has been added in a recent JDK and further there is another also class dealing with string called a string builder.

(Refer Slide Time: 01:03)



So, in today's discussion includes the different ways the string can be manipulated other than the class string. So, basically we will discuss about the new class which is called a StringBuffer class and then few illustration with the StringBuffer class, then I will give an idea about string builder class and then finally, I will conclude this discussion giving a comparison among string, StringBuffer and StringBuilder class.

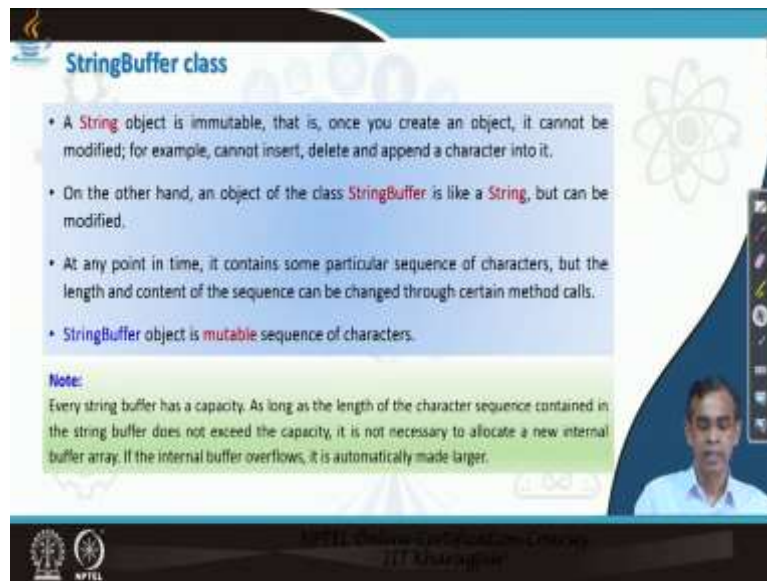
(Refer Slide Time: 01:34)



So, let us first have out the concept fast about the StringBuffer. Now, I told you in our previous discussion that string is immutable that means once a string object is created, it cannot grow further that is why neither you cannot remove one character or you cannot add another character into it.

So that is why the string is basically a constant in that sense whereas, a StringBuffer is basically to address this limitation. Now, this means that if you declare a string with StringBuffer, you will be able to enjoy all benefits that is available part the string as well as some extra flexibility that you can enjoy with this class.

(Refer Slide Time: 02:26)



StringBuffer class

- A **String** object is immutable, that is, once you create an object, it cannot be modified; for example, cannot insert, delete and append a character into it.
- On the other hand, an object of the class **StringBuffer** is like a **String**, but can be modified.
- At any point in time, it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.
- **StringBuffer** object is **mutable** sequence of characters.

Note:
Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger.

Now, let us see what are the different ways the StringBuffer can support to the programmer is better than the string class. So, a string is immutable this means that once you create an object it cannot be modified you cannot change it. So, you cannot insert any character into it or you cannot delete a character from it even you cannot add a character into it append whatever it is there.

On the other hand, the StringBuffer is like a string, but it can be modified that means it is mutable. So, we can say the StringBuffer object is a mutable sequence of characters and at any point while you are processing a string, it can contains, a string can contains some particular sequence of characters, but the length and the content of the sequence can be changed by certain call up methods of the StringBuffer class.

So, this means that if the string is a static one object, then the StringBuffer can give you dynamic objects which can grow or it can sync as per the requirement. So everything, every StringBuffer object actually of type any string of type StringBuffer in fact can have one buffer of it that is why it is name and that buffer is basically having certain capacity which the default capacity as the time when you initialize with the length of the string that you have initialized.

So, with that it is a default capacity, otherwise it will go. So, if you add one string with another string. For example, a string s which say Java and another string say and you want to add it into C plus-plus then s equals to s plus C plus-plus. So, initially it has the length say of the same as Java's length but later on it can expand and then it can accommodate another string C plus-plus, so this way they are immutable.

So, internally that data can be allocated. So, there will be no concept of overflow conservatives there. Whatever be the last string you want to add it will do it comfortably.

(Refer Slide Time: 04:51)



The slide displays a table with two columns: 'Constructor' and 'Description'. The table lists four different constructors for the StringBuffer class, each with its signature and a brief description of its function. The table is highlighted with a light green border.

Constructor	Description
<code>StringBuffer()</code>	Constructs a string buffer with no characters in it and an initial capacity of 16 characters.
<code>StringBuffer(char[] seq)</code>	Constructs a string buffer that contains the same characters as the specified CharSequence.
<code>StringBuffer(int capacity)</code>	Constructs a string buffer with no characters in it and the specified initial capacity.
<code>StringBuffer(String str)</code>	Constructs a string buffer initialized for the contents of the specified string.

Now, let us see how we can create object of type StringBuffer class. So, like String class, there are many constructors are defined for this method also. So, here I have listed this is the default constructors. So, it is basically with no characters in it and its initial capacity is 16. So, this is the default capacity of the string by which it will create and you can create a string object by giving a sequence of characters of type this one, so it is basically the string object will be instantiated with input as an argument here.

Rather you can create a StringBuffer object specifying the initial capacity of dusting. So, which can be anything other than 16 or any value it is there. Now here also you can create a

StringBuffer object with passing as an input to the stream. So, there is basically the idea is that a string is immutable, how you can change it. So, best idea is that we can create a StringBuffer object for that string do whatever it is there, this way a immutable string, if it is there, we can make it changeable by means of StringBuffer class.

So, this is basically the another constructor which takes an input as a string object and then it basically create a StringBuffer objects.

(Refer Slide Time: 06:28)

```
import java.lang.*;
// It is not necessary as it is by default imported in all programs.

class StringBufferDemo {
    public static void main(String args[] ) {
        StringBuffer strObj1 = "123456789";
        StringBuffer strObj2 = "0000000000";
        StringBuffer strObj3 = strObj1 + " and " + strObj2;
        System.out.println(strObj1);
        System.out.println(strObj2);
        System.out.println(strObj3);

        StringBuffer sb1 = new StringBuffer(16);
        StringBuffer sb2 = new StringBuffer("abc");
        StringBuffer sb3 = new StringBuffer("abc" + " " + sb2);
    }
}
```

Now, let us illustrate with an example to understand how we can create the StringBuffer objects. So, this is a simple program. So, we are giving a demonstration of StringBuffer demo. So, here we create as we see StringBuffer, string object 1 is a StringBuffer type. So, there are three objects and we can print the same thing as we have already done for the string object.

So, it is doing and here also you see this is the concatenation that means everything is very similar to the string, but it is basically with dynamically that it can grow. Now, here we just we see, we create a StringBuffer object with a initial capacity say 16th we create another string object passing string this one is the initial string that we can pass it to this.

Another string object is created and here we see, we call this constructor passing a string, this is basically our usual string and concatenate with a StringBuffer object. So, that is also possible. That means string object and StringBuffer object can be used inter-mixingly, no issue and a string it is there.

Now, we can extend these things with an example that you can check it, here a string the object that you have created and you can see it is possible a string this is equals to again a string plus maybe a, b, c that is also possible, that means for the, if a string this is a string object, if it is a StringBuffer object, then it is possible, but if it is a string object, then it is not possible. So, you can understand about the difference between StringBuffer and string object. So it is mutable actually, that is how you are able to perform this operation.

(Refer Slide Time: 08:36)

The top screenshot shows a slide titled "Methods of StringBuffer" with a coffee cup icon. The bottom screenshot shows a table titled "Methods of StringBuffer class" with the following content:

Method	Description
<code>append(boolean b)</code>	Appends the string representation of the boolean argument to this sequence.
<code>append(char c)</code>	Appends the string representation of the char argument to this sequence.
<code>append(char[] str)</code>	Appends the string representation of the char array argument to this sequence.
<code>append(char[] str, int start, int end)</code>	Appends the string representation of a subarray of the char array argument to this sequence.
<code>append(CharSequence cs)</code>	Appends the specified CharSequence to this sequence.
<code>append(CharSequence cs, int start, int end)</code>	Appends a subsequence of the specified CharSequence to this sequence.
<code>append(double d)</code>	Appends the string representation of the double argument to this sequence.

Now, let us come to the discussion of methods of StringBuffer, what are the different methods that we can think about like string class there are many methods all methods those are there in string class also implemented in StringBuffer class. Further in addition to those methods in string class that we have discussed it has its own methods.

Now, there are few methods which are basically new in this is basically append and its argument can be any type as we can see, so append can be, so a string object can be appended with any string or any other data type, any other primitive data type as we see here. So, if we call this append method for a StringBuffer object and then we can append to this string with any primitive data type, automatically converting to a string type.

So, these are the statement that you can check it and you can run it and then small program you can write calling this method for the StringBuffer object and then you can understand how they are work, they work for you.

(Refer Slide Time: 09:44)

Method	Description
<code>append(Object o)</code>	Appends the string representation of the first argument to this sequence.
<code>append(int i)</code>	Appends the string representation of the int argument to this sequence.
<code>append(Long l)</code>	Appends the string representation of the long argument to this sequence.
<code>append(Object obj)</code>	Appends the string representation of the Object argument.
<code>append(String str)</code>	Appends the specified string to this character sequence.
<code>append(StringBuffer sb)</code>	Appends the specified StringBuffer to this sequence.
<code>appendCodePoint(int codePoint)</code>	Appends the string representation of the codePoint argument to this sequence.
<code>capacity()</code>	Returns the current capacity.
<code>trimToSize()</code>	Attempts to reduce storage used for the character sequence.

Now, there are a few more methods actually as I told you append method can be called for many type of this one. So, here also append and a string that means a string object can be appended to a StringBuffer object. StringBuffer object also can be appended to another StringBuffer object and it basically from any point of the part of the string can be appended to any other point.

So, here append a string representation of the, this as an argument that basically the integer value and then capacity is basically if you want to know what is the current capacity of the string. So, capacity and length are the two things, capacity means what is the maximum size of the buffer, that the string can currently hold and the length is basically how many characters at present the string is having.

So, that can be obtained and trimToSize, whatever the capacity it is there, it basically reduced to the total size of the ())(10:42). So, this method can be used for minimizing the space also.

So, these are the few methods that we have discussed about there are many more methods actually.

(Refer Slide Time: 10:54)

Method	Description
<code>charAt(int index)</code>	Returns the char value of this sequence at the specified index.
<code>codePointAt(int index)</code>	Returns the character (Unicode code point) at the specified index.
<code>codePointBefore(int index)</code>	Returns the character (Unicode code point) before the specified index.
<code>codePointCount(int beginIndex, int endIndex)</code>	Returns the number of Unicode code points in the specified text range of this sequence.
<code>delete(int start, int end)</code>	Removes the characters in a substring of this sequence.
<code>deleteCharAt(int index)</code>	Removes the char at the specified position in this sequence.
<code>ensureCapacity(int minimumCapacity)</code>	Ensures that the capacity is at least equal to the specified minimum.
<code>getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)</code>	Characters are copied from this sequence into the destination character array dst.
<code>toString()</code>	Returns a string representing the data in this sequence.

And few other methods that can be of interesting here I have mentioned character at, so this is regarding indexing. So, CodePointAt it basically returns that Unicode value for a particular string actually and starting from a given index. So this basically return the Unicode starting from a given index of all the characters, those are the Unicode of all those characters.

So, now, delete is another method, here basically you can remove a portion of a string objects I mean StringBuffer object starting from start and end both inclusive. Now delete character at is a particular array character at a particular index position also can be removed. ensureCapacity, if you want to increase the capacity then you can use this method and get characters is basically gives the all characters starting from a particular point index to another index.

So, this basically get all characters in between the two, it is just like a substring calculation and this is a toString as you have known that any object can be converted to the string. So, this is the one method by which you can convert to string, automatically a string is represented in the form of a string, but it can be called for any other type of data.

(Refer Slide Time: 12:16)

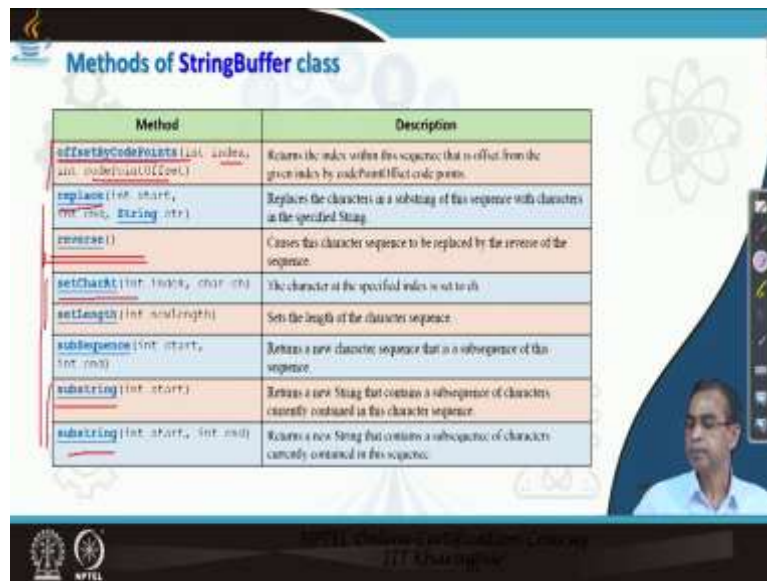
Method	Description
<code>indexOf(String str)</code>	Returns the index within this string of the first occurrence of the specified substring.
<code>indexOf(String str, int fromIndex)</code>	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
<code>insert(int offset, boolean b)</code>	Inserts the string representation of the boolean argument into this sequence.
<code>insert(int offset, char c)</code>	Inserts the string representation of the char argument into this sequence.
<code>insert(int offset, char[] str)</code>	Inserts the string representation of the char array argument into this sequence.
<code>insert(int from, char[] str, int offset, int len)</code>	Inserts the string representation of a subarray of the str array argument into this sequence.
<code>insert(int offset, CharSequence cs)</code>	Inserts the specified CharSequence into this sequence.
<code>insert(int offset, CharSequence cs, int start, int end)</code>	Inserts a subsequence of the specified CharSequence into this sequence.

Now, here is a few more method, this is the index off again. So, it basically to indicate for a given string, what is that different index, so return the index within the string of the first occurrence of the specified substring that means a substring is given and it is basically the return the index of occurrence of that substring in a string for which this method is called.

So, there are different form of this index also there are, these are another form and insert is basically if you want to insert a particular Boolean value into a particular point, then this method can be called. Now, like Boolean many other type of primitive data also can be insert in a particular position of the string.

So, it basically represents these are the different type of this one also and these are the many methods who is basically related to insert by which the different way, the insertion operation to a StringBuffer object can be carried out.

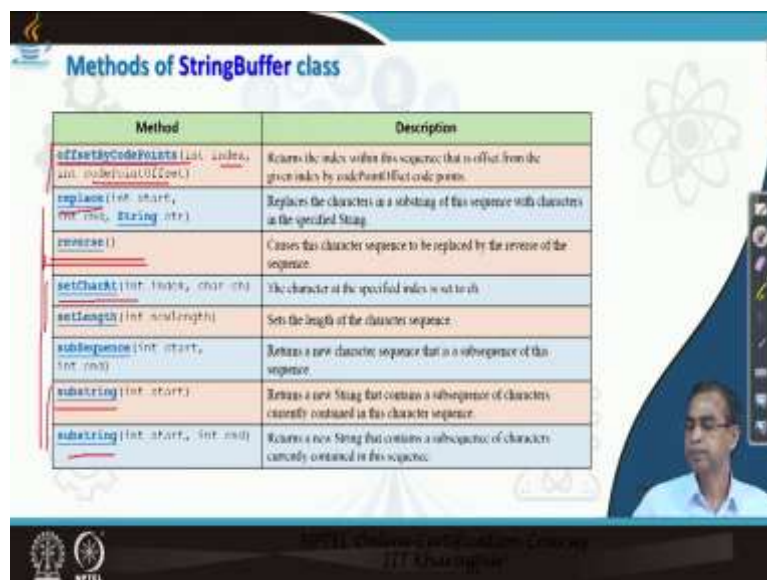
(Refer Slide Time: 13:26)



Method	Description
<code>offsetByCodePoints (int index, int codePointOffset)</code>	Returns the index within this sequence that is offset from the given index by codePointOffset code points.
<code>replace (int start, int end, String str)</code>	Replaces the characters in a substring of this sequence with characters in the specified String.
<code>reverse ()</code>	Causes this character sequence to be replaced by the reverse of the sequence.
<code>setCharAt (int index, char ch)</code>	The character at the specified index is set to ch.
<code>setLength (int newLength)</code>	Sets the length of the character sequence.
<code>subSequence (int start, int end)</code>	Returns a new character sequence that is a subsequence of this sequence.
<code>substring (int start)</code>	Returns a new String that contains a subsequence of characters currently contained in this character sequence.
<code>substring (int start, int end)</code>	Returns a new String that contains a subsequence of characters currently contained in this sequence.

Now, there are a few more methods again related to the Insert, there are various ways the insertion can be done. The last index method is same as the string object that we have already studied, it is for this StringBuffer class, their last index of string and form index also you can same way we have already illustrated these basically return the length of the string is not that capacity is basically total number of characters in the string that it is there. So that will return.

(Refer Slide Time: 13:59)



Method	Description
<code>offsetByCodePoints (int index, int codePointOffset)</code>	Returns the index within this sequence that is offset from the given index by codePointOffset code points.
<code>replace (int start, int end, String str)</code>	Replaces the characters in a substring of this sequence with characters in the specified String.
<code>reverse ()</code>	Causes this character sequence to be replaced by the reverse of the sequence.
<code>setCharAt (int index, char ch)</code>	The character at the specified index is set to ch.
<code>setLength (int newLength)</code>	Sets the length of the character sequence.
<code>subSequence (int start, int end)</code>	Returns a new character sequence that is a subsequence of this sequence.
<code>substring (int start)</code>	Returns a new String that contains a subsequence of characters currently contained in this character sequence.
<code>substring (int start, int end)</code>	Returns a new String that contains a subsequence of characters currently contained in this sequence.

Now, there are a few more methods there are plenty of methods as I told. So, this is also `offsetByCodePoint` it is basically Unicode point value starting from an index for a particular set of characters it can return. `Replace` is basically changing the sequence, `reverse` the usual method if you want to make the things into a reverse order. So, if you call the method for a

StringBuffer, it automatically reverse and it is a mutable way reverse, reverse method is not there for the class string that you can recall and set character at basically modifying, these are the methods of modifying the StringBuffer object and these are the method for finding the substring of a given StringBuffer objects.

(Refer Slide Time: 14:48)



So, there are many methods that is defined here, we have listed all methods, those are defined for the StringBuffer class. Now let us take some time to illustrate few important aspects of this illustration of the StringBuffer class. First we will see exactly how a StringBuffer object can be modified.

(Refer Slide Time: 15:02)



Modifying a string

- Because **String** objects are immutable, whenever you want to modify a **string**, you must either copy it into a **StringBuffer** or **StringBuilder**, or use a **String** method that constructs a new copy of the string with your modifications complete.
- On the other hand, you can modify **StringBuffer** object with the methods, for example, **append()**, **insert()**, **delete()**, **concat()**, **reverse()**, etc.

Now, this is one idea basically many programmer follows how a string object which is basically immutable can be modified here. So, this is a clue that I have given as the string objects are immutable whenever you want to modify a string, you can copy into a StringBuffer or StringBuilder we will discuss StringBuffer shortly and then using the method that is defined there, you can just simply modify.

So, this way a string which is immutable, can be somehow can be modified. And in the StringBuffer method in order to modify string there are many methods which are defined, which we have already studied and let us have some illustration of some methods towards the modification of StringBuffer objects.

(Refer Slide Time: 15:59)

Example 58.2: Modifying strings

```
public class ModifyStringDemo {
    public static void main(String args[]) {
        StringBuffer text = new StringBuffer("Data Structure ");
        text.append("C++");
        System.out.println(text);
        text.insert(15, "with ");
        System.out.println(text);
        text.replace(20, 23, "Java");
        System.out.println(text);
        text.delete(14, 19);
    }
}
```

Output

```
Data Structure C++
Data Structure with C++
Data Structure with Java
```

So, here is an example, this example this program to demo the modification of string objects and here we have created a string objects and these of type StringBuffer as you see. Now, this is the append method we can call, you can understand what it returns. So, as it is see this is output that it will give because it will append and then insert 15.

So, it basically takes .append now, it content this one and then insert 15 with so, it basically insert in this string at this position the another part of the string, which passed as an argument will be inserted as we see this basically gives the output. Now replace this one, so 20 to 23 possibly, this is basically 20 to 23. So, this will be replaced by this one.

So, it basically say that, this is the part of the substring which will be replaced between this portion by the new string, it is this one. So, the string will be expanded as required and then it basically modified this one. Delete 14 to 19, you can understand what is the output that it will give it, so it will basically remove the character position from 14 to 19 and it will print here. So that print you can check it is okay, if you give a system.out.println text then it basically give you after the modification, the output it is there.

(Refer Slide Time: 17:30)

Example 58.2: Modifying strings

```
public class ModifyingStringDemo {
    public static void main(String args[]) {
        StringBuffer text = new StringBuffer("Data Structure C++");
        text.append(" ");
        System.out.println(text);
        text.insert(1, "with ");
        System.out.println(text);
        text.replace(1, 10, "Java");
        System.out.println(text);
        text.delete(1, 10);
    }
}
```

Output

```
Data Structure C++
Data Structure with C++
Data Structure with Java
```

Anyway so you can understand then how if you call, if you create, for example in `(())`(17:34) of this as a simple string and you can call this method you see this will throw a compile time exception error you will not be able to do that, because it will check it there. Anyway, so these are the programs that you can check and then hope you have understood the difference between string and then StringBuffer there are a few more aspects there, which is interesting to learn about the string manipulation and using the StringBuffer class.

(Refer Slide Time: 18:01)

Example 58.3: The reversing a string

`reverse()` This is used to reverse the whole string.

```
public class StringReversingDemo {
    public static void main(String args[]) {
        StringBuffer text = new StringBuffer("Data Structure with Java");
        text.reverse();
        System.out.println(text);
    }
}
```

Output

```
avaI hW hW eracturS atA
```

So, this is a reverse method which is also not possible for that class string, but it is possible for the StringBuffer. So, here we create a StringBuffer object as it is there and then call the

reverse method then it basically print the string. So, you can see this basically modified by storing in the reverse order as we see this is the output that you can get for this program.

So, this is basically the way that the StringBuffer objects can allow to modify the string. Then concatenation is one of the important, concatenation although it is possible for the class string but it always concatenate to a new string actually, but in case of the StringBuffer, it basically concatenate to the same string. So, string will go automatically as you want to have it.

(Refer Slide Time: 18:34)

The slide features a central graphic of a steaming coffee cup on a saucer. To the right of the cup, the title "Merging of Strings" is displayed in a blue font. The background is light blue with faint icons of gears, a tree, and a molecular structure. A small video feed of a man in a light blue shirt is visible in the bottom right corner. At the bottom of the slide, there are logos for NPTEL and the text "NPTEL National Institute of Technology Karnataka, Surathkal" and "Dr. Khuram Khan".

The slide is titled "String merging: concat()". It contains the following text and code:

- You can concatenate two strings using `concat()`, shown here:

```
String concat(String str);
```
- This method creates a new object that contains the invoking string with the contents of `str` appended to the end. `concat()` performs the same function as `+`. For example,

```
String s1 = "one";  
String s2 = s1.concat("two");  
String s1 = "one";  
String s2 = s1 + "two";
```
- You can do the same better with `StringBuffer`, in fact.

Note:
• `s1 = s1 + "two";` it will not work for you as `s1` is immutable.

A small video feed of a man in a light blue shirt is visible in the bottom right corner. At the bottom of the slide, there are logos for NPTEL and the text "NPTEL National Institute of Technology Karnataka, Surathkal" and "Dr. Khuram Khan".

Now here is the few usual concatenation method that you can done, that you can do using the string class object of string class string like as you see, these are the different there are concat method is also possible, but it returns the new string as you can see, this method is different

than the StringBuffer method that it can allow you. So as we see here for the StringBuffer, if s1 is a string object of type StringBuffer then you can do it.

But if these are the string, then you cannot do it, as this is not mutable for the string that is why. So in this case, you will not be able to do, but if you define them, s1 s2 as a StringBuffer, you will be able to do that. So, whatever you can do is in string one and you can declare StringBuffer for them. Also, you will be able to do that. In addition to this, you will be able to do that also.

(Refer Slide Time: 19:54)

The slide displays two code snippets comparing string concatenation methods. The first snippet uses `String.concat()` and the second uses `StringBuffer.append()`. Red annotations highlight the `concat()` and `append()` methods and the `toString()` call in the second snippet.

```
public class ConcatDemo {
    public static String concatWithString() {
        String s = "Java";
        for (int i = 0; i < 10000; i++)
            s.concat("NPTEL");
    }
}

public static String concatWithStringBuffer() {
    StringBuffer sb = new StringBuffer("Java");
    for (int i = 0; i < 10000; i++)
        sb.append("NPTEL");
    return sb.toString();
}
```

Now here are few example that is very interesting to observe it. So this is a concatenation and we are doing concatenation with string object, so it is there and here basically go on concatenating it basically `concat()`(20:07) we call it concatenate return it, but here as we call this concatenate method, this concatenate method is return the concatenated value into the string. So, this string will automatically go by each time in NPTEL string is added and then it 10000 time the string will be added.

But here this is only NPTEL string will be added to this Java and then new string will be created this is a different output you can check the print statement here and if you give the print statement here, then you can understand how the result will be obtained. Now, let us see the difference between the two concatenate method that is there in string and string `concat()`(20:47).

(Refer Slide Time: 20:49)

The slide displays a Java program comparing the performance of string concatenation using `String` and `StringBuffer`. The code is as follows:

```
public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    concatWithString();
    long endTime = System.currentTimeMillis();
    System.out.println("Time taken by Concatenation with String: "
        + (endTime - startTime) + "ms");

    startTime = System.currentTimeMillis();
    concatWithStringBuffer();
    System.out.println("Time taken by Concatenation with StringBuffer: "
        + (endTime - startTime) + "ms");
}
```

The output of the program is:

```
Time taken by Concatenation with String: 578ms
Time taken by Concatenation with StringBuffer: 0ms
```

The slide also features a small video inset of a presenter in the bottom right corner and logos for IIT Bombay and NPTEL at the bottom.

Now, here is that in order to understand the difference, we can write this program and then you can check it. So, here this is the main class we are calling the two methods that we have declared and now, we want to check that how much time that is required for the first concatenation and then, the second concatenation, the second concatenation namely with `StringBuffer` class and the first concatenation namely with `String` class.

So, the two methods are called and time we just calculate, this is one method by which the time can be calculated. So, it is defined in `java.lang`. So, the start time `System.currentTimeMillis()`, it is a system what is called the class define and we can calculate here.

(Refer Slide Time: 21:42)

The slide displays a Java code snippet and its output. The code is as follows:

```
public static void main(String[] args) {
    long startTime = System.currentTimeMillis();
    concatWithStrings();
    System.out.println("Time taken by Concatation with String: "
        + (System.currentTimeMillis()-startTime)+"ms");
    startTime = System.currentTimeMillis();
    concatWithStringBuffer();
    System.out.println("Time taken by Concatation with StringBuffer: "
        + (System.currentTimeMillis()-startTime)+"ms");
}
```

The output shows:

```
Time taken by Concatation with String: 578ms
Time taken by Concatation with StringBuffer: 0ms
```

The slide also features a small video inset of a presenter in the bottom right corner and logos for NPTEL and IIT Madras at the bottom.

We can calculate the starting time at the time of running the program using this method, current time millisecond and then at the end of this call, we just calculate the total time that it required. So, it is basically current time, millisecond and start time and then it gives you basically time. So, now this basically gives you how much time that is required to do this concatenation operation; that mean it will basically call the method 10000 time.

(Refer Slide Time: 22:20)

This slide is identical to the previous one but includes red annotations. In the code, the following lines are underlined in red:

```
long startTime = System.currentTimeMillis();
concatWithStrings();
System.out.println("Time taken by Concatation with String: "
    + (System.currentTimeMillis()-startTime)+"ms");
startTime = System.currentTimeMillis();
concatWithStringBuffer();
System.out.println("Time taken by Concatation with StringBuffer: "
    + (System.currentTimeMillis()-startTime)+"ms");
```

In the output, the first line is underlined in red:

```
Time taken by Concatation with String: 578ms
```

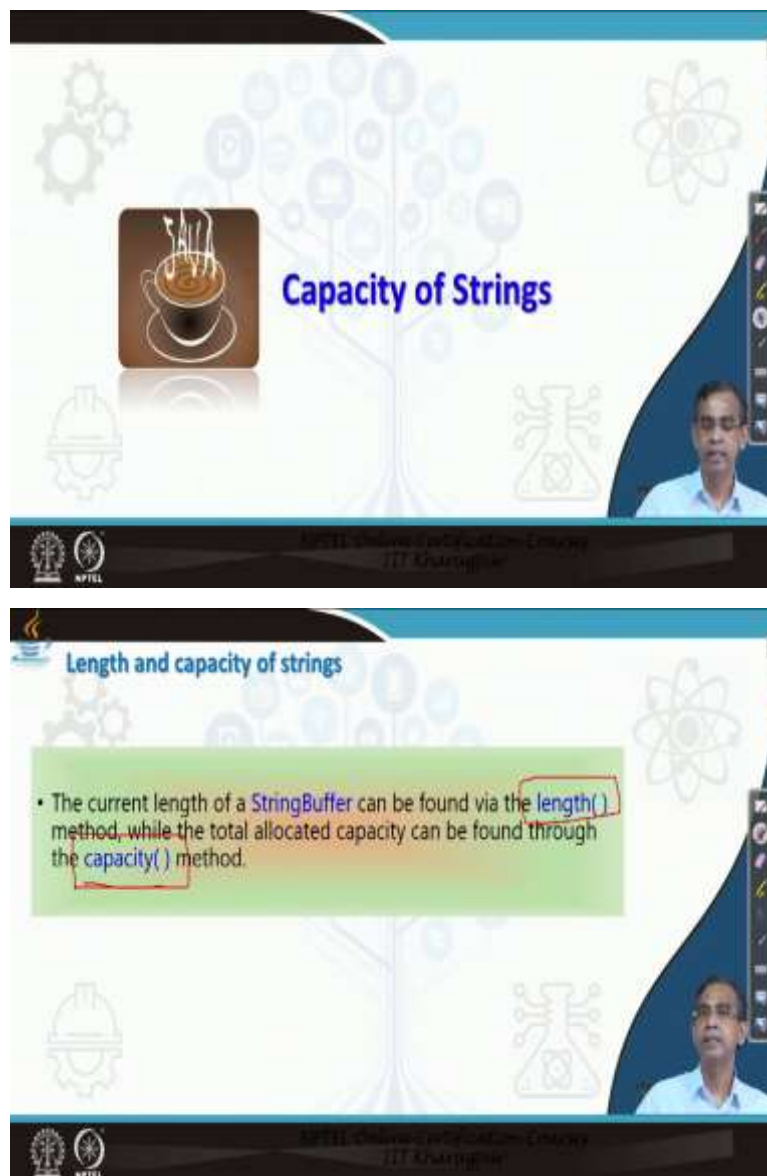
The slide also features a small video inset of a presenter in the bottom right corner and logos for NPTEL and IIT Madras at the bottom.

Now, again alternatively here again if we do the same thing, let us know the current time milliseconds and start time it is always there, this is the start time and then we can call this method again for the StringBuffer object concatenation and calculate what is the time that is

required and you can see the output the two methods will give that the first this method takes around this amount of time.

Whereas this will take this amount of time, this indicates a good realization that StringBuffer object operations are really very fast compared to that usual string plus operation. So, in general you can prefer the StringBuffer class to store your object, because it is faster it gives many more methods to perform any operation and it is also allow you to provide a mutable string generation. So, string buffer is more desirable then the string class we can say.

(Refer Slide Time: 23:13)



Now, let us see the capacity of string that as you see there is a method called capacity that means it will, basically every StringBuffer object associated with a buffer and the capacity of this buffer can be learned from this method StringBuffer and the length method basically

gives you how many characters are there in the current string objects. So, these are the two methods are there.

(Refer Slide Time: 24:00)

The slide displays a Java code snippet and its output. The code defines a class `StringBufferDemo` with a `main` method that creates a `StringBuffer` object, appends the string "Hello", and prints its length and capacity. The output shows the buffer contains "Hello", has a length of 5, and a capacity of 21.

```
// StringBuffer length vs. capacity.
class StringBufferDemo {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer(" ");
        System.out.println("buffer = " + sb);
        System.out.println("length = " + sb.length());
        System.out.println("capacity = " + sb.capacity());
    }
}
```

Output

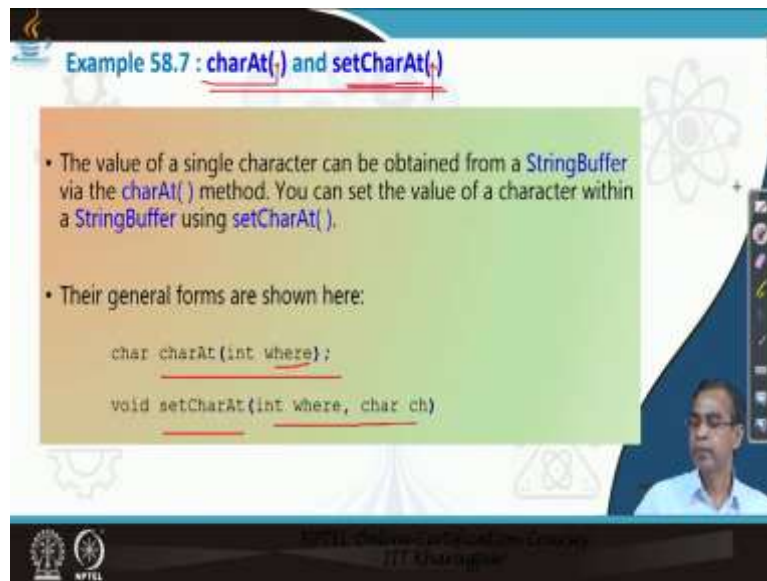
```
buffer = Hello
length = 5
capacity = 21
```

Now, let us have some illustration about the installation of these two methods with the simple program, we create an object of type string buffer name of the object is this one and then we just print the method, the object as we here it will print this one and then length if we print. So, this basically print the length means, how many characters are there it will print there and capacity whenever it is there so it basically 21.

Now, here you see the `StringBuffer` object when you create, it initially it is basically 16 and when you just instantiated with this one, it basically automatically grow that means 16 buffer remain intact and then what ever the additional string is added it automatically grow. So, this basically the, the buffer size at the moment that mean it has that 21 is the buffer.

So it can hold maximum 21 character at the moment. If we add some one thing it automatically grow. So, this is the idea about the `StringBuffer` object and then its capacity and the length the output.

(Refer Slide Time: 25:07)



Example 58.7 : charAt() and setCharAt()

- The value of a single character can be obtained from a `StringBuffer` via the `charAt()` method. You can set the value of a character within a `StringBuffer` using `setCharAt()`.
- Their general forms are shown here:

```
char charAt(int where);  
void setCharAt(int where, char ch)
```

The slide also features a small video inset of a man in the bottom right corner and logos for NPTEL and IIT Madras at the bottom.

Now, so far the modification of this concern, there are two more methods usually very much popular and frequently used the character at and set character at. So, this is basically to retrieve the character at a particular index, where the argument should be passed as an index and set character at it basically, as the index and what is the set of characters that can be passed is basically, some range also can be given, so that a substring of the character also can be obtained.

So, here are the different example as you can see character and where means this is the index position and from two, basically to see that what are set of character that you want to get it. Now let us illustrate this usefulness of these two methods.

(Refer Slide Time: 25:53)

Example 58.5: charAt() and setCharAt()

```
class setCharAtDemo {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        System.out.println("buffer before = " + sb);
        System.out.println("charAt(1) before = " + sb.charAt(1));
        sb.setCharAt(1, 'i');
        System.out.println("buffer after = " + sb);
        System.out.println("charAt(1) after = " + sb.charAt(1));
    }
}
```

Output

```
buffer before = Hello
charAt(1) before = e
buffer after = Hi
charAt(1) after = i
```

And this is an example that you can think about. So, these are demo program regarding the character set and set character at. So as usual this is one StringBuffer object we have created and we print the method there and here character at one. So, in the first location of this one of this one, so it is there.

So, as you see it basically is sb character at first location. So, it basically this is the string before it basically this print and this basically character at one it basically print this one. Now here set character at 1 i you can understand what it does, it basically at 1, it basically replaced by i. So, as we see this is the one output you can get.

Set length 2 that means we can say what is the length of the character at that second position so it basically this one or you can say set character at 2 also you can do. So, instead of length we can do it anyway. So, these are the print method you can see these are the different output that you can get and you can check the program that it can run for you.

(Refer Slide Time: 27:07)

The image contains two screenshots from a video lecture. The top screenshot shows a slide titled "Hash Code of Strings" with a coffee cup icon. The bottom screenshot shows a slide titled "Hash code of a string object" with a list of bullet points explaining the hashCode() method.

Hash Code of Strings

Hash code of a string object

- The `hashCode()` of string returns a unique value for the object. For an object of type `String`, it returns different hash code values, from one object to another.
- On the other hand, it returns the same value irrespective of the contents it contains.

Now, there are, now hash code of a string one important methods, hash code method is defined in object class which basically overwrite any other class. So, you can either override this method for your string objects, otherwise, it will automatically return the default hash code value for the string object. So, this is the idea about let us see some method for which it is defined there in the StringBuffer class.

(Refer Slide Time: 27:39)

The slide displays a Java program and its output. The code defines a class `hashCodeDemo` with a `main` method. It creates a `String` object `str` with the value "NPTEL" and a `StringBuffer` object `str1` with the value "Java NPTEL". It then prints the hash codes for both objects. The output shows that the hash code for `str` is 22941993 and for `str1` is 1486520963, demonstrating that the hash code changes when the string is modified.

```
public class hashCodeDemo {
    public static void main(String arg[]) {
        System.out.println("HashCode test of String");
        String str = "NPTEL";
        StringBuffer str1 = "Java NPTEL";
        System.out.println("HashCode of str: " + str.hashCode());
        System.out.println("HashCode of str1: " + str1.hashCode());

        System.out.println("HashCode test of StringBuffer");
        StringBuffer str2 = new StringBuffer("Java");
        str2.append("NPTEL");
        System.out.println("HashCode of str2: " + str2.hashCode());
    }
}
```

Output

```
HashCode test of String
22941993
17954436
HashCode test of StringBuffer
1486520963
1486520963
```

There is a hash code method is a method, which can be over writeable. Now, here is an example that you can check about hash code method here this program actually show the difference the hash code method for the string class as well as for the StringBuffer class. So, this is the code which, where we apply the hash code to StringBuffer object and this is the code where we apply the hash code to the string object.

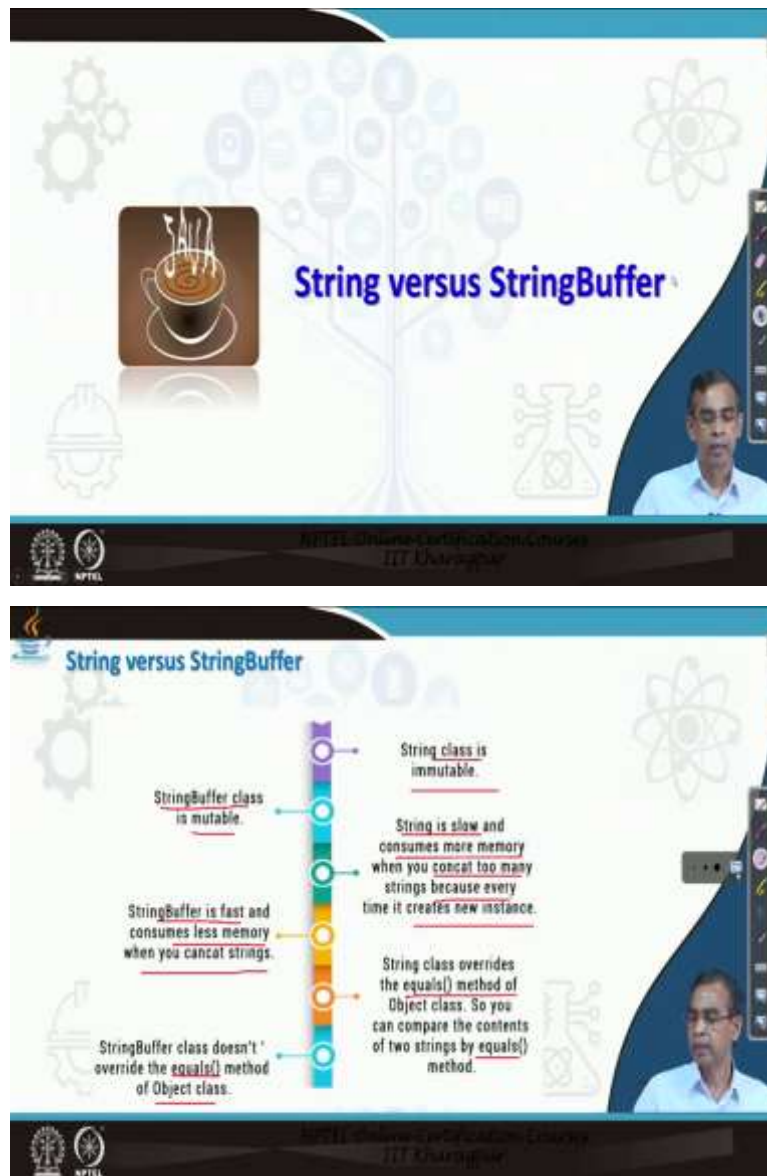
So, here is the initial string, this is the string object is created and we call the hash code method for the string and now here again, this is not possible as it will give an error. So, now if you change it anyway, so this basically change the hash code and it will give it, you can say to this to str 1 and str that is fine then and str1. So, this is the modification that you please do.

So, it is str1 equals to str plus NPTEL and here equals we call str1 hash code. Now if you print it, so this basically this hash code of this Java it basically print this one and you can modify this string, I mean here is basically concatenate the Java NPTEL and then next hash code, you can see it is there. So, that two hash codes gives the different that is possible.

Now let us come to that same hash code generation for the StringBuffer object. So, here we declare one StringBuffer object with this one and here we can have the, hash code as you can see, this is the hash code. Now here append this NPTEL to this string buffer and if we generate the hash code, it basically ream the static, so that it will not produce that the different hash code.

It always produce only hash code of initial StringBuffer object and that will remain same and if you add it, it will not give that different hash code which is basically different than the previous string objects.

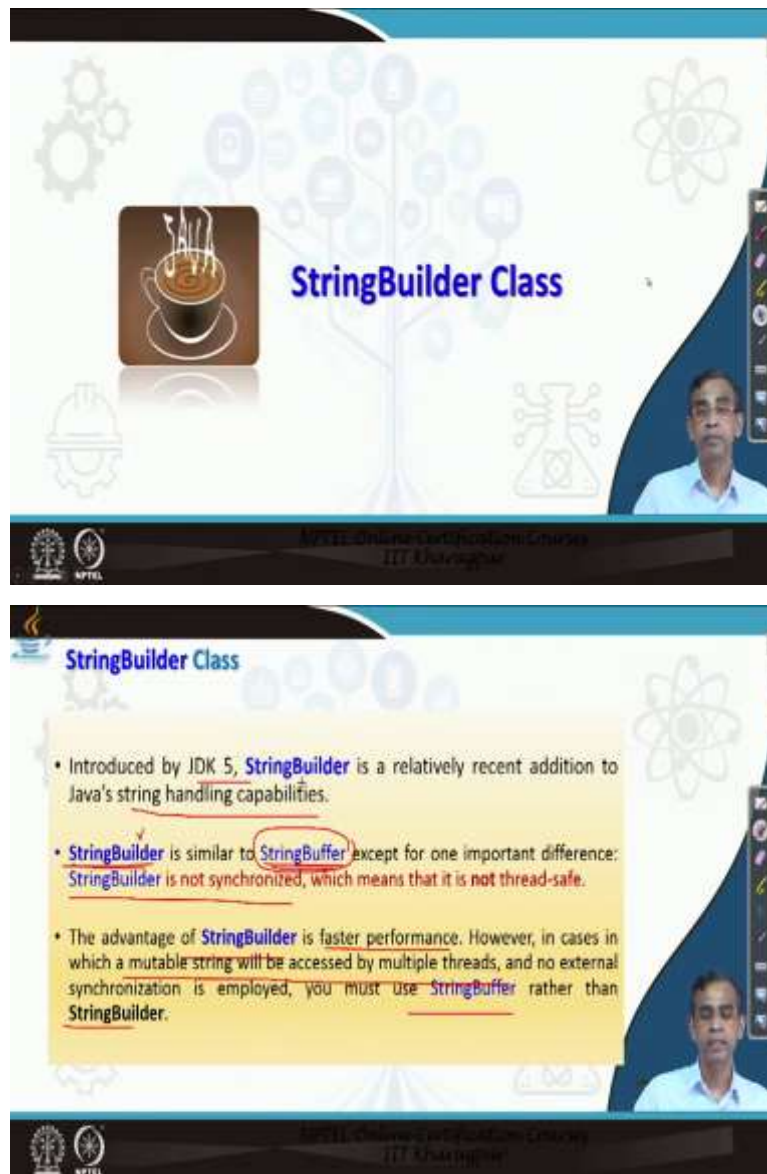
(Refer Slide Time: 29:57)



Now string and now you will just, compare string versus StringBuffer and as we have learned about, so this is related to the StringBuffer, StringBuffer class is mutable where the string class is immutable. String is slow, this is compared to StringBuffer which is fast and it consumes more memory compared to the StringBuffer which consumes less memory and you concat too many strings because every time it creates new instances. So, when you concat string is basically is the same object is modified.

Now String class overrides the equals method that you can do it for object class. So you can compare the content of two strings by equal method. Now here, it does not override the equal method of object class, that is a different that you can verify and writing the program you can check it and so this is basically string versus StringBuffer.

(Refer Slide Time: 30:59)



The image shows two screenshots of a presentation slide titled "StringBuilder Class". The top screenshot shows the title and a coffee cup icon. The bottom screenshot shows a list of bullet points explaining the class.

- Introduced by JDK 5, **StringBuilder** is a relatively recent addition to Java's string handling capabilities.
- **StringBuilder** is similar to **StringBuffer** except for one important difference: **StringBuilder** is not synchronized, which means that it is **not thread-safe**.
- The advantage of **StringBuilder** is faster performance. However, in cases in which a mutable string will be accessed by multiple threads, and no external synchronization is employed, you must use **StringBuffer** rather than **StringBuilder**.

Now, there is another class that is called a StringBuilder class. This StringBuilder class is a recent addition, introduced by JDK 5 and it basically provides more string handling capabilities. In fact, StringBuffer is, StringBuilder is very similar to StringBuffer except there are, one important difference. The differences here that StringBuilder is not synchronized where the StringBuffer is synchronized.

That means for multi-threading application, if you want to use then you can use the StringBuffer, but StringBuilder cannot be used. Otherwise StringBuilder and StringBuffer same. This means the StringBuffer is basically thread safe, while the StringBuilder is not thread safe.

Now here the advantage of string builder is that it is it basically gives more faster performance than StringBuffer. However, in case in which a mutable string will be accessed by multiple threads and no external synchronization is employed, then you must use StringBuffer rather than string builder.

So, this is the purpose by which new Java developer from JDK 5 onwards, they introduced string builder and all the constructors methods are basically same, that of the StringBuffer as well as StringBuilder, so I do not want to repeat them here again.

(Refer Slide Time: 32:29)



So for further study, these are links that you can follow and then you can learn about whatever the example that I have given, you should practice in addition to all the methods that I have mentioned, you can write your program so that you can check it. Thank you. Thank you very much.