STRING HANDLING IN JAVA

Vijay Kumar Meena
Assistant Professor
KIIT University

WHAT WE HAVE COVERED SO FAR?

- **Programming Paradigms**
  - Imperative Paradigms
    - Procedural Programming
    - Structured Programming
    - Object Oriented Programming
  - Declarative Paradigms
    - Functional Programming
    - Logic Programming - Prolog

- **Object Oriented Programming**
  - **Syntax** - Classes, Objects, Attributes, Methods, Constructors, etc.
  - Principles of OOP
    - Encapsulation
    - Abstraction
    - Inheritence
    - Polymorphism

WHAT WE HAVE COVERED SO FAR?

- **Basics of Java**
  - Java Architecture
  - Variables & Data Types
  - Input from user
  - If-else
  - loops
  - Arrays

- **OOP in Java**
  - Java classes
  - Access Modifiers
  - Constructors
  - Inheritence in Java
  - Polymorphism in Java

WHAT WE HAVE COVERED SO FAR?

- Packages in Java
- Abstract class in Java
- Interfaces in Java
- Inner classes

WHAT WE ARE GOING TO STUDY NEXT?

- String Handling in Java
- Exception Handling
- GUI Programming & Event Handling
- Multithreading
- Java Database Connectivity
- Input / Output Stream

STRING HANDLING IN JAVA

WHAT ARE STRINGS IN JAVA?

- A string is a sequence of characters. Most programming languages like C, C++, etc. implement string as a character array but not Java.
- Java implements strings as object of class **String.** One string object represents a sequence of characters.
- When you create a String object, you are creating a string that cannot be changed i.e. Strings are immutable in Java and can’t be changed after initialization.
- To create an mutable string, Java provides **StringBuffer** and **StringBuilder** classes.
- All **String, StringBuffer** and **StringBuilder** classes are declared inside *java.lang* packages and these classes are declared as final which means these classes can’t be inherited by some other class.

HOW TO CREATE STRINGS IN JAVA?

- There are two ways to create String object in Java -
  - By string literal
  - By new keyword
- Using new keyword will invoke the constructor of the String class and create an object.

USING STRING LITERAL

- Java string literal is created by using double quotes. For example, **String name = “abc”;** will create a string literal “abc” and assign address of that literal to variable *name*.
- Each time you create a string literal, the JVM checks the ***string constant pool (Special memory area)*** first. If the string already exists in the pool then a reference to the pooled instance is returned. If the string doesn’t exist in the pool, a new string instance is created and placed in the pool.

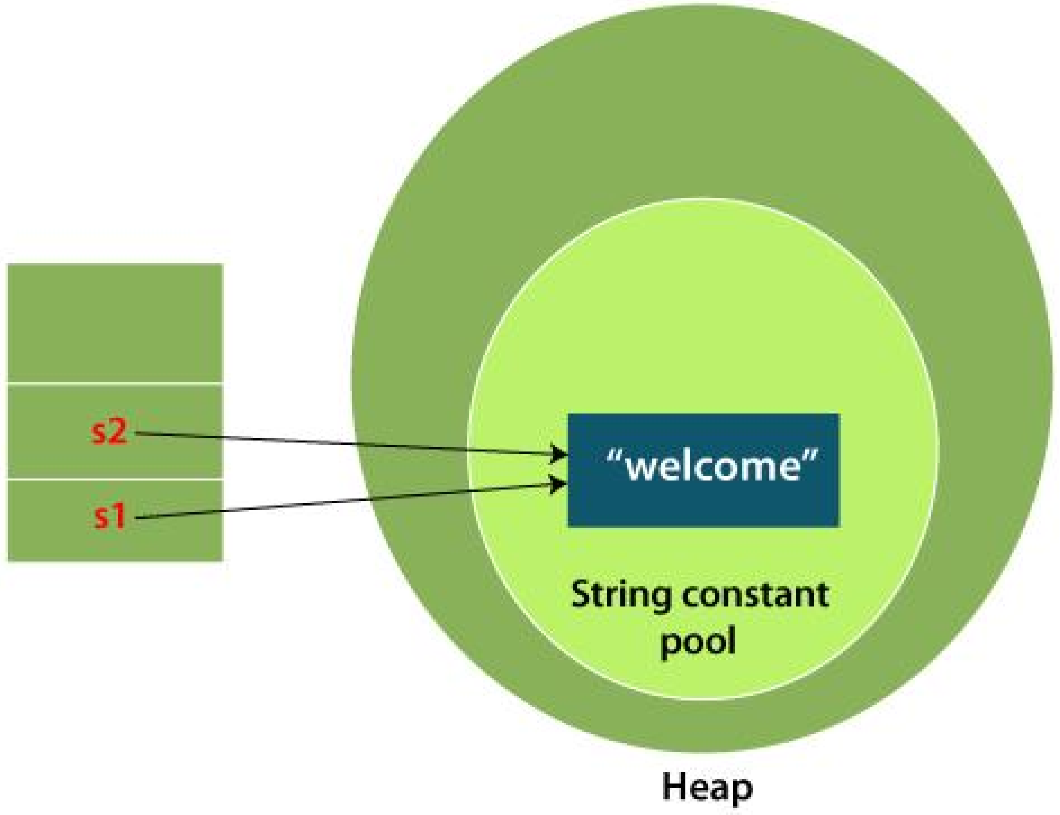
For example -

String s1 = “Welcome”;

String s2 = “Welcome”; // It doesn’t create a new object.

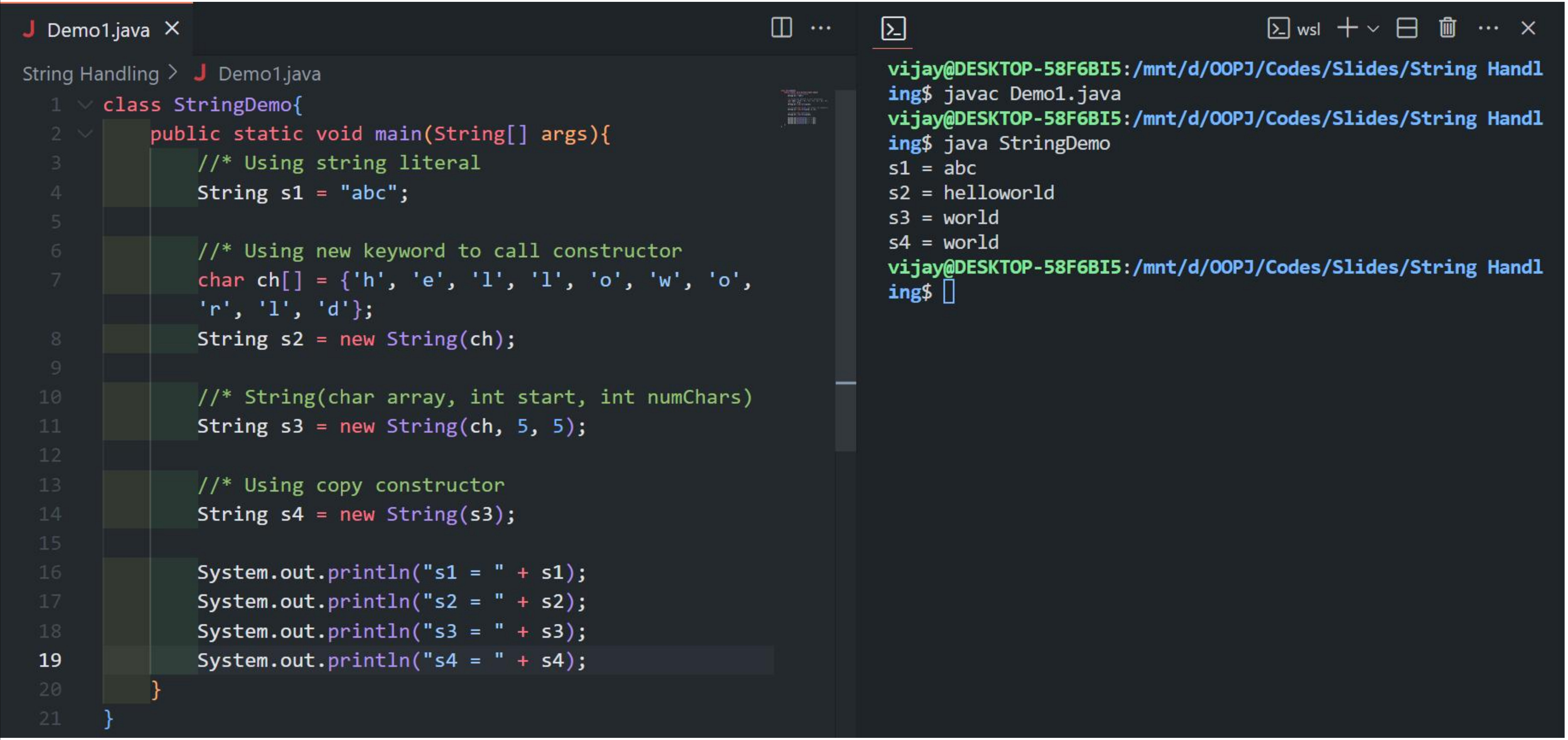
USING STRING LITERAL

- String literals are used to make Java more memory efficient, as no new objects are created if string already exists in the string constant pool.



USING STRING CONSTRUCTORS

- The **String** class supports several constructors.
- **String s = new String();** Calls default constructor and creates an empty string.
- **char chars[] = {‘a’, ‘b’, ‘c’}; String s = new String(chars);** Creates string s with initial value “abc”.
- **String s = new String(“abc”)** will create a string object and will place “abc” in string constant pool also. Here variable **s** will point to the object created not the literal.



Demo1.java

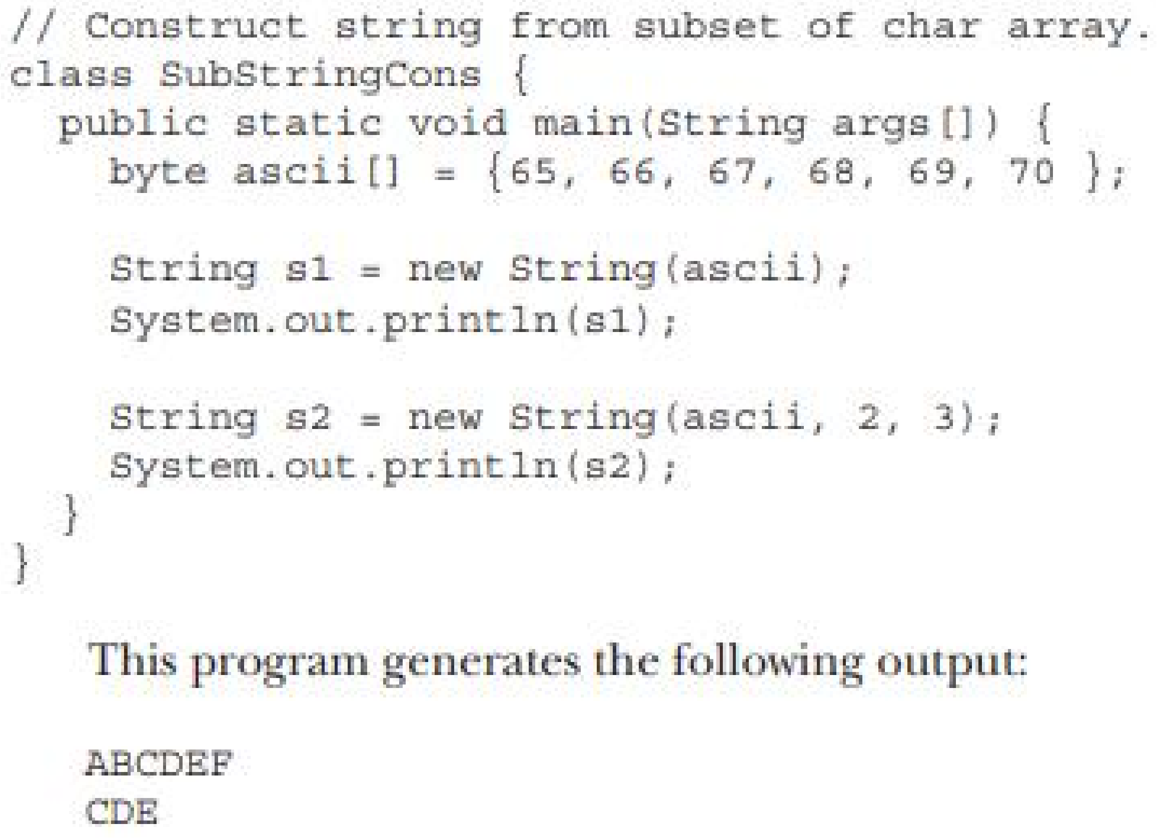
String Handling > Demo1.java

```java
class StringDemo{
    public static void main(String[] args){
        //* Using string literal
        String s1 = "abc";

        //* Using new keyword to call constructor
        char ch[] = {'h', 'e', 'l', 'l', 'o', 'w', 'o',
        'r', 'l', 'd'};
        String s2 = new String(ch);

        //* String(char array, int start, int numChars)
        String s3 = new String(ch, 5, 5);

        //* Using copy constructor
        String s4 = new String(s3);

        System.out.println("s1 = " + s1);
        System.out.println("s2 = " + s2);
        System.out.println("s3 = " + s3);
        System.out.println("s4 = " + s4);
    }
}
```

```
vijay@DESKTOP-58F6BI5:/mnt/d/OOPJ/Codes/Slides/String Handl
ing$ javac Demo1.java
vijay@DESKTOP-58F6BI5:/mnt/d/OOPJ/Codes/Slides/String Handl
ing$ java StringDemo
s1 = abc
s2 = helloworld
s3 = world
s4 = world
vijay@DESKTOP-58F6BI5:/mnt/d/OOPJ/Codes/Slides/String Handl
ing$
```

USING STRING CONSTRUCTORS

- String class also provides ways to create string objects from byte array. You can use **String(byte chrs[]);** or **String(byte chrs[], int startIndex, int numChars);** to create string objects from byte arrays.



```
// Construct string from subset of char array.
class SubStringCons {
  public static void main(String args[]) {
    byte ascii[] = {65, 66, 67, 68, 69, 70 };

    String s1 = new String(ascii);
    System.out.println(s1);

    String s2 = new String(ascii, 2, 3);
    System.out.println(s2);
  }
}
```

This program generates the following output:

```
ABCDEF
CDE
```

STRING CLASS CONSTRUCTORS RECAP

- *String s = new String();* Default constructor
- *String s = new String(char chars[]);*
- *String s = new String(char chars[], int startIndex, int numChars);*
- *String s = new String(String strObj);*
- *String s = new String(byte asciiChars[])*
- *String s = new String(byte asciiChars[], int startIndex, int numChars);*
- *String s = new String(StringBuffer strBufObj);*
- *String s = new String(StringBuilder strBuildObj);*

OPERATIONS ON STRINGS

- You can concatenate multiple strings together using + operator. For example **“Hello”** + “, ”+ **“World”** will return a concatenated string “**Hello, World”.**
- Java can automatically do some type conversions while concatenating two string. For Java example, Java will automatically convert int into string and generate results accordingly.
- For example, **String s = “He is ” + 9 + “ years old.”;** Here int would be automatically converted to string.

EXERCISE

What would be the output of following code?

```
        String s = “four:  ” + 2 + 2;
        System.out.println(s);
```

STRING FUNCTIONS: CHARACTER EXTRACTION

String str = “This is a string literal.”;

- **int length():** Returns the number of characters in string.
  - int len = str.length(); // len = 25
- **char charAt(int n):** Returns the character at index n.
  - char ch = str.charAt(6); // ch = s
- **void getChars(int start, int end, char target[], int targetStart):**
  - char buf[] = new char[10];
  - str.getChars(0, 5, buf, 5); // buf = [‘’, ‘’, ‘’, ‘’, ‘’, ‘T’, ‘h’, ‘i’, ‘s’, ‘’]
- **byte[] getBytes():** Similar to *getChars()* function, it returns bytes instead of chars.
  - byte[] bytes = str.getBytes(); // bytes = [84, 104, 105, 115, 32, ..., 97, 108, 46] ASCII values of each character in string.
- **char[] toCharArray():** Converts string into character array.
  - char[] chars = str.getCharArray(); // chars = [‘T’, ‘h’, ‘i’, ‘s’, ..., ‘a’, ‘l’, ‘.’]

STRING FUNCTIONS: STRING COMPARISON

Consider two strings, String s1 = “Hello”, String s2 = “HELLO”

- **boolean equals(String temp):** Compares the string with string temp
  - boolean match = s1.equals(s2); // match = false as both s1 and s2 are different.
- **boolean equalsIgnoreCase(String temp):** Compares two strings without their case i.e. UpperCase and LowerCase both are treated as the same.
  - boolean match = s1.equalsIgnoreCase(s2); // match = true as both s1 and s2 are same if we don’t consider the case.

STRING COMPARISONS

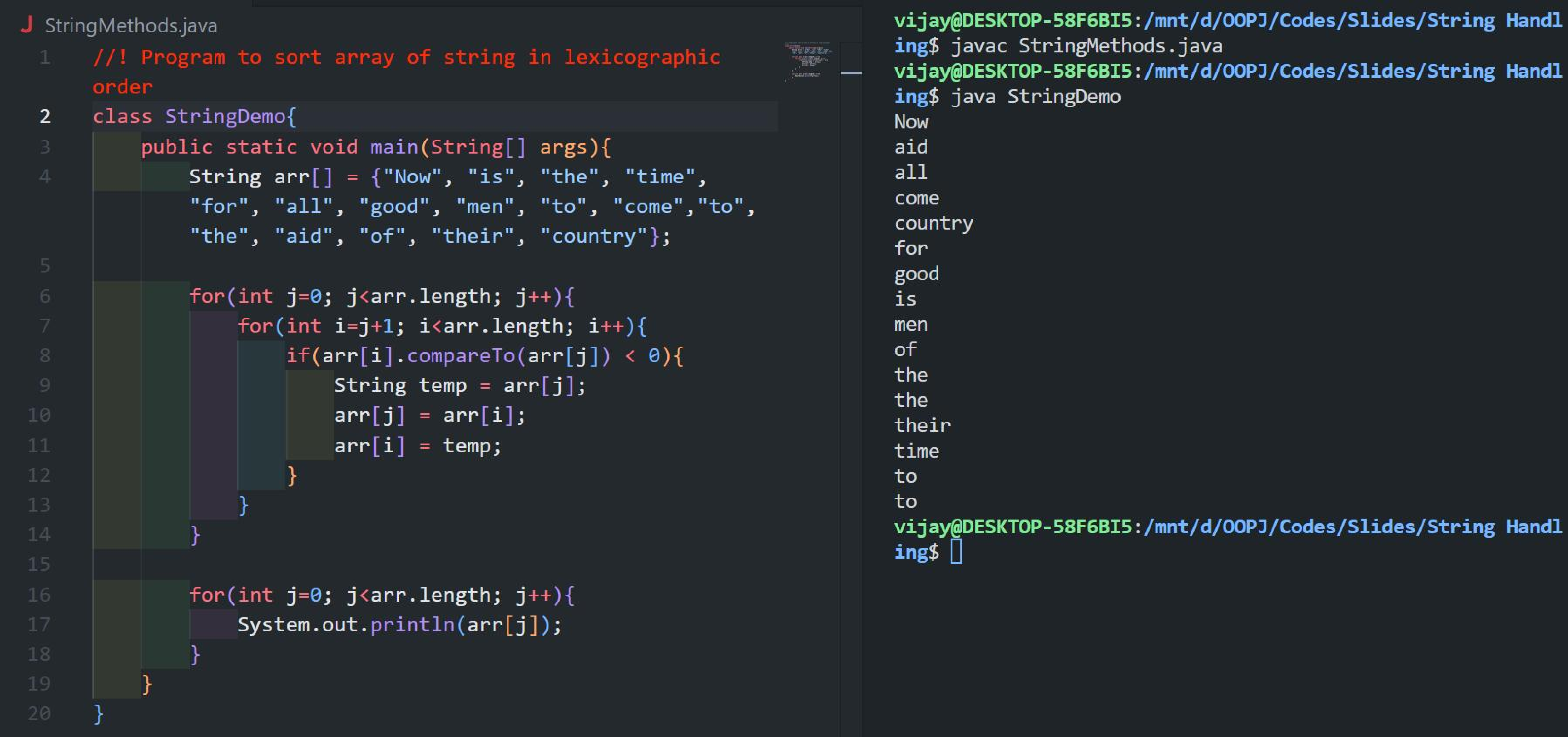
- **regionMatches():** Compares specific part of a string to specific part of another string.
  - **boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)**
  - **boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars)**
  - String s1 = “This is a test”; String s2 = “This can be a TEST”;
  - boolean isSame = s1.regionMatches(10, s2, 14, 4); // isSame = false
  - boolean isSame = s1.regionMatches(true, 10, s2, 14, 4); // isSame = true

STRING COMPARISONS

- **boolean startsWith(String str):** Checks whether the string begins with str.
  - “Foobar”.startsWith(“Foo”); // Returns true
- **boolean startsWith(String str, int index):** Checks whether the substring at index starts with str.
  - “Foobar”.startsWith(“bar”, 3); // Returns true
- **boolean endsWith(String str):** Checks whether the string end with str.
  - “Foobar”.endsWith(“bar”); // Returns true

STRING COMPARISONS

- **equals() vs ==:**
  - equals() compares the characters of two strings.
  - == compares the address of two string objects. It checks whether two variables points to a same object.
- **int compareTo(String str):** Compare two strings
  - int result = s1.compareTo(s2);
  - result < 0 means s1 is less than s2 i.e. s1 comes before s2 in dictionary.
  - result = 0 means s1 and s2 are equal.
  - result > 0 means s1 is greater than s2 i.e. s1 comes after s2 in dictionary.


StringMethods.java

```java
//! Program to sort array of string in lexicographic order
class StringDemo{
    public static void main(String[] args){
        String arr[] = {"Now", "is", "the", "time",
        "for", "all", "good", "men", "to", "come","to",
        "the", "aid", "of", "their", "country"};

        for(int j=0; j<arr.length; j++){
            for(int i=j+1; i<arr.length; i++){
                if(arr[i].compareTo(arr[j]) < 0){
                    String temp = arr[j];
                    arr[j] = arr[i];
                    arr[i] = temp;
                }
            }
        }

        for(int j=0; j<arr.length; j++){
            System.out.println(arr[j]);
        }
    }
}
```

```
vijay@DESKTOP-58F6BI5:/mnt/d/OOPJ/Codes/Slides/String Handl
ing$ javac StringMethods.java
vijay@DESKTOP-58F6BI5:/mnt/d/OOPJ/Codes/Slides/String Handl
ing$ java StringDemo
Now
aid
all
come
country
for
good
is
men
of
the
the
their
time
to
to
vijay@DESKTOP-58F6BI5:/mnt/d/OOPJ/Codes/Slides/String Handl
ing$
```

STRING FUNCTIONS: SEARCHING STRINGS

- **int indexOf(char ch):** Returns the first index of ch in the string.
- **int indexOf(String str):** Returns the first index of str in the string.
  - int index = “This is a test”.indexOf(‘i’); // index = 2
  - int index = “This is a test”.indexOf(“is”); // index = 2
- **int lastIndexOf(char ch):** Returns the last index of ch in the string.
- **int lastIndexOf(String str):** Returns the last index of str in the string.
  - int index = “This is a test”.lastIndexOf(‘s’); // index = 12
  - int index = “This is a test”.lastIndexOf(“is”); // index = 5

STRING FUNCTIONS: 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.
- **String substring(int index);** // Returns the substring starting from index.
  - String s1 = “Hello”.substring(2); // s1 = “llo”
- **String substring(int startIndex, int endIndex);** // Returns the substring starting from startIndex and ending at endIndex.
- **String concat(String str);** // Appends *str* to the string and returns a new string
- **String replace(char *original*, char *replacement*);** // Replaces all occurances of *original* with *replacement*.
- **String trim();** // Removes all leading and trailing whitespaces and returns a new modified string.

STRING FUNCTIONS: MODIFYING STRINGS

- **String toLowerCase();** // Converts string to lowercase and returns new string
- **String toUpperCase();** // Converts string to uppercase
  - String s1 = “Hello”.toLowerCase(); // s1 = “hello”
  - String s2 = “Hello”.toUpperCase(); // s2 = “HELLO”
- **static String valueOf();** // Converts any datatype into string.
  - static String valueOf(double num)
  - static String valueOf(long num)
  - static String valueOf(Object obj)
  - static String valueOf(char chars[])
- **static String join(String delim, String str1, String str2, …);** // Joins all strings separated by given delimeter
  - String s1 = String.join(“, ”, “Alpha”, “Beta”, “Gamma”); // s1 = “Alpha, Beta, Gamma”

STRINGBUFFER CLASS

- A string buffer is like a string but it 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.
- Java StringBuffer class is used to create mutable (modifiable) string.
- Java StringBuffer class is thread-safe i.e. multiple threads cannot access same string buffer simultaneously which allows data to be consistent.

STRINGBUFFER CONSTRUCTORS

- **StringBuffer();** // Creates a string buffer with no characters in it and initial capacity of 16 characters.
- **StringBuffer(int size);** // Creates a string buffer with no characters and specified initial capacity
- **StringBuffer(String str);** // Creates a string buffer intialized with the contents of the given string and reserves room for 16 more characters.
- **StringBuffer(CharSequence seq);** // Creates a string buffer that contains sama characters as the specified *CharSequence*. And reserves room for 16 more characters.
- If more characters are added in string buffer than a new string buffer with larger size would be allocated and content will be copied into new string buffer. This all happens internally in Java so you don't have to worry about it.

STRINGBUFFER FUNCTIONS

- **int length():** Returns number of characters in the string buffer.
- **int capacity():** Returns number of characters which can be stored in string buffer.
  - StringBuffer sb = new StringBuffer("Hello");
  - int len = sb.length(); // len = 5
  - int cap = sb.capacity(); // cap = 21 as constructor will reserve space for 16 more characters.
- **void ensureCapacity(int minCapacity):** Ensures that string buffer created is as of at least size *minCapacity*.
  - StringBuffer sb = new StringBuffer("Hello");
  - sb.ensureCapacity(50);
  - int cap = sb.capacity(); // cap >= 50 as Java might create larger buffer for efficiency.

STRINGBUFFER FUNCTIONS

- **void setLength(int len):** sets the length of string in string buffer. If *len* is less than the actual length of buffer than characters after *len* will be deleted.
- **char charAt(int index):** Returns the character stored at *index*.
- **void setCharAt(int index, char ch):** Updates the character at *index* with updated character *ch*.
- **void getChars(int sourceStart, int sourceEnd, char target[], int targetStart):** Copies characters *between [sourceStart, sourceEnd*) from buffer into target array.

STRINGBUFFER FUNCTIONS

- **StringBuffer append()**: This method concatenates the string representation of any type of data to the end of the invoking StringBuffer object. The string representation of each parameter is obtained, often by calling *String.valueOf()* function from String class.
  - StringBuffer sb = new StringBuffer(40);
  - String s = sb.append(“a = ”).append(30).append(“!”).toString(); // s = “a = 30!”
- **StringBuffer insert(int index, String str):** This method inserts one string into another. It is overloaded to accept values of all primitive types, Strings, Objects, etc.
  - StringBuffer sb = new StringBuffer(“A B”);
  - sb.insert(2, ‘C’); // sb = “A C B”
  - sb.insert(3, “Hello”).insert(8, 5); // sb = “A CHello5B”

STRINGBUFFER FUNCTIONS

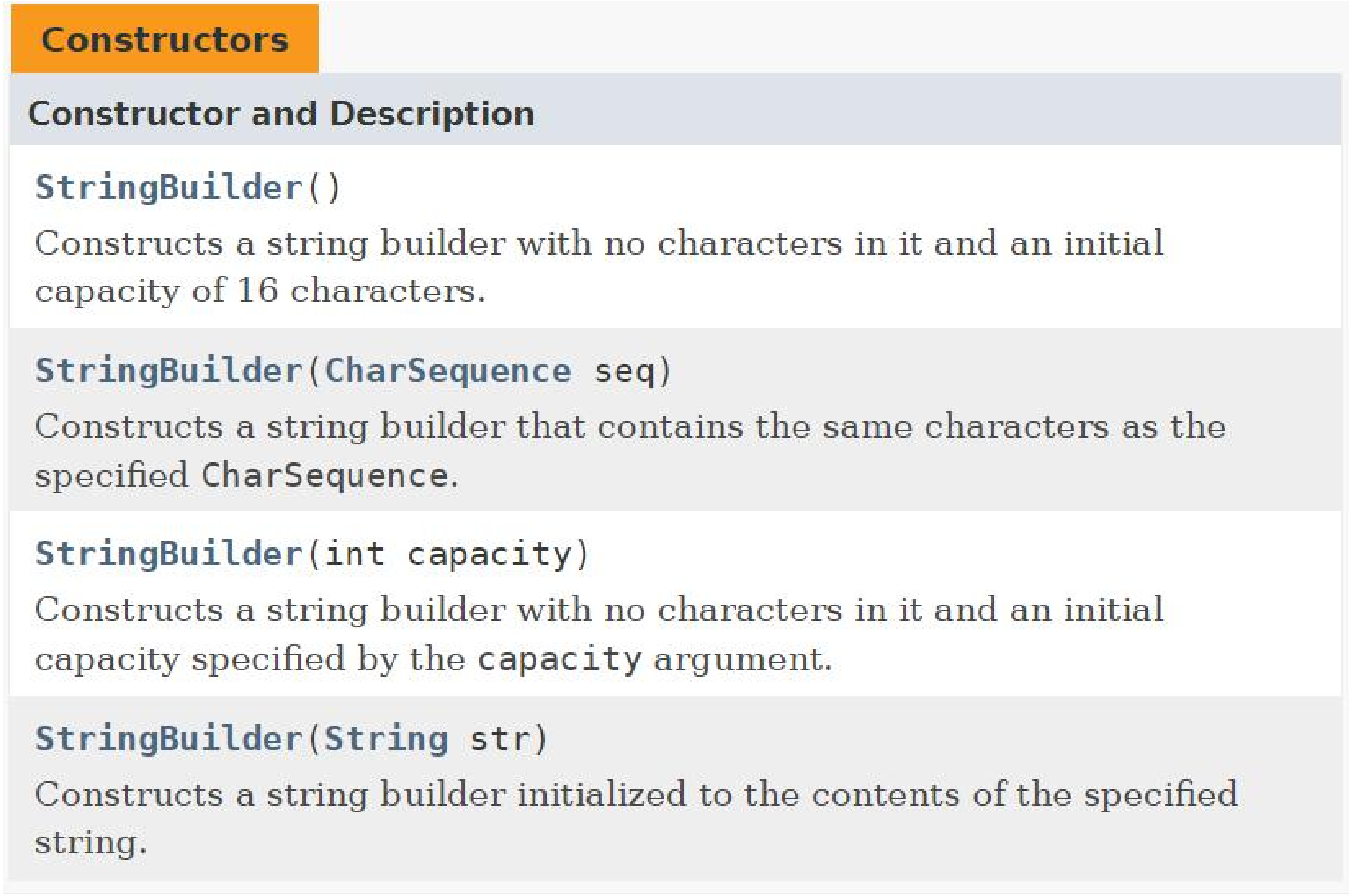
- **StringBuffer reverse():** Reverse the characters within a string buffer.
  - StringBuffer sb = new StringBuffer("abc");
  - sb.reverse(); // sb = "cba"
- **StringBuffer delete(int startIndex, int endIndex):** Deletes characters in range [startIndex, endIndex)
- **StringBuffer deleteCharAt(int index):** Deletes character from particular index
- **StringBuffer replace(int startIndex, int endIndex, String str):** Replace characters in range [startIndex, endIndex) with characters in str.
- **StringBuffer substring(int startIndex):** Returns substring from startIndex
- **StringBuffer substring(int startIndex, int endIndex):** Returns substring made of characters in range [startIndex, endIndex)

STRINGBUILDER CLASS IN JAVA

- StringBuilder is a relatively recent addition to Java's string handling capabilities.
- StringBuilder is similar to StringBuffer i.e. StringBuilder also provides mutable strings in Java.
- Only difference between StringBuilder and StringBuffer is that StringBuffer is thread-safe but StringBuilder is not, which means StringBuilder cannot be used in multiple threading as is might result in data inconsistencies.
- Advantage of StringBuilder is that it is faster compared to StringBuffer. So if your program is performance sensitive then StringBuilder might be better choice compared to StringBuffer.

STRINGBUILDER CONSTRUCTORS

- StringBuilder class have constructors similar to StringBuffer class.



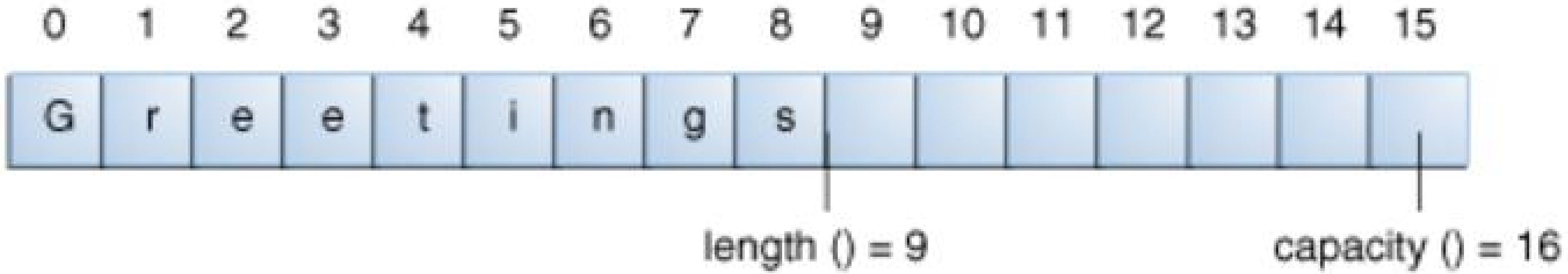
Constructors

Constructor and Description
`StringBuilder()` Constructs a string builder with no characters in it and an initial capacity of 16 characters.
`StringBuilder(CharSequence seq)` Constructs a string builder that contains the same characters as the specified `CharSequence`.
`StringBuilder(int capacity)` Constructs a string builder with no characters in it and an initial capacity specified by the `capacity` argument.
`StringBuilder(String str)` Constructs a string builder initialized to the contents of the specified string.

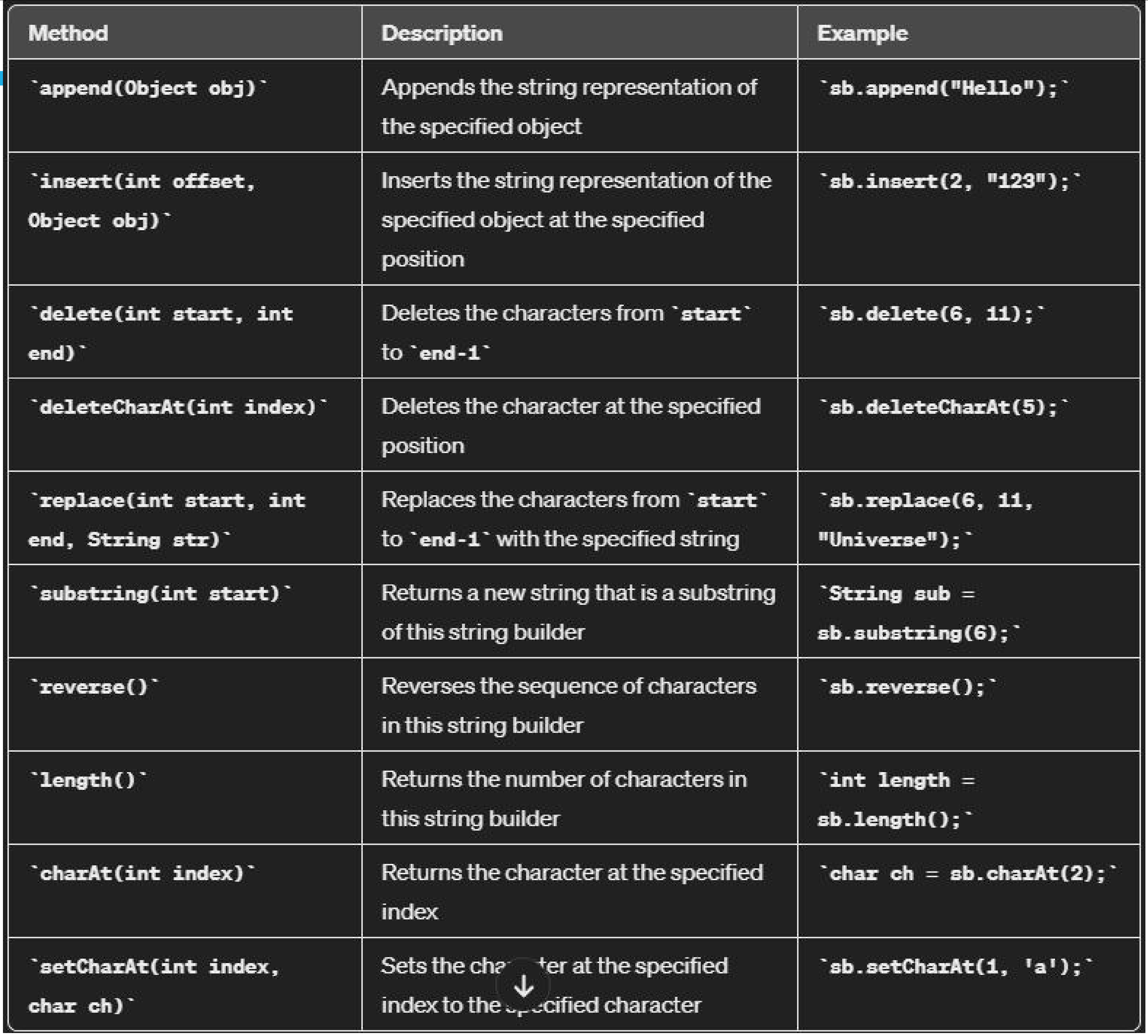
STRINGBUILDER METHODS

Length and Capacity Methods

Method	**Description**
`void setLength(int newLength)`	Sets the length of the character sequence. If `newLength` is less than `length()`, the last characters in the character sequence are truncated. If `newLength` is greater than `length()`, null characters are added at the end of the character sequence.
`void ensureCapacity(int minCapacity)`	Ensures that the capacity is at least equal to the specified minimum.



STRINGBUILDER METHODS



Method	Description	Example
`append(Object obj)`	Appends the string representation of the specified object	`sb.append("Hello");`
`insert(int offset, Object obj)`	Inserts the string representation of the specified object at the specified position	`sb.insert(2, "123");`
`delete(int start, int end)`	Deletes the characters from `start` to `end-1`	`sb.delete(6, 11);`
`deleteCharAt(int index)`	Deletes the character at the specified position	`sb.deleteCharAt(5);`
`replace(int start, int end, String str)`	Replaces the characters from `start` to `end-1` with the specified string	`sb.replace(6, 11, "Universe");`
`substring(int start)`	Returns a new string that is a substring of this string builder	`String sub = sb.substring(6);`
`reverse()`	Reverses the sequence of characters in this string builder	`sb.reverse();`
`length()`	Returns the number of characters in this string builder	`int length = sb.length();`
`charAt(int index)`	Returns the character at the specified index	`char ch = sb.charAt(2);`
`setCharAt(int index, char ch)`	Sets the cha[illegible]ter at the specified index to the [illegible]cified character	`sb.setCharAt(1, 'a');`